Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Жереба Дениса Валерійовича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Вебзастосунок для централізації електронних листів з різних поштових сервісів
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 2101099.01.02.ПЗ

Виконав студент III курсу, група ІПЗс-21-1 _____ Денис, ЖЕРЕБ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник д-р фіз.-мат. наук, проф. _____ Леонід, БЕДРАТЮК
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. пед. наук, доцент _____ Наталія, ПРАВОРСЬКА
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

6 червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Жереб Денис Валерійович
Прізвище, ім'я, по батькові студента

1. Тема роботи Вебзастосунок для централізації електронних листів з різних поштових сервісів

Керівник роботи Бедратюк Л. П. д-р фіз.-мат. наук, професор
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 08.01.2024 р. № 20

2. Строк подання студентом роботи на кафедру 01.06.2024 р.

3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація та тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Чотири креслення:

1. Діаграма варіантів використання
2. Діаграма послідовності для варіанту використання перегляд тікетів
3. Діаграма послідовності для варіанту використання авторизація
4. Схема даних

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Праворська Н. І., канд. пед. наук, доцент.	05.06.24	05.06.24
Антиплагіат	Форкун Ю. В., доцент	06.06.24	06.06.24

7. Дата видачі завдання «_02_» _січня_ 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Збір матеріалу за темою кваліфікаційної роботи (КвР); дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2024	
2 Проектування програмного забезпечення	21.02 – 20.03 2024	
3 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2024	
4 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2024	
5 Попередній захист КвР	травень 2024	
6 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2024	
7 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2024	

Студент _____ Денис, ЖЕРЕБ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Леонід, БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Вебзастосунок для централізації електронних листів з різних поштових сервісів».

Автор роботи: Жереб Денис Валерійович

Керівник роботи: Бедратюк Леонід Петрович

Пояснювальна записка: 102 с., 30 рис., 2 таб., 2 дод., 20 джерел.

Графічна частина: 4 креслення.

ВЕБЗАСТОСУНОК, ПОШТОВИЙ СЕРВІС, ЕЛЕКТРОНИЙ ЛИСТ, JAVASCRIPT, REACT, REDUX, C#, SQP NET CORE API, SQL Server, EF CORE.

Метою роботи є створення вебзастосунку для централізації електронних листів з різних поштових сервісів.

У кваліфікаційній роботі проведено аналіз предметної області та її інформаційного забезпечення, визначені вимоги до вебзастусунку з централізації листів, розроблена загальна архітектура додатку, спроектована структура бази даних та структура додатку.

Для розробки програмної системи використано мови програмування Js та C#, бібліотека React, фреймворк ASP Net Core API, сервер бази даних SQL Server.

У результаті проектування здійснена програмна реалізація застосунку для централізації електронних листів з різних поштових сервісів.

Вебзастосунок призначений для користувачів, які працюють з великою кількістю електронних листів на різних платформах.

06.06.2024
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Приміт ка
			Текстові документи			
1	А4	КвРІПЗ. 2101099.01.02.ПЗ	Пояснювальна записка	82		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
4	А4		Код програми	18		
5	А4		Презентаційний матеріал	9		
			Графічні документи			
6	А3	КвРІПЗ. 2101099.01.02.Е8	Діаграма варіантів	1		
			використання			
7	А3	КвРІПЗ. 2101099.01.02.Е8	Діаграма послідовності	1		
			для варіанту використа-			
			ння перегляд тікетів			
8	А3	КвРІПЗ. 2101099.01.02.Е8	Діаграма послідовності	1		
			для варіанту використа-			
			ння авторизація			
9	А3	КвРІПЗ. 2101099.01.02.Е8	Схема даних	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 2101099.01.02.ВД		
Виконав		Жереб Д. В.		06.06	Вебзастосунок для централізації електронних листів з різних поштових сервісів	Літ.	Арк. 1 / Аркушів 1
Керівник		Бедратюк Л.П.		06.06			
Рецензент		Говорущенко Т. О.		06.06			
Н. контр.		Праворська Н. В.		06.06	Відомість документів	ХНУ, ІПЗс-21-1	
Зав. каф.		Бедратюк Л.П.		06.06			

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

АРМ	–	автоматизоване робоче місце
БД	–	база даних
ЕОМ	–	електронно-обчислювальна машина
ЕС	–	експертна система
ІС	–	інформаційна система
КС	–	комп’ютерна система
ООП	–	об’єктно-орієнтоване програмування
ПЗ	–	програмне забезпечення
ПП	–	програмний продукт
СУБД	–	система управління базами даних
ШІ	–	штучний інтелект
API	–	Application programming interface
CRM	–	Change Request Management
DBS	–	Data Base Server
FTP	–	File Transfer Protocol
HTTP	–	HyperText Transfer Protocol
UML		Unified Modeling Language
XML	–	eXtensible Markup Language
SLA	–	A service-level agreement

ВСТУП

Електронна пошта, відома також як e-mail, стала невід'ємною частиною нашого повсякденного життя і відіграє ключову роль як у сфері бізнесу, так і у особистому користуванні.

У сучасному світі велика кількість користувачів стикається з проблемою управління численними обліковими записами електронної пошти на різних платформах. Це ускладнює процес комунікації, призводить до втрати важливих повідомлень та знижує ефективність роботи як у бізнесі, так і в особистому житті. Відсутність централізованого підходу до обробки електронної пошти створює низку проблем, серед яких:

– розпорошеність інформації: Користувачі часто мають кілька електронних скриньок на різних сервісах, що ускладнює пошук необхідних листів та інформації;

– зниження продуктивності: Необхідність перевіряти кілька поштових скриньок витрачає час і зусилля, які могли б бути використані більш продуктивно;

– відсутність інтеграції: Неможливість інтегрувати електронну пошту з іншими інструментами для управління завданнями та календарем, що ускладнює планування та організацію роботи;

– безпека та конфіденційність: Користувачі змушені вводити свої облікові дані на різних платформах, що може підвищувати ризики порушення безпеки та конфіденційності.

Таким чином, основною проблемою є необхідність створення єдиного централізованого сервісу для управління електронною поштою, який би об’єднав різні поштові скриньки та надав інструменти для ефективної організації процесу комунікації.

Такий сервіс повинен забезпечити користувачам можливість швидко та зручно доступати до всіх своїх електронних листів з одного місця, підвищуючи таким чином їх продуктивність та покращуючи організацію роботи.

Отже, централізація електронних листів з різних поштових скриньок на різних платформах є кроком у майбутнє, який відповідає вимогам сучасного світу.

Об'єктом дослідження є сервіси, які надають доступ до електронних скриньок для онлайн листування та організації процесу комунікації.

Предметом дослідження даної кваліфікаційної роботи є програмне забезпечення (ПЗ), яке спрощує керування електронною поштою шляхом об'єднання різних поштових скриньок у централізовану систему.

Головною метою під час виконання кваліфікаційної роботи є створення вебзастосунку з функціоналом для централізації електронних листів з різних поштових сервісів. Даний вебзастосунок дозволить користувачам об'єднувати свої різні електронні поштові скриньки в одній системі, забезпечуючи легкий доступ до всіх листів через єдиний інтерфейс. Крім того, за допомогою даного вебзастосунку буде доступна можливість аналізу статистики використання електронної пошти, що дозволить користувачам отримати детальний аналіз свого користування поштовими сервісами.

Шлях до досягнення мети даної кваліфікаційної роботи є вирішення наступних задач:

– провести змістовний аналіз предметної області, її структурних та функціональних особливостей;

– провести аналіз наявних програмних рішень;

– ґрунтуючись на основі первиного аналізу постановити вимоги до ПЗ, щоб обрати найкращі та найефективніші технології і підходи для розробки;

– створення необхідних UML діаграм для візуальної наглядності структури та архітектури ПЗ;

– розробка ПЗ відповідно до поставлених вимог та створених діаграм;

– тестування створеного продукту.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовий аналіз предметної області, її структурних та функціональних особливостей

Електронна пошта [1] — це спосіб обміну повідомленнями, які зберігаються в комп’ютері, від одного користувача до одного чи кількох одержувачів через Інтернет. Електронні листи – це швидкий, недорогий і доступний спосіб спілкування для бізнесу чи особистого використання. Користувачі можуть надсилати електронні листи з будь-якого місця, якщо у них є підключення до Інтернету, яке зазвичай надається постачальником послуг Інтернету.

Обмін електронною поштою здійснюється через комп’ютерні мережі, переважно в Інтернеті, але нею також можна обмінюватися як між загальнодоступними, так і між приватними мережами, такими як локальна мережа. Електронну пошту можна розсилати як спискам людей, так і окремим особам. Спільним списком розсилки можна керувати за допомогою відбивача електронної пошти. Деякі списки розсилки дозволяють користувачам підписатися, надіславши запит адміністратору списку розсилки. Список розсилки, який адмініструється автоматично, називається сервером списку.

Повідомлення електронної пошти зазвичай кодуються у форматі американського стандартного коду для обміну інформацією (ASCII). Однак користувачі також можуть надсилати нетекстові файли, такі як графічні зображення та звукові файли, як вкладені файли. Електронна пошта була однією з перших видів діяльності в Інтернеті, і досі залишається найпопулярнішою. Велику частку загального трафіку в Інтернеті становить електронна пошта.

Нижче наведено найпоширеніші випадки використання електронної пошти [2].

Індивідуальне чи групове спілкування. Електронна пошта – це зручний спосіб спілкування з окремими людьми або невеликими групами друзів чи колег. Це дозволяє користувачам легко надсилати та отримувати документи, зображення, посилання та інші файли. Це також дає користувачам можливість

спілкуватися з іншими за власним розкладом.

Сповіщення, нагадування та подальші дії. Окремі особи, невеликі групи чи організації можуть надсилати подальші електронні листи після зустрічей, зустрічей чи співбесід або нагадувати учасникам про наближення подій, термінів виконання та термінових заходів. Календарі з можливістю додавати зустрічі та події інтегровані в більшість платформ електронної пошти. Ці функції допомагають користувачам керувати часом, візуалізуючи фіксовану кількість часу, який вони мають на день. Це дозволяє користувачам розставляти пріоритети своїх рішень і часу.

Донесення інформації до великої групи людей. Компанії можуть використовувати електронну пошту для передачі інформації великій кількості співробітників, клієнтів і потенційних клієнтів. Електронна пошта часто використовується для інформаційних бюлетенів, де передплатникам списків розсилки надсилається конкретний рекламований вміст від компанії та прямих маркетингових кампаній електронною поштою, де реклама чи реклама надсилаються цільовій групі клієнтів.

Перетворення потенційних клієнтів на клієнтів, які платять. Електронна пошта також може бути використана для перетворення потенційного продажу в завершену покупку. Наприклад, компанія може створити автоматичний електронний лист, який надсилатиметься онлайн-покупцям, які зберігають товари у кошику для покупок певний час. Електронний лист може нагадувати клієнту, що в кошику є продукти, і заохочувати його завершити покупку до того, як товар закінчиться.

Огляди та опитування. Подальші листи електронної пошти з проханням надіслати відгук після покупки можуть містити опитування з проханням оцінити якість послуг або продукт, який вони нещодавно отримали.

Ось деякі приклади популярних безкоштовних веб-сайтів електронної пошти [3].

Gmail – це безкоштовна служба електронної пошти від Google. Gmail також пропонує платні плани для бізнес-користувачів, які включають додатковий обсяг пам’яті, розширені функції та параметри підтримки. За даними Litmus в оновленому звіті «Частка ринку поштових клієнтів за липень 2022 року», станом

на квітень 2022 року Gmail займає 29,5% частки ринку поштових клієнтів.

Microsoft Outlook доступний як частина пакету Microsoft Office і пропонує як безкоштовні, так і платні версії. Microsoft Outlook працює на кількох операційних системах і пристроях і надає такі функції, як відновлення видаленої електронної пошти та автоматична організація електронної пошти.

Yahoo Mail була запущена в 1997 році і є одним із найстаріших клієнтів веб-пошти. Yahoo Mail корисний для особистих електронних листів і доступний у версії мобільного додатка.

AOL Mail була однією з найпопулярніших служб електронної пошти в минулому, а тепер є частиною Verizon Communications. Він пропонує необмежений розмір поштової скриньки та дозволяє користувачам пов'язувати свою пошту AOL з іншими обліковими записами електронної пошти, такими як Outlook і Gmail.

Zoho Mail був запущений у 2008 році та є частиною Zoho Office Suite. Цей клієнт електронної пошти забезпечує високий рівень безпеки та доступні плани як для особистого, так і для ділового використання. Згідно з опитуванням, проведеним Zoho, у 2020 році у нього було 15 мільйонів користувачів у всьому світі.

Попри велику кількість доступних поштових клієнтів і сервісів електронної пошти, існує низка проблем і невирішених питань, які можуть значно ускладнювати користування електронною поштою. Серед них:

- фрагментація інформації. Користувачі часто мають кілька облікових записів електронної пошти на різних платформах, що призводить до фрагментації інформації і ускладнює її управління;
- відсутність єдиного інтерфейсу для доступу до всіх електронних скриньок одночасно;
- часові витрати. Перевірка та управління кількома обліковими записами електронної пошти на різних платформах вимагає значних часових витрат. Перемикання між різними платформами і їх опанування займає додатковий час;
- відсутність інтеграції з інструментами для управління завданнями, календарями та іншими бізнес-додатками. Обмежені можливості для

автоматизації робочих процесів, що ускладнює організацію роботи;

– безпека та конфіденційність. Необхідність введення облікових даних на різних платформах може підвищувати ризики порушення безпеки та конфіденційності;

– відсутність єдиних стандартів безпеки для управління всіма обліковими записами електронної пошти.

Основною цільовою аудиторією розроблюваного програмного забезпечення є люди віком від 16 до 60 років, які активно використовують електронне листування в бізнесі та особистому житті. Вони мають такі інформаційні потреби:

– централізоване управління електронною поштою;

– можливість доступу до всіх облікових записів електронної пошти з одного інтерфейсу;

– зручний пошук і управління електронними листами незалежно від платформи;

– підвищення продуктивності. інструменти для автоматизації рутинних завдань, таких як сортування електронної пошти та нагадування про важливі події;

– єдиний стандарт безпеки для всіх облікових записів електронної пошти;

– функції ідентифікації та аутентифікації користувачів для захисту особистих даних.

Розроблюване програмне забезпечення повинно забезпечити процес ефективного перетворення вхідної інформації у вихідну. Це включає:

– об'єднання електронних скриньок. Збір електронних листів з різних платформ та об'єднання їх у єдиному інтерфейсі;

– фільтрація та сортування. Автоматична фільтрація та сортування електронних листів за важливістю, датою, відправником тощо.

Ідентифікація та аутентифікація користувачів:

– використання двофакторної аутентифікації для захисту облікових записів;

– регулярне оновлення паролів та використання унікальних паролів для різних облікових записів;

– зберігання даних у зашифрованому вигляді на серверах;

– обмеження доступу до конфіденційної інформації лише авторизованим користувачам.

1.2 Аналіз наявного програмно забезпечення предметної області

Програмне забезпечення LiveAgent (рисунок 1.1) зазвичай фіксує всі вхідні повідомлення від клієнтів і перетворює їх на тікети для полегшення керування. Клієнти використовують усілякі способи зв'язку, як-от електронну пошту, телефон, месенджери чи чат, тож збереження всього спілкування в одному місці допоможе вашим агентам із обслуговування клієнтів краще зрозуміти проблему та швидше відповісти [4].

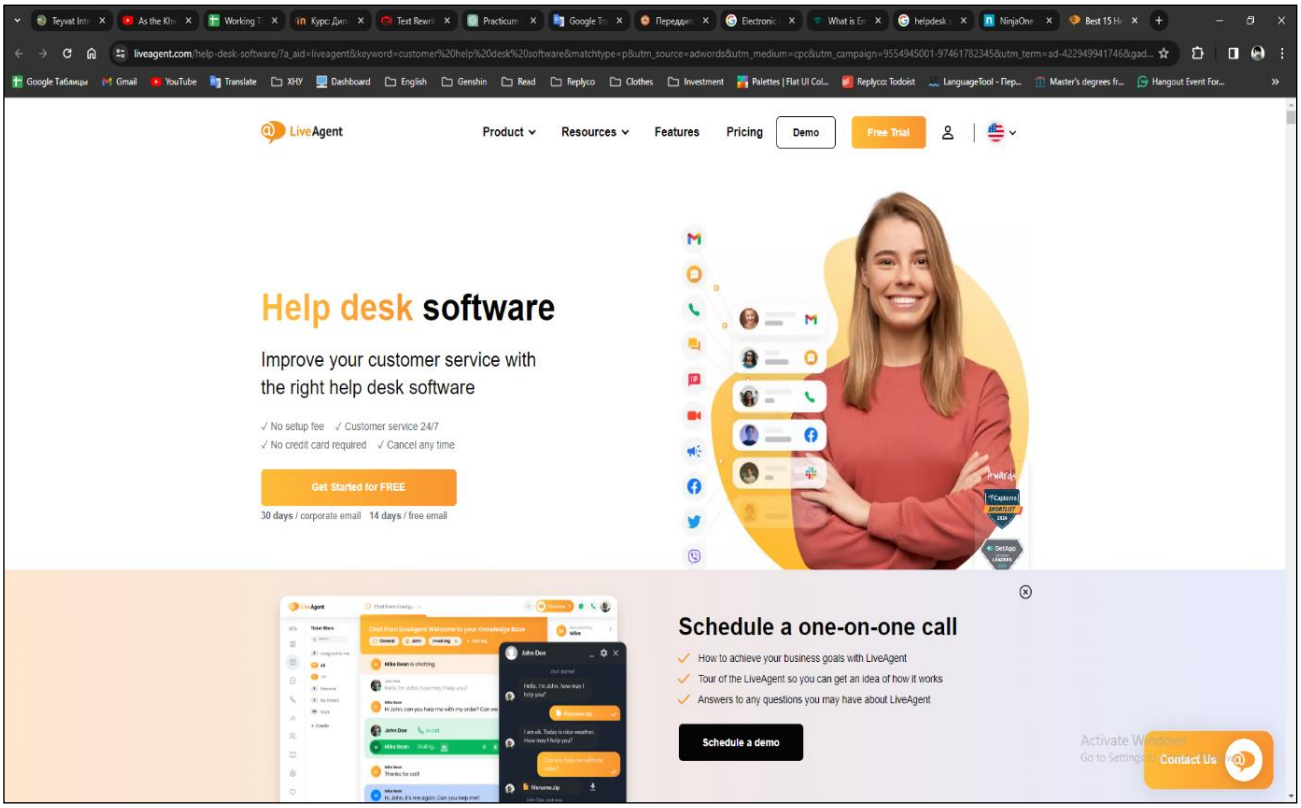


Рисунок 1.1 – Вебзастосунок LiveAgent

Будь-яка компанія, яка пропонує підтримку клієнтів або обслуговування клієнтів, може використовувати програмне забезпечення LiveAgent для легкої обробки запитів клієнтів. Відповідальність лежить на групах підтримки клієнтів, які є основними користувачами програмного забезпечення служби підтримки. LiveAgent часто мають розширені функції, щоб розширити їх корисність для інших відділів.

Команда продажів може скористатися можливостями кол-центру або живого чату, тоді як служба IT-сервісу може використовувати електронну пошту та портал для клієнтів із базою знань.

Використовуючи безкоштовне програмне забезпечення бази знань, ваші агенти можуть витрачати менше часу на відповіді на повторювані запитання та більше зосереджуватися на критичних питаннях.

Однією з головних переваг LiveAgent є підвищення ефективності роботи завдяки багатоканальному рішенню. Це програмне забезпечення дозволяє обробляти повідомлення з різних каналів зв’язку, таких як електронна пошта, телефон, месенджери та чат, що значно спрощує роботу агентів. Усі комунікації з клієнтами зберігаються в одному місці, що дозволяє агентам швидко знаходити необхідну інформацію та оперативно реагувати на запити. Крім того, LiveAgent має розширені функції, які дозволяють інтегрувати інші відділи компанії, такі як відділ продажів або IT-сервісу. Використання бази знань значно скорочує час на відповіді на повторювані запитання, що дає агентам можливість зосередитися на вирішенні більш складних проблем.

Серед недоліків LiveAgent варто відзначити відсутність деяких популярних платформ для мейлінгу, що може обмежити можливості інтеграції з іншими системами. Висока вартість програмного забезпечення також може стати перешкодою для невеликих компаній або стартапів.

Крім того, короткий випробувальний термін не дає достатньо часу для повного оцінювання всіх можливостей та функцій програми. Інколи користувачі

також зазначають про складність налаштування та освоєння всіх доступних функцій, що може вимагати додаткового часу та ресурсів для навчання персоналу.

NINJAONE TICKETING - це платформа служби підтримки електронної комерції, яка централізує всі повідомлення ваших клієнтів в одній інтуїтивно зрозумілій скриньці вхідних повідомлень. На рисунку 1.2 головна сторінка сервісу. Допомагає автоматизувати завдання, оптимізувати робочі процеси та надає виняткову підтримку клієнтів за частку часу [6].

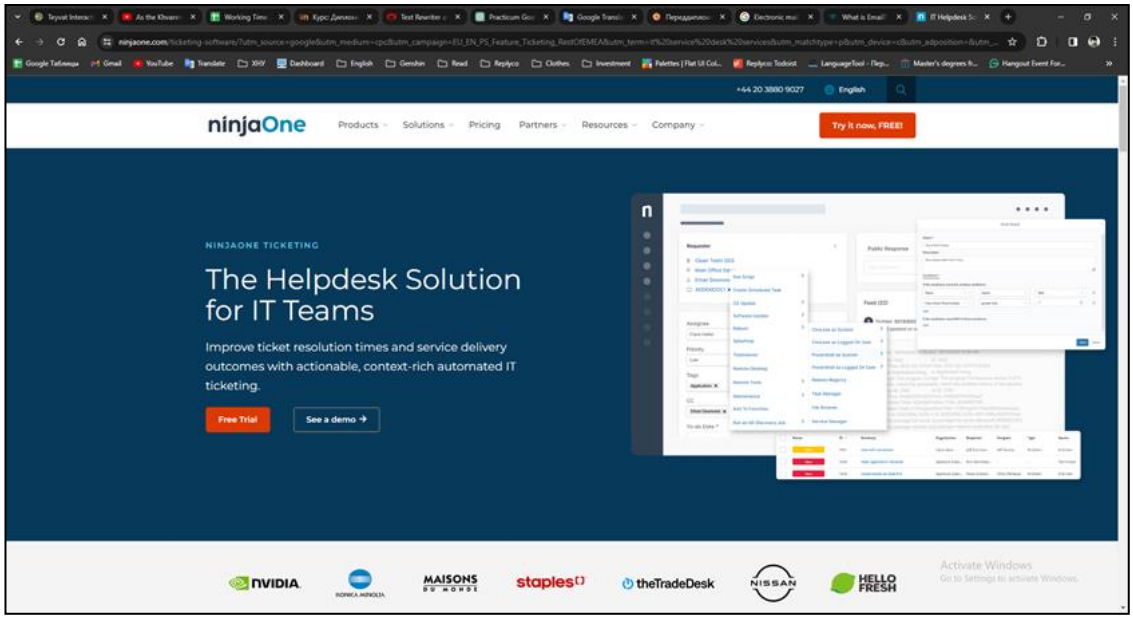


Рисунок 1.2 – Сайт NINJAONE TICKETING

Попередньо відформатовані шаблони відповідей NINJAONE TICKETING у поєднанні з тегами даних клієнтів, інтелектуальними автовідповідачами та автоматизацією логістики дозволяють зв'язуватися з клієнтами, навіть коли офіс закритий на день. Усуває повторення, передбачивши найчастіші запитання ваших покупців і відповідаючи за допомогою попередньо відформатованого або спеціального шаблону електронного листа. Шаблонні відповіді дають змогу відповісти за частку часу, який зазвичай потрібен для вирішення типових тем, як-от стан доставки, відстеження посилок, повернення та запити на зміну адреси. Автовідповідачі з готовими шаблонами надають повний контроль над тригерами автоматичної відповіді, такими як ключове слово, ринок і домен електронної

пошти, тож можна бути впевненим, що в кожній ситуації буде надіслано відповідну автоматичну відповідь. Навіть можна встановити автоматичні відповіді на затримку, щоб збільшити ймовірність того, що ваша відповідь буде зарахована до показників відповіді.

Має можливість автоматично заповнювати електронні листи та шаблони відповідними даними клієнтів і замовлень для персоналізованої відповіді щоразу. Завдяки інтегрованим ринковим даним готові відповіді не є загальними, а специфічними для кожного клієнта за допомогою таких тегів, як «customer_first_name», «order_id» і багато іншого.

NINJAONE TICKETING має свої унікальні переваги. Платформа дозволяє централізувати всі повідомлення клієнтів в одній інтуїтивно зрозумілій скриньці вхідних повідомлень, що значно спрощує процес обробки запитів. Серед основних переваг варто відзначити автоматизацію завдань та оптимізацію робочих процесів. Використання попередньо відформатованих шаблонів відповідей, інтелектуальних автовідповідачів та автоматизації логістики дозволяє зменшити час на обробку типових запитів, таких як стан доставки або відстеження посилок. Наявність мобільного додатку дозволяє агентам працювати з будь-якого місця, що підвищує гнучкість та мобільність.

Проте, NINJAONE TICKETING також має свої недоліки. Висока вартість програмного забезпечення може бути суттєвою перешкодою для деяких компаній. Користувачі також скаржаться на повільну роботу сайту та поганий дизайн, що може впливати на зручність використання. Наявність лише однієї мови обмежує можливості використання програмного забезпечення в міжнародних компаніях. Крім того, через велику кількість інструментів і можливостей додатку, освоєння роботи з платформою може зайняти значний час. Це може вимагати додаткових ресурсів для навчання персоналу та адаптації до нових робочих процесів.

Для більш детального аналізу було створено порівняльну таблицю, яка підсумовує сильні та слабкі сторони обох наявних програмних забезпечень. Такий підхід дозволяє наочно продемонструвати ключові характеристики кожного

продукту, допомагаючи визначити, яке програмне забезпечення краще відповідає потребам котрситувачів.

Таблиця 1.1 – Порівняльна характеристка наявних програмних рішень

Характеристика	LiveAgent	NINJAONE TICKETING
Багатоканальне рішення	Так	Так
Керування всіма зв’язками	Так	Так
Підтримка різних відділів	Так (продажі, IT-сервіс)	Ні (лише підтримка клієнтів)
Безкоштовна база знань	Так	Ні
Популярні платформи мейлінгу	Ні	Так
Вартість	Висока	Висока
Випробувальний термін	Короткий	Відсутній
Мобільний додаток	Відсутній	Так
Автоматизація завдань	Так (тільки базова автоматизація)	Так (розширена автоматизація з інтелектуальними автовідповідачами)
Підтримка кількох мов	Так	Ні
Дизайн та адаптивність	Задовільний	Поганий
Інтеграція ринкових даних	Ні	Так
Час засвоєння	Середній	Довгий

1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення

На основі змістовного аналізу предметної області, її структурних та функціональних особливостей та наявних існуючих програмних рішень з’ясовано що дане програмне забезпечення буде реалізацією централізованої системи для синхронізації електронних листів з різних поштових сервісів.

Завдяки аналізу було встановлено потенційних користувачів та їх потреби, що допоможе максимально ефективно встановити функціональні та нефункціональні вимоги до ПЗ.

Основні вимоги до вебзастосунку включають низку функціональних можливостей для забезпечення ефективної роботи користувачів:

– авторизація користувача в вебзастосунку. Користувачі зможуть увійти до системи, використовуючи особисті облікові дані, для забезпечення конфіденційності та безпеки своїх даних;

– відновлення паролю. Вебзастосунок повинен надавати можливість користувачам відновлювати свій пароль у випадку втрати чи забуття;

– перегляд тікетів. Користувачі матимуть доступ до списку тікетів, що включають всі запити або питання, які були надіслані або отримані через систему;

– присвоєння категорії тікету. Можливість класифікувати тікети за різними категоріями для полегшення організації та швидкого пошуку необхідних запитів;

– присвоєння відповідального користувача за тікет. Можливість призначати конкретного користувача або команду для вирішення певного тікету для ефективного розподілу робочих завдань;

– зміна статусу тікету. Можливість змінювати стан тікету, наприклад, відкритий, у розгляді, в роботі, закритий тощо, для відображення поточного стану запиту;

– встановлення обмеження часу на відповідь та закриття статусу: Можливість встановлювати терміни відповіді на тікети та закриття їх статусу для забезпечення вчасного вирішення питань;

– відповідь на лист. Функціонал для надсилання відповідей на листи безпосередньо з вебзастосунку для швидкого і зручного комунікування з користувачами;

– автоматична синхронізація листів з доданих інтеграцій. Можливість автоматичної синхронізації електронних листів з іншими платформами чи сервісами для забезпечення єдиної точки доступу та управління;

– додавання інтеграцій. Можливість додавання нових інтеграцій з різними сервісами електронної пошти для розширення функціоналу та забезпечення

максимальної зручності для користувачів.

На рисунку 1.3 представлена діаграма варіантів використання, яка докладно ілюструє всі можливості та функціональні можливості, доступні користувачу у вебзастосунку. Ця діаграма показує різні сценарії, в яких користувач може взаємодіяти із системою, і включає такі ключові дії, як авторизація у вебзастосунку, процес відновлення паролю у разі його втрати, перегляд існуючих тікетів, присвоєння тікетів відповідним категоріям, призначення відповідального користувача для кожного тікету, зміна статусу тікету, встановлення часових обмежень на відповіді та закриття тікетів, можливість відповідати на отримані електронні листи, автоматичну синхронізацію листів з доданих інтеграцій та додавання нових інтеграцій для розширення функціональності системи. Діаграма варіантів використання є важливим інструментом для візуалізації та розуміння всіх функцій і можливостей, що надаються користувачам вебзастосунку, і допомагає визначити, як саме користувачі можуть взаємодіяти з системою для досягнення своїх цілей.

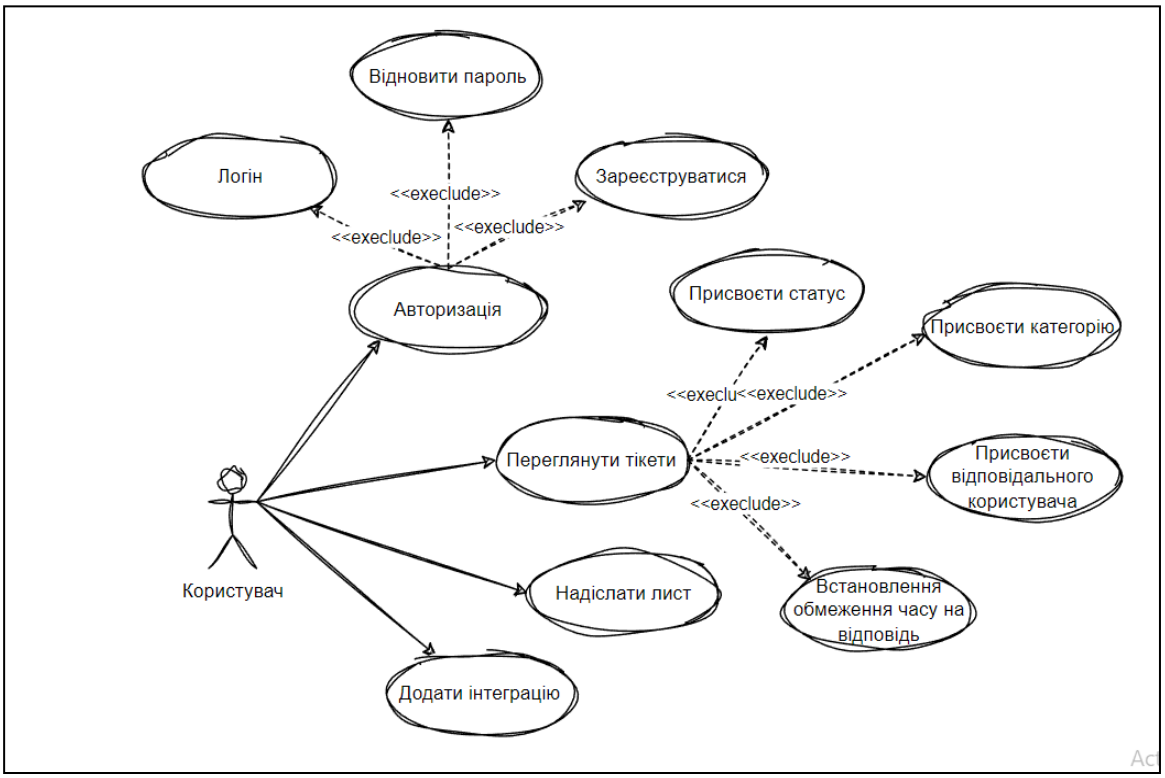


Рисунок 1.3 – Діаграма варіантів використання

На рисунку 1.4 подано детальну діаграму послідовності, яка ілюструє варіант використання, що стосується перегляду тікетів. Ця діаграма послідовності чітко показує, як саме користувач взаємодіє із системою для того, щоб переглянути список тікетів, які містять запити або питання, надіслані через вебзастосунок. Діаграма починається з того, що користувач ініціює запит на перегляд тікетів, після чого система звертається до бази даних для отримання списку тікетів, пов’язаних з даним користувачем. Далі система отримує необхідну інформацію з бази даних і відправляє її назад користувачеві у вигляді структурованого списку тікетів. Користувач може бачити цей список у своєму інтерфейсі, де кожен тікет містить важливу інформацію, таку як ідентифікатор тікету, категорія, відповідальний користувач, статус і строки виконання. Ця діаграма послідовності наочно демонструє всі етапи та взаємодії, які відбуваються між користувачем і системою під час перегляду тікетів, і допомагає зрозуміти, як відбувається обмін інформацією в рамках цього процесу.

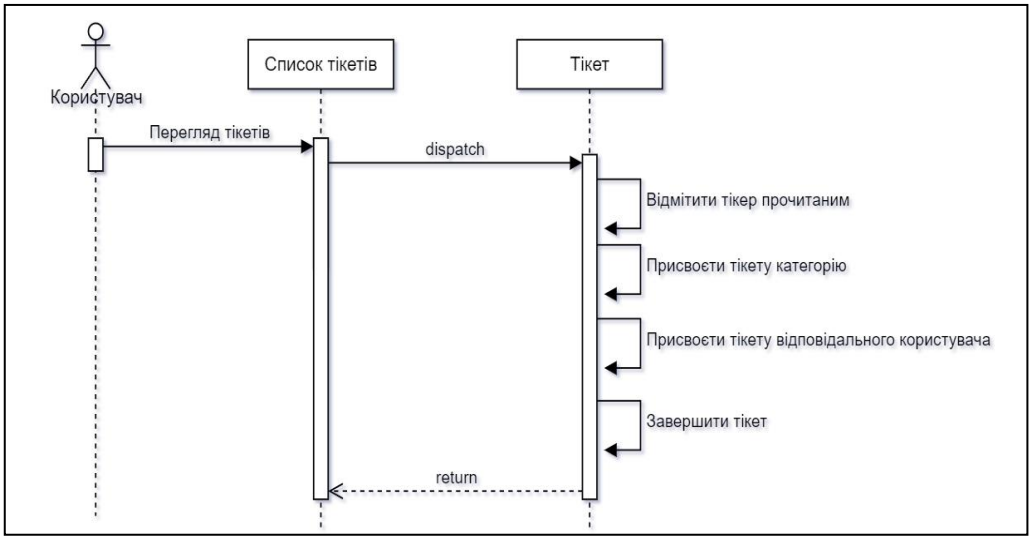


Рисунок 1.4 – Діаграма послідовності для варіанту використання перегляд тікетів

На рисунку 1.5 представлена детальна діаграма, яка показує процес авторизації користувача, розподілений на кілька етапів, що включають клієнтську сторону, вебзастосунок, сервер та базу даних. Ця діаграма наочно ілюструє кожний крок, який відбувається від моменту, коли користувач ініціює

авторизацію, до моменту підтвердження його доступу до системи.

Процес починається на стороні клієнта, де користувач вводить свої облікові дані, такі як ім'я користувача та пароль, у форму авторизації вебзастосунку. Після натискання кнопки "Увійти", ці дані передаються до вебзастосунку. Вебзастосунок отримує облікові дані та передає їх на сервер для подальшої обробки. Сервер перевіряє отриману інформацію та формує запит до бази даних для підтвердження введених даних.

База даних отримує запит на перевірку автентичності облікових даних користувача. Вона здійснює пошук у відповідних таблицях і порівнює надані дані з збереженими записами. Якщо дані співпадають, база даних надсилає підтвердження успішної автентифікації назад на сервер. Сервер, отримавши підтвердження від бази даних, генерує сесію для користувача та відправляє відповідний токен авторизації до вебзастосунку.

Вебзастосунок, у свою чергу, отримує цей токен та передає його клієнту, завершуючи процес авторизації. Користувач тепер може отримати доступ до захищених розділів вебзастосунку, використовуючи наданий токен для підтвердження своєї особи при подальшій взаємодії з системою.

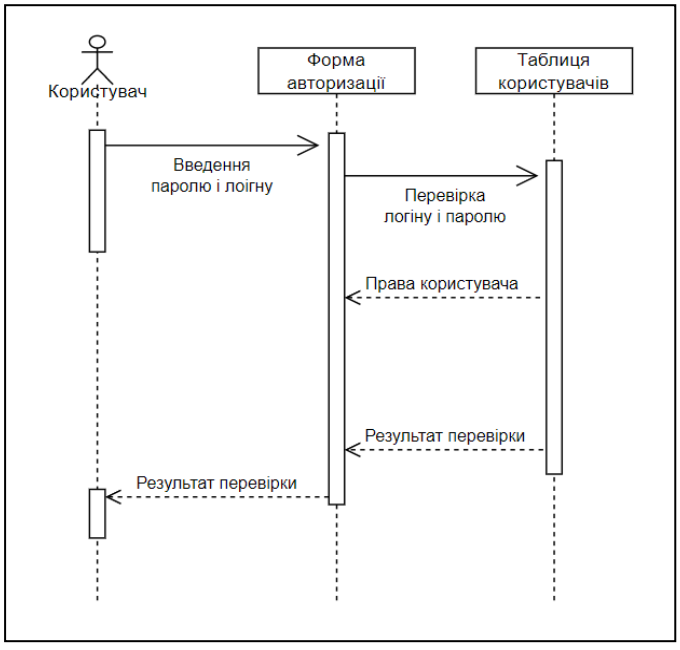


Рисунок 1.5 – Діаграма послідовності для варіанту використання авторизація

Таким чином, діаграма на рисунку 1.5 чітко демонструє всі етапи та взаємодії між клієнтом, вебзастосунком, сервером та базою даних під час процесу авторизації, показуючи, як дані проходять через кожен з цих компонентів для забезпечення безпечного доступу користувача до системи.

До нефункціональних вимог до вебзастосунку для централізації електронних листів з різних поштових сервісів належить забезпечення можливості одночасної роботи великої кількості користувачів із системою. Для цього потрібно гарантувати високу стійкість системи до різних видів навантажень, що включає оптимізацію серверної частини та бази даних, а також використання сучасних методів балансування навантаження.

У подальшому планується можливість розширення системи шляхом впровадження нових функціональних можливостей, тому розроблений вебзастосунок має дозволяти додавання нового функціоналу без необхідності внесення значних змін у програмний код. Це забезпечить гнучкість і масштабованість системи, дозволяючи їй адаптуватися до змінних вимог та нових викликів.

Вебзастосунок повинен забезпечувати високий рівень безпеки даних. Це включає шифрування даних під час передачі та зберігання, захист від несанкціонованого доступу. Це гарантуватиме, що особисті та конфіденційні дані користувачів будуть захищені від потенційних загроз.

У кольоровій гамі інтерфейсу користувача мають переважати пастельні кольори з невеликою кількістю відтінків, що сприятиме зниженню навантаження на очі користувачів та забезпечить приємний візуальний досвід. Вікна інтерфейсу повинні містити функціонал, який відповідає їхній назві, та бути згруповані у блоки відповідно до їхнього призначення. Це допоможе користувачам легко орієнтуватися в системі та швидко знаходити необхідні функції.

Отже, нефункціональні вимоги до вебзастосунку включають високу стійкість до навантажень, можливість легкого розширення функціональності та зручний, естетично привабливий інтерфейс користувача.

1.4 Висновки. Постановка задачі

Першим етапом виконання кваліфікаційної роботи було проведення детального та змістовного аналізу предметної області, її структурних та функціональних особливостей. Для цього було здійснено ґрунтовне дослідження різних поштових сервісів, зокрема їхнього поділу на види залежно від призначення та сфери використання. Аналіз допоміг глибше зрозуміти структуру та функціонал електронних поштових сервісів, що є важливим для подальшої роботи.

На основі отриманої інформації було визначено основну мету даної кваліфікаційної роботи. В результаті було визначено потенційних користувачів системи та розроблено діаграму варіантів використання для демонстрації основних функціональних можливостей програмного продукту.

Далі було проведено аналіз вже існуючих рішень на основі популярних сервісів LiveAgent та NINJAONE TICKETING. На основі проведеного аналізу було виявлено ключові характеристики, позитивні аспекти та недоліки різних сервісів. Це дозволить уникнути включення непотрібного функціоналу та покращити якість програмного продукту.

На основі зібраної інформації під час аналізу предметної області та вивчення існуючих рішень були визначені основні вимоги до розробки програмного продукту, його ключовий функціонал та вимоги до інтерфейсу користувача. Основні завдання на реалізацію включають:

- реалізація системи авторизації/реєстрації;
- розробка системи централізації електронних листів;
- впровадження можливості перегляду листів та створення відповіді;
- розробка системи додавання інтеграцій.

Виконання цього розділу допомогло встановити основні вимоги до програмного продукту, що спростить подальший процес вибору технологій та розробки програмного забезпечення.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз та вибір архітектури вебзастосунку

Вибір технології є ключовим етапом у процесі розробки програмного продукту, оскільки від цього вирішально залежить якість, продуктивність та функціональні можливості програми. Правильно підібрані технології можуть значно зменшити час розробки та вартість проекту, а також сприяти підвищенню ефективності його роботи. Наприклад, використання відкритих стандартів та інструментів може сприяти швидшому розвитку та підтримці продукту завдяки активній спільноті розробників.

Прикладом важливого впливу вибору технології може бути використання мови програмування. Наприклад, вибір мови з урахуванням швидкості виконання операцій або легкості розробки може значно вплинути на продуктивність та швидкість розробки. Крім того, вибір правильної архітектури додатку може значно полегшити розширення та модифікацію програми в майбутньому.

Наприклад, Airbnb, яка активно використовує технології розробки програм для створення своєї платформи для бронювання помешкань. Airbnb використовує фреймворк React для розробки веб-інтерфейсу своєї платформи, що дозволяє їм створювати інтерактивні та ефективні користувацькі інтерфейси. Крім того, Airbnb використовує технологію Node.js для розробки серверної частини своєї платформи, що дозволяє їм створювати швидкі та масштабовані додатки [7].

Ще одним прикладом є вибір бази даних. Вибір між реляційною та нереляційною базою даних може визначити масштабованість, продуктивність та зручність управління даними.

Наприклад, для великих обсягів даних та високої швидкодії запитів нереляційні бази даних можуть бути більш підходящим варіантом. Використання цих технологій допомагає Airbnb забезпечувати високу якість та продуктивність своєї платформи для користувачів по всьому світу за що і отримала свою популярність.

Для вимог поставленої задачі найкраще підходить архітектура RESTful API, яка базується на клієнт-серверній моделі. Це рішення забезпечує чітке розділення відповідальностей між компонентами, що займаються зберіганням та оновленням даних (сервером), і компонентами, які займаються відображенням даних на інтерфейсі користувача та реагуванням на дії користувача (клієнтом). Такий підхід дозволяє компонентам еволюціонувати незалежно один від одного, що значно спрощує подальше обслуговування та розвиток системи.

RESTful API вимагає, щоб дані передавалися у вигляді стандартних форматів, таких як HTML, XML, або JSON, що дозволяє забезпечити сумісність між різними системами та платформами. Як показано на рисунку 2.1, взаємодії між сервером та клієнтом є безстанними, тобто кожен запит містить всю необхідну інформацію для його обробки і не покладається на збереження стану з попередніх запитів. Це робить систему більш надійною та масштабованою.

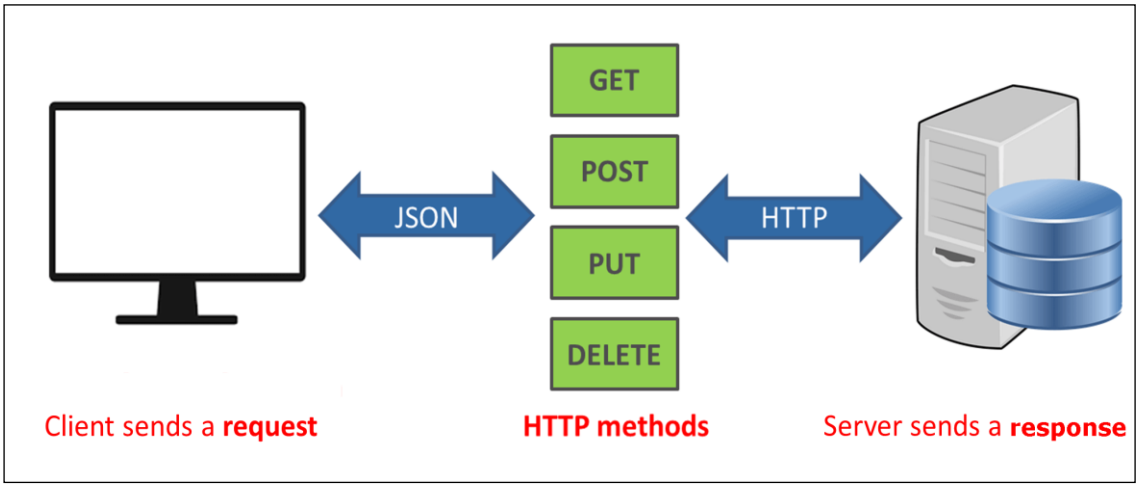


Рисунок 2.1 – Загальна схема взаємодії елементів архітектури

Кожне програмне забезпечення можна поділити на окремі частини або об'єкти, збираючи їх разом для створення готового продукту. Цей підхід особливо зручний у створенні програм командою розробників, оскільки дозволяє розробляти відносно незалежні модулі окремо, а потім інтегрувати їх разом. Обрана мова програмування складається з найменших складових частин – модулів, що сприяє легшій організації коду та його підтримці.

Для порівняння, розглянемо архітектуру SOAP (Simple Object Access Protocol). SOAP також використовується для побудови веб-сервісів, однак має кілька відмінностей від RESTful API.

SOAP (Simple Object Access Protocol) – це протокол обміну повідомленнями, який використовується для обміну структурованими даними в рамках розподілених обчислень у мережах. Він забезпечує спосіб взаємодії між програмами, що працюють на різних операційних системах, з використанням різних технологій і мов програмування. SOAP використовує XML для кодування повідомлень, що забезпечує високий рівень гнучкості та сумісності.

Повідомлення SOAP складається з трьох основних частин: заголовку, тіла і додаткових структур (зазвичай використовується WSDL - Web Services Description Language для опису послуг). Заголовок SOAP є опціональною частиною, яка може містити метадані про повідомлення, такі як інформація про безпеку, маршрутизацію або транзакції. Він використовується для обробки додаткової інформації, яка не відноситься безпосередньо до обробки запиту чи відповіді. Тіло SOAP є обов'язковою частиною, що містить основні дані повідомлення, які будуть оброблятися. Тіло може містити інформацію про запит або відповідь веб-сервісу.

SOAP зазвичай використовує HTTP/HTTPS як транспортний протокол, але також підтримує інші транспортні протоколи, такі як SMTP. Завдяки використанню HTTP, SOAP легко проходить через брандмауери та проксі-сервери.

Однією з переваг SOAP є його незалежність від платформи та мови програмування. Завдяки використанню XML, SOAP дозволяє взаємодіяти системам, що працюють на різних платформах і написані на різних мовах програмування.

SOAP також забезпечує високий рівень безпеки завдяки підтримці WS-Security, що забезпечує комплексні механізми автентифікації, авторизації, шифрування і цілісності повідомлень. Це робить SOAP придатним для корпоративних додатків, де необхідна висока безпека.

Проте SOAP має і свої недоліки. Одним з них є складність. Використання XML для кодування повідомлень робить SOAP більш складним і важким у порівнянні з іншими протоколами, такими як RESTful API. Високий обсяг накладних витрат на обробку та передачу XML-повідомлень може уповільнювати продуктивність.

Конфігурування та розгортання SOAP-сервісів може бути складним процесом, особливо при використанні розширених можливостей, таких як безпека та транзакції. Більший розмір повідомлень і складність обробки XML-повідомлень може призводити до нижчої ефективності в порівнянні з легшими протоколами, такими як REST.

Причини вибору RESTful API:

– простота та легкість використання. RESTful API легко зрозуміти та використовувати завдяки використанню стандартних HTTP методів. Зрозумілі формати даних (JSON) роблять його більш доступним для розробників;

– гнучкість і масштабованість. Безстанні взаємодії роблять систему більш надійною та масштабованою, що важливо для великих та динамічних проєктів. Клієнт та сервер можуть оновлюватись незалежно один від одного без потреби повної реконструкції системи;

– широка підтримка і сумісність. RESTful API підтримується багатьма мовами програмування та платформами, що полегшує інтеграцію та взаємодію між різними системами. Простота формату JSON дозволяє швидше обробляти дані у порівнянні з XML;

– швидкість та ефективність. RESTful API легкий за своєю природою, що забезпечує швидше виконання запитів. Мінімальний накладний витрат на обробку повідомлень у порівнянні з SOAP.

RESTful API забезпечує простоту, гнучкість та ефективність, що робить його кращим вибором для багатьох сучасних додатків, особливо тих, що потребують швидкого обміну даними та легкої інтеграції з різними системами. SOAP, хоча і має свої переваги, такі як висока безпека та підтримка транзакцій, може бути занадто складним та повільним для багатьох сучасних застосунків.

2.2 Аналіз та вибір технологій і засобів реалізації системи

Далі проведемо характеристику технологій розробки програмного забезпечення, які будуть використовуватися для реалізації ПЗ. Цей огляд охоплює вибір інструментів, платформ і технологій, які сприятимуть ефективній розробці, впровадженню та підтримці системи.

Перш за все, мова програмування і фреймворк, які будуть використані для розробки основної логіки та інтерфейсу користувача. Вибір падає на JavaScript.

JavaScript є однією з найпопулярніших мов програмування у світі з кількох причин, які роблять її вибір очевидним для багатьох розробників та компаній. Ось декілька ключових аргументів, чому обирають JavaScript:

– універсальність та застосування. JavaScript є стандартом для створення динамічних і інтерактивних веб-сторінок. Усі основні веб-браузери підтримують JavaScript без потреби у додаткових плагінах;

– велика спільнота та екосистема;

– бібліотеки та фреймворки: Є безліч потужних бібліотек та фреймворків, таких як React, Angular, Vue.js для фронтенду;

– простий синтаксис. JavaScript має відносно простий і зрозумілий синтаксис, що робить його доступним для новачків у програмуванні;

– миттєвий результат: Завдяки можливості виконувати код безпосередньо у веб-браузері, розробники можуть одразу бачити результати своєї роботи, що стимулює навчання та експериментування.

Таким чином, JavaScript є відмінним вибором завдяки своїй універсальності, великій спільноті, легкості вивчення, інтеграції з іншими технологіями та постійному розвитку. Це робить його однією з найпопулярніших мов програмування у світі.

Проаналізуємо кілька популярних фреймворків для розробки користувацького інтерфейс на мові програмування JavaScript, а саме Angular, Vue.js та React.

Angular — це повноцінний фреймворк для розробки веб-додатків, створений компанією Google, який використовує TypeScript і забезпечує потужні можливості для створення складних додатків з архітектурою Model-View-Controller (MVC). Він включає вбудовані функції для роботи з формами, маршрутизацією, HTTP-запитами та багато іншого, що робить його ідеальним для великих і масштабованих проектів.

З плюсів Angular можна виділити:

– надає все необхідне для створення великих додатків, включаючи вбудовану підтримку для маршрутизації, форми, HTTP-клієнта тощо;

– використання TypeScript забезпечує статичну типізацію, що допомагає виявляти помилки на етапі написання коду;

– Модель-Vue-Модель забезпечує чисту архітектуру.

З мінусів Angular можна виділити:

– Angular має круту криву навчання через свою складність і велике число концепцій;

– продуктивність. Іноді може бути повільнішим у порівнянні з легшими фреймворками.

Vue — це прогресивний фреймворк для створення користувацьких інтерфейсів, який відзначається своєю простотою, гнучкістю та легкістю в інтеграції з іншими проєктами. Його можна використовувати як для розробки невеликих компонентів, так і для створення великих односторінкових додатків.

З плюсів Vue можна виділити:

– легкий для вивчення, з простим і зрозумілим синтаксисом;

– можна використовувати як для малих, так і для великих проектів;

– малий розмір бібліотеки, що покращує продуктивність.

З мінусів Angular можна виділити:

– менша спільнота порівняно з React або Angular, що може ускладнити пошук ресурсів і підтримки;

– компанія розробник фреймворку не велика, що додає певні ризики;

– менш структурована розробка великих додатків.

React - це бібліотека JavaScript для розробки користувацьких інтерфейсів, що використовується для створення веб-додатків і мобільних додатків. Він пропонує компонентний підхід до побудови інтерфейсу, що робить розробку більш організованою та ефективною.

З плюсів React можна виділити:

– компонентний підхід: Зручний для створення багаторазових і модульних компонентів;

– JSX. Дозволяє писати в JavaScript, що спрощує розробку UI;

– велика екосистема. Багато бібліотек і інструментів, таких як Redux, React Router, забезпечують розширюваність;

– велика спільнота. Широке використання та активна спільнота, що забезпечує велику кількість навчальних матеріалів, бібліотек та інструментів;

– можливість використовувати React для мобільних додатків через React Native.

З мінусів Angular можна виділити:

– для деяких розробників може бути незвичним писати HTML в JavaScript;

– потребує більше налаштувань інструментарію для початку роботи порівняно з іншими фреймворками.

Для кращого візуального сприйняття переваг та недоліків кожної технології було створено порівняльну таблицю 2.1.

Таблиця 2.1 – Порівняльна характеристка переваг та недоліків кожної технології

Характеристика	**React**	**Angular**	**Vue.js**
Розробник	Facebook	Google	Evan You
Початковий реліз	2013	2010	2014
Архітектура	Бібліотека для побудови UI	Повноцінний фреймворк MVC	Прогресивний фреймворк
Основний принцип	Компонентний підхід	MVVM (Model-View-ViewModel)	Компонентний підхід
Мова програмування	JavaScript (ES6+), JSX	TypeScript	JavaScript (ES6+)

Продовження таблиці 2.1

Характеристика	**React**	**Angular**	**Vue.js**
Документація	Відмінна	Відмінна	Відмінна
Складність	Середня	Висока	Низька
Розширюваність	Висока	Висока	Висока
Продуктивність	Висока	Висока	Висока
Крива навчання	Помірна	Крута	Помірна
Двостороннє зв'язування даних	Ні, односторонній (односпрямований потік даних)	Так	Так
Шаблони	JSX	HTML-шаблони	HTML-шаблони
Управління станом	Redux, MobX	NgRx	Vuex
Роутинг	React Router	Вбудований Angular Router	Vue Router
Тестування	Jest, Enzyme	Jasmine, Karma	Mocha, Chai
Популярність	Дуже висока	Висока	Зростаюча
Підтримка мобільних платформ	React Native	NativeScript, Ionic	Weex
Корпоративна підтримка	Висока	Висока	Зростаюча
Екосистема	Широка	Широка	Зростаюча

React був обраний для реалізації кваліфікаційної роботи через свою гнучкість, потужні можливості та активну підтримку з боку спільноти та індустрії. Компонентний підхід React дозволяє створювати складні інтерфейси з незалежних частин, що спрощує підтримку та розширення коду. Односторонній потік даних забезпечує передбачуваність поведінки додатка, а використання віртуального DOM підвищує продуктивність, мінімізуючи оновлення реального DOM.Крім того, React і має велику спільноту, яка створює та підтримує безліч бібліотек і інструментів, що полегшують розробку. Широка екосистема для управління станом (Redux, MobX), роутингу (React Router) та тестування (Jest, Enzyme) забезпечує наявність перевірених рішень для типових завдань, що прискорює розробку і знижує ризик помилок. Це робить React відмінним вибором для створення якісного, ефективного та сучасного вебзастосунку.

ASP.NET Core API було обрано для реалізації бекенду з міркувань ефективності та надійності, оскільки це потужний фреймворк для розробки веб-сервісів та API, який пропонує безліч переваг у порівнянні з іншими технологіями.

По-перше, він забезпечує високу продуктивність та швидкодію завдяки своїй оптимізації під сучасні потреби веб-розробки. ASP.NET Core відкриває шлях для розробки крос-платформових додатків, оскільки може працювати як на Windows, так і на Linux та macOS.

По-друге, ASP.NET Core API має різноманітні засоби автентифікації та авторизації, що дозволяють забезпечити безпеку додатків на високому рівні. Вбудовані механізми підтримки JWT, OAuth та інших стандартів забезпечують зручну і надійну захист інформації.

Крім того, ASP.NET Core API має велику екосистему бібліотек та інструментів, які допомагають розробникам швидко створювати різноманітні та потужні додатки. Це включає в себе ORM-бібліотеки, такі як Entity Framework Core, а також інструменти для тестування, логування та моніторингу додатків.

Нарешті, вибір ASP.NET Core API забезпечує інтеграцію з іншими сервісами та технологіями, зокрема з облачними платформами, базами даних та фронтенд-фреймворками. Це дозволяє розробникам створювати складні та масштабовані додатки, які задовольняють потреби сучасного ринку програмного забезпечення.

Entity Framework та SQL Server були обрані для розробки бекенду з міркувань ефективності, надійності та інтеграції, що забезпечують ефективну роботу з базами даних. Ця комбінація дозволяє розробникам швидко і легко працювати з даними, використовуючи сучасні технології для створення надійних і масштабованих додатків. Завдяки Entity Framework, розробники можуть працювати з базою даних на рівні об'єктів, що значно підвищує продуктивність та знижує ймовірність помилок, пов'язаних з написанням SQL-запитів вручну. SQL Server, в свою чергу, забезпечує потужні засоби для зберігання та обробки даних,

включаючи підтримку складних запитів, транзакцій та процедур, що робить його надійним вибором для будь-яких масштабів проектів. Entity Framework є об'єктно-реляційним маппером (ORM), який спрощує взаємодію з базою даних за допомогою об'єктно-орієнтованих підходів. Це дозволяє розробникам працювати з даними на рівні об'єктів, що значно підвищує продуктивність і знижує кількість помилок, пов'язаних з написанням SQL-запитів вручну. Використання LINQ у поєднанні з Entity Framework забезпечує інтуїтивно зрозумілий спосіб запитів до бази даних, що підвищує швидкість розробки та читабельність коду.

SQL Server надає широкі можливості для управління даними, підтримуючи складні запити, транзакції та процедури. Він є надійним та перевіреним рішенням для зберігання даних у різних масштабах, від невеликих додатків до великих корпоративних систем.

Інтеграція з Entity Framework робить процес розробки та обслуговування бази даних більш ефективним, оскільки обидві технології добре підтримуються та документовані, що забезпечує розробникам доступ до великої кількості ресурсів та прикладів.

SQL Server забезпечує високу продуктивність і надійність завдяки своїм оптимізованим механізмам зберігання та обробки даних. Він також пропонує потужні засоби для забезпечення безпеки, такі як шифрування даних, управління ролями і доступом, а також вбудовані інструменти для резервного копіювання та відновлення. Це робить SQL Server ідеальним вибором для зберігання критично важливих даних, що потребують високого рівня захисту. Поєднання Entity Framework та SQL Server дозволяє створювати масштабовані додатки, які можуть легко адаптуватися до зростаючих потреб бізнесу. SQL Server підтримує роботу з великими обсягами даних та забезпечує ефективне управління ресурсами, що є критично важливим для розробки масштабованих рішень.

Крім того, вбудовані інструменти аналітики та звітності в SQL Server дозволяють розробникам отримувати глибокі інсайти з даних, що допомагає приймати обґрунтовані рішення та оптимізувати бізнес-процеси.

Таким чином, вибір Entity Framework та SQL Server для розробки бекенду обумовлений їхніми потужними можливостями, інтеграцією, продуктивністю, безпекою та підтримкою масштабованості.

Ця комбінація забезпечує ефективну роботу з даними та сприяє створенню надійних, масштабованих та безпечних додатків, які можуть задовольнити потреби сучасних бізнес-середовищ.

2.3 Проектування структури даних та моделі бази даних

Наступним етапом є проектування структур даних. Аналіз структури даних починається з перетворення даних з їхнього вихідного формату (наприклад, первинного документа) у формат двовимірної таблиці, яка містить певну кількість рядків і стовпців. Структура реляційних відносин повинна бути оптимальною, тобто найбільш стійкою до змін у даних. У нашій схемі можуть виникнути проблеми надмірності даних та потенційної суперечливості (аномалії).

Надмірність означає повторення даних у різних рядках однієї таблиці або в різних таблицях бази даних. Наприклад, багаторазове повторення одного виду для певної групи тварин. Аномалії – це проблеми, що виникають через дефекти проектування бази даних. Існують три типи аномалій:

Аномалії вставки проявляються при введенні даних у дефектну таблицю. Додаючи інформацію про тварину, ми маємо додати код, прізвище власника та номер лікування. Якщо ввести дані, що не відповідають наявним у таблиці, буде незрозуміло, який з рядків бази даних містить правильну інформацію. Існують три види:

– аномалії видалення виникають при видаленні даних з дефектної схеми;

– аномалії модифікації виникають при зміні даних дефектної схеми;

– розв'язання цих проблем полягає у розділенні даних і зв’язків між ними, що досягається за допомогою процедури нормалізації.

Згідно першої нормальної форми усі атрибути відношення повинні бути унікальними, тобто не допускається їх дублювання. Для зведення відношення до 1НФ потрібно усунути дублювання таких атрибутів, як ім’я (Ім’я користувача, ім’я інтеграції).

Згідно першої нормальної форми усі атрибути відношення повинні бути атомарними, тобто неподільними. У даному відношенні всі атрибути є атомарні. Згідно першої нормальної форми визначені поля БД, що обраховуються з вхідних даних, будуть виводитись у формі чи звіті, але не зберігаються у БД. При аналізі всі атрибути є постійними.

Згідно першої нормальної форми повинні бути визначено первинні ключі. Очевидно що для даного відношення первинним ключем будуть наступні атрибути:

- код інтеграції;
- код тікету;
- код меседжу;
- код лейблу;
- код кастомера;
- код правила (рул);
- код автовідповідача;
- код системного повідомлення;
- код логу змін;
- код таски;
- код шаблону;
- код СЛА;
- код тегу;
- код нотатку;
- код користувачу.

Відношення в другій нормальній формі мають відповідати таким вимогам:

- відношення знаходиться у 1НФ;

– усунення функціональних залежностей – неключове поле залежить від частини ключа, для усунення використовуємо розбиття на 2 сутності.

Для приведення даного відношення до 2НФ необхідно його розбити на проекції (відношення), при цьому повинна бути виконана умова відновлення вихідного відношення без втрат. Відношення набудуть такий вигляд, що зображено на рисунку 2.2.

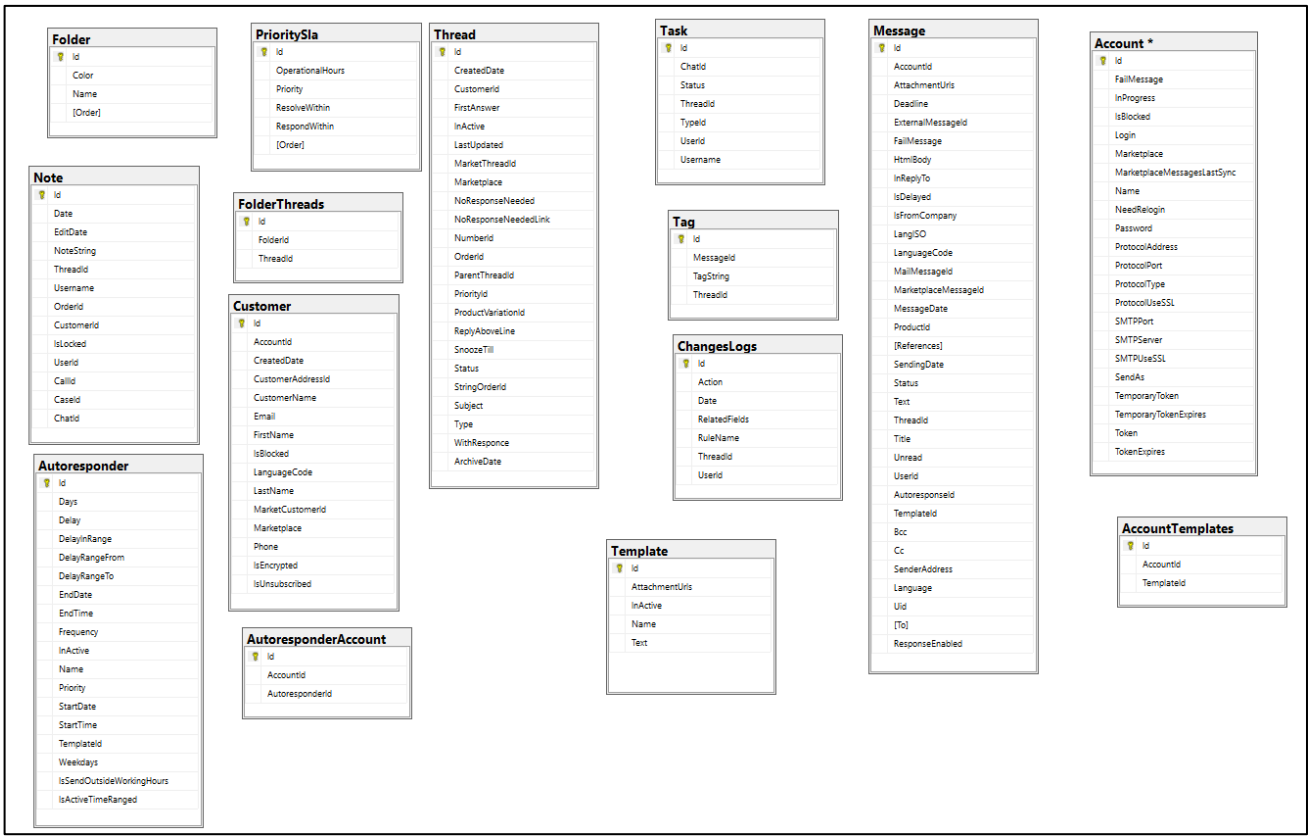


Рисунок 2.2 – Відношення в другій нормальній формі

Відношення в третій нормальній формі мають відповідати таким вимогам:

– відношення знаходиться у 2НФ;

– кожен його не ключовий атрибут не є транзитивно залежним від іншого не ключового атрибуту.

Для зведення відношення до 3НФ потрібно усунути транзитивну залежність не ключових атрибутів від інших не ключових атрибутів, а саме:

– тікет – лист, тому потрібно створити додаткову табличку «Тікети», до

якої потрібно прив’язувати нову таблицю «Листи»;

– тікет – нотаток, тому потрібно створити додаткову табличку «Тікети», до якої потрібно прив’язувати нову таблицю «Нотатки»;

– тікет – кастомер, тому потрібно створити додаткову табличку «Кастомери», до якої потрібно прив’язувати нову таблицю «Тікети»;

– тікет – лог зміни, тому потрібно створити додаткову табличку «Тікети», до якої потрібно прив’язувати нову таблицю «Логи змін»;

– тікет – завдання, тому потрібно створити додаткову табличку «Тікети», до якої потрібно прив’язувати нову таблицю «Завдання»;

– тікет – кастомер, тому потрібно створити додаткову табличку «Кастомери», до якої потрібно прив’язувати нову таблицю «Тікети»;

– інтеграція – лист, тому потрібно створити таблицю «Інтеграція», куди потрібно прив’язати таблицю «Листи».

Відношення в нормальній формі Бойса-Кодда мають вже знаходитися у третій нормальній формі, а також мають бути відсутні залежності первинного ключа від неключових атрибуті.

Всі таблиці в спроєктованій базі даних знаходяться в даній формі, так як всі атрибути мають єдині значення (атомарні), будь-який неключовий атрибут функціонально залежить від первинного ключа, відсутня транзитивна залежність, а також ключ являється потенційним ключем.

Відношення в четвертій нормальній формі мають вже знаходитися в нормальній формі Бойса-Клойда, а також в них відсутні многозначні залежності, що не являються функціональними залежностями. Отже, всі таблиці бази даних знаходяться в даній формі.

Відношення в четвертій нормальній формі мають відповідати таким вимогам:

– відношення знаходиться в BCNF;

– відсутні многозначні залежності, що не являються функціональними залежностями.

Всі таблиці в спроєктованій БД знаходяться в 4НФ так як відсутні многозначні залежності, що не являються функціональними залежностями.

Відношення в п’ятій нормальній формі мають вже знаходиться в четвертій, а також тоді і тільки тоді, коли будь-яка залежність по з'єднанню в ньому визначається тільки його можливими ключами - іншими словами, кожна проекція такого відношення містить не менше одного можливого ключа і не менше одного неключевого атрибута.

Всі таблиці в спроєктованій БД знаходяться в 5НФ так як кожна проекція такого відношення містить не менше одного можливого ключа і не менше одного неключевого атрибута.

Після нормалізації таблиць варто створити зв’зки між таблицями:

– один тікет може мати багато листів, один належить одному тікету, тому зв’язок один-до-багатьох;

– одна інтеграція може мати багато тікетів, один тікет належить одній інтеграції, тому зв’язок один-до-багатьох;

– один тікет може мати багато нотатків, один нотаток належить одному тікету, тому зв’язок один-до-багатьох;

– один кастомер може мати багато тікетів, один тікет належить одному кастомеру, тому зв’язок один-до-багатьох.

В результаті нормалізації і створення зв’язків між таблицями було отриману наступну схему даних, що зображена на рисунку 2.3 .

На спроектованій схемі бази даних присутні таблиці – це ті дані, які будуть вводитись і виводитись у вебзастусунку в процесі його роботи.

Основними структурами даних у проєкті являються масиви та об’єкти. Здебільшого типами даних, які зберігаються у цих перерахуваннях є описані вище об’єкти.

Після успішного проєктування структур даних можна перейти безпосередньо до їх програмної реалізації.

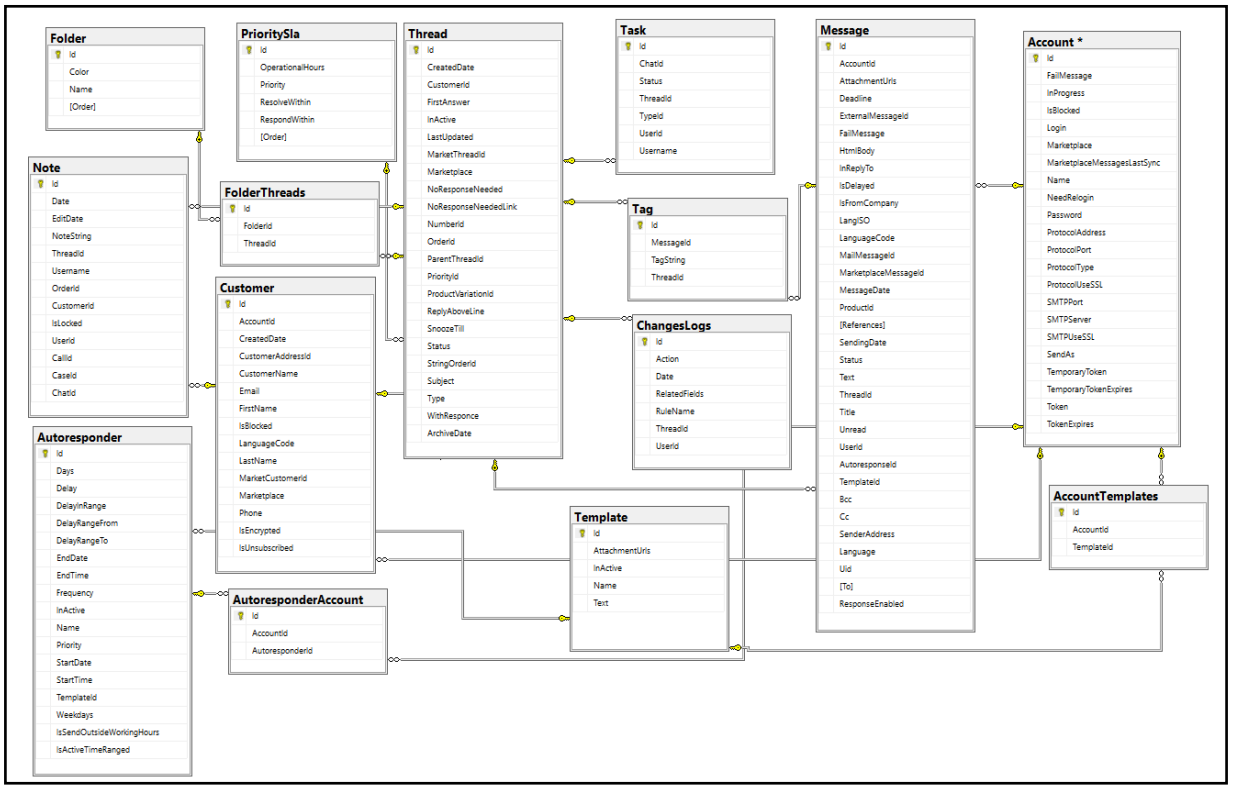


Рисунок 2.3 – Схема даних

2.4 Створення макета вебзастосунку та дизайн

Популярність вебзастосунку напряму залежить від того, наскільки якісно спроєктовано користувацький інтерфейс, наскільки він є зручним.

Для оцінки якості інтерфейсу можна скористуватись принципом Most Advanced Acceptable (найбільш прогресивний і в той же час прийнятний). На практиці це означає, що інтерфейс має бути інноваційним, але при цьому достатньо зрозумілим. Отже, ускладнюючи стандартні процеси, можна тільки роздратувати користувача.

При перегляді веб-сторінки, користувачі її не читають, а тільки сканують її за Z-паттерном, шаблоном у формі букви “Z”. Коли людина сканує сторінку, погляд природнім образом рухається зліва направо, зверху вниз. Z-паттерн повторює цю траєкторію. За ним погляд спочатку рухається по верхній горизонталі – з лівого кута в правий, потім з правого верхнього кута

переміщується в лівий нижній, малюючи діагональ, і нарешті по горизонталі в з нижнього лівого кута в правий, як зображено на рисунку 2.4.

Суть в тому, щоб управляти поглядом користувача і задавати траєкторію руху, тому необхідно розробляти, хоча б головну сторінку, опираючись на це правило. На початковій точці шляху користувача найкраще розмістити логотип, в правому верхньому куту ті елементи, на які слід звернути увагу в першу чергу, а в центральній частині сторінки цікавий контент.

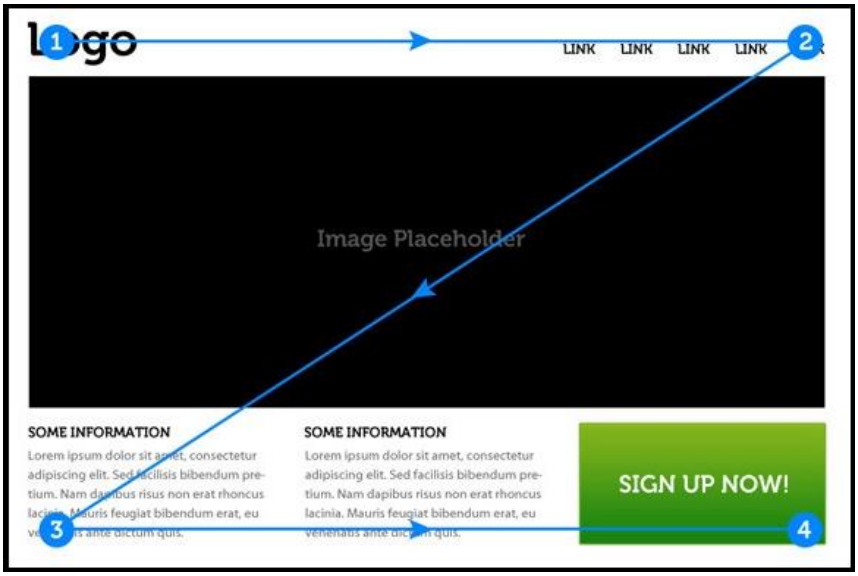


Рисунок 2.4 - Траєкторія погляду за Z-паттерном

Після проєктування головної сторінки слід визначити вигляд форм сайту.

У користувачів сайту є практична ціль. Часто буває, що між ним і його ціллю стоїть одна річ – форма. Форми досі залишаються одним з голоних типів взаємодії на сайтах і додатках. Більше того, форми часто вважають фінальним кроком на шляху користувача до цілі, тому їх заповнення має бути швидким і зрозумілим.

Стандартна форма складається з наступних компонентів:

– структура (включає в себе порядок полів, загальний вид на сторінці і логічні зв’язки між полями);

– назви полів (говорять користувача, що означає відповідне поле);

– кнопки дії (наприклад, інформація відправляється в базу);

– зворотній зв’язок (сповіщення про успішне заповнення форми, або ж навпаки помилку).

Правильно розроблена форма включає в себе логічне і правильне розміщення полів, а також передбачує, що постійне читання форми по Z-паттерну заповільнює обробку інформації і робить процес заповнення форми переривистим, а розміщення в одній колонці просте і зрозуміле – зверху вниз, як зображено на рисунку 2.5.

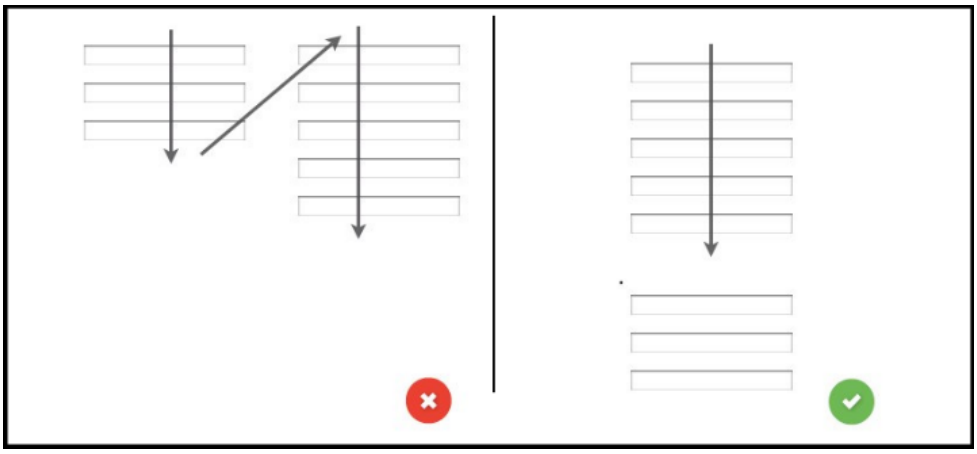


Рисунок 2.5 - Приклад розміщення полів в формі

Також кількість полів вводу слід звести до мінімуму, назви полів мають бути над ними або ж всередині, а кнопки дії мають візуально вирізнятись, інакше користувач може помилитись.

Зручний інтерфейс обов’язково включає в себе адаптивність веб-дизайну, тобто коректне і зручне для різних пристроїв представлення сайту. Адаптивний веб-додаток автоматично підлаштовується під розміри вікна браузера, що є дуже зручним. Всі сторінки в гнучкій версії доступні по одній URL-адресі, що вже спочатку розробки позбавляє від ряду проблем, а також при завантаженні зберігається дизайн і структура.

Найпростішим типом адаптивності для простих ресурсів є масштабування зображень і текстів, при цьому змінюється не весь сайт, а його окремі блоки і частини, так як зображено на рисунку 2.6.

Рисунок 2.6 - Масштабування веб-додатку

Для кращого сприймання інформації користувачем було спроєктовано мінімалістичний і гарний дизайн. Використовуючи розглянуті принципи побудови інтерфейсу для кращого сприйняття інформації користувачем та інтуїтивно зрозумілий.

Для авторизації користувачів було спроєктовано форма для реєстрації нового облікового запису, яка зображена на рисунку 2.8 та входу вже існуючий, яка зображена на рисунку 2.7.

Рисунок 2.7 – Форма для входу

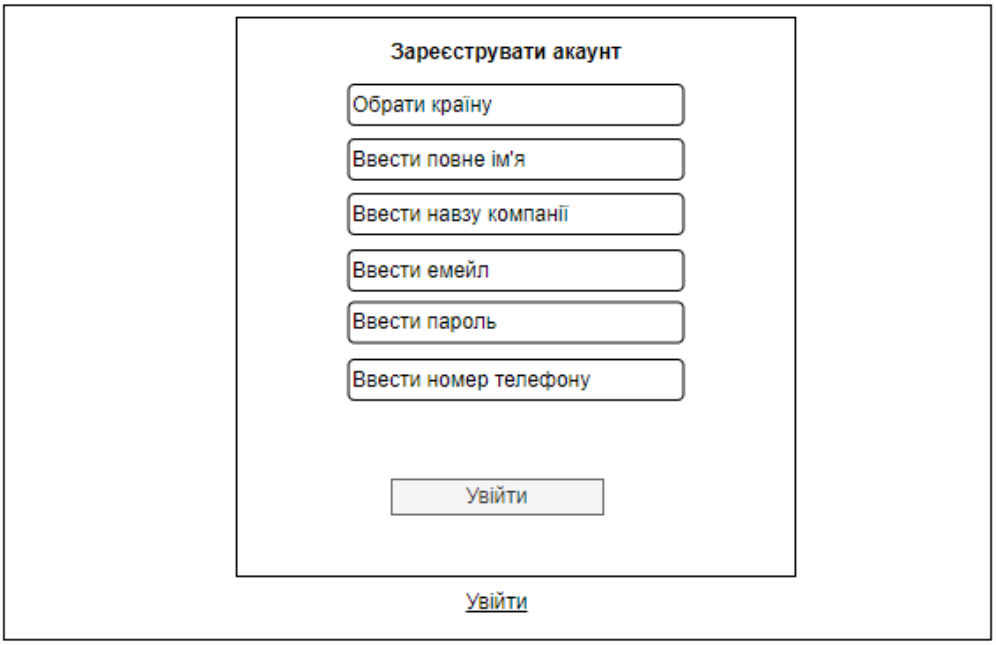


Рисунок 2.8 – Форма для реєстрації

Далі було спроєктовано загальний шаблон вебзастусунку, який зображений на рисунку 2.9.

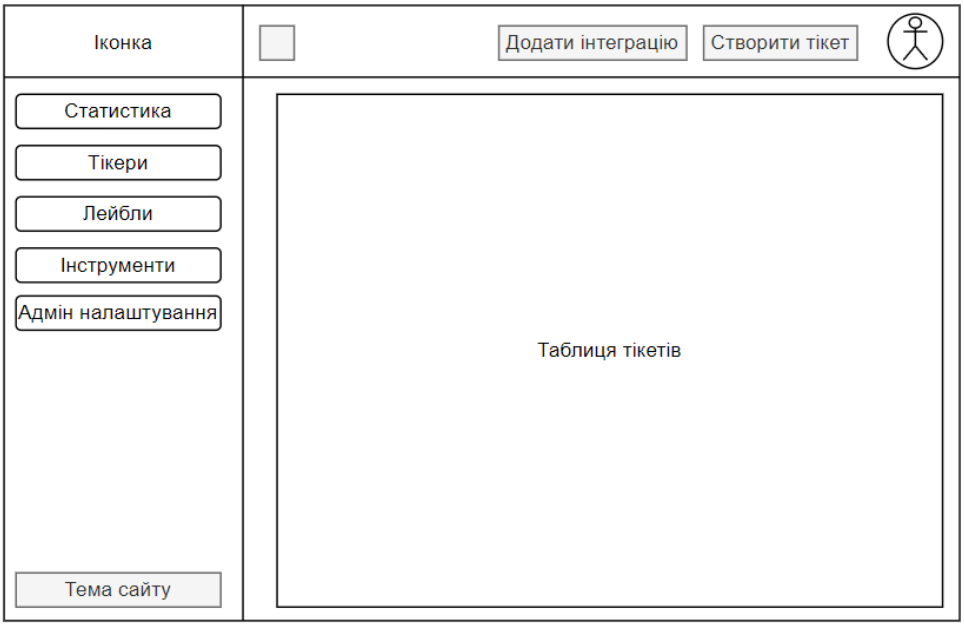


Рисунок 2.9 – Загальний шаблон вебзастосунку

Наступним був спроєктован компонент для додавання інтеграцій, який зображений рисунку 2.10.

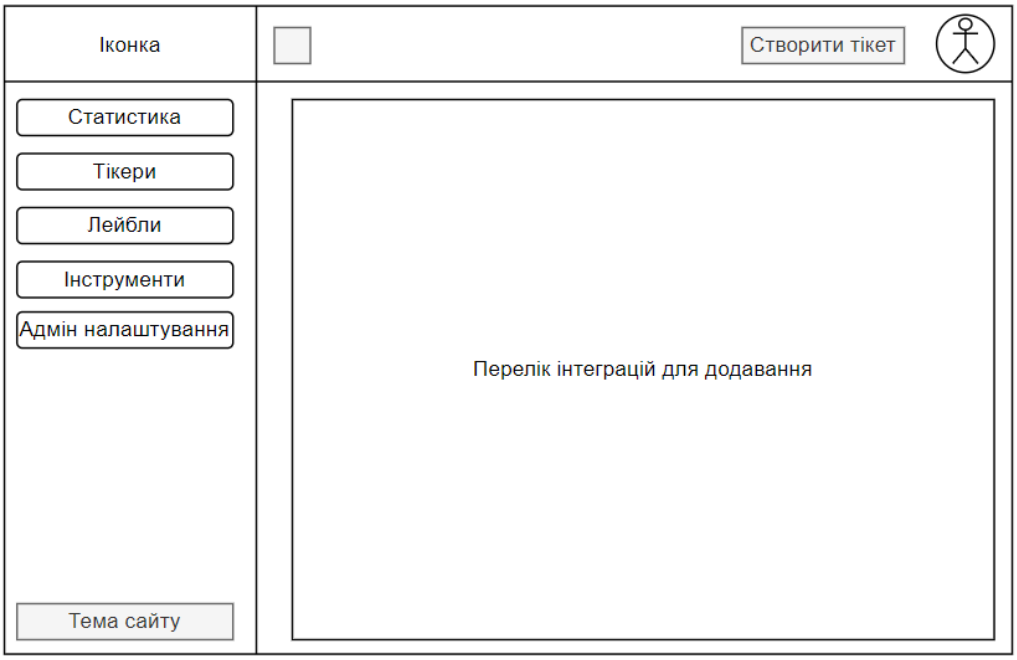


Рисунок 2.10 – Компонент для додавання інтеграцій

Далі було сторінку для створення тікету в вебзастусунку, яка зображений на рисунку 2.11.

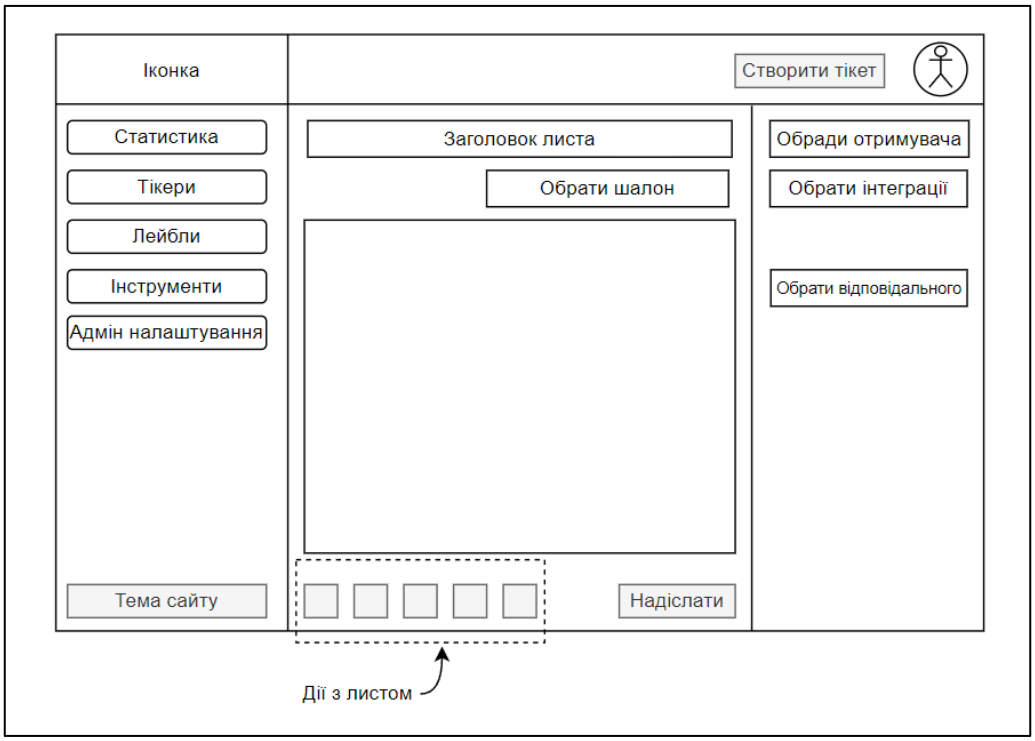


Рисунок 2.11 – Створення листу

Далі було сторінку перегляду лмстів, яка зображений на рисунку 2.12.

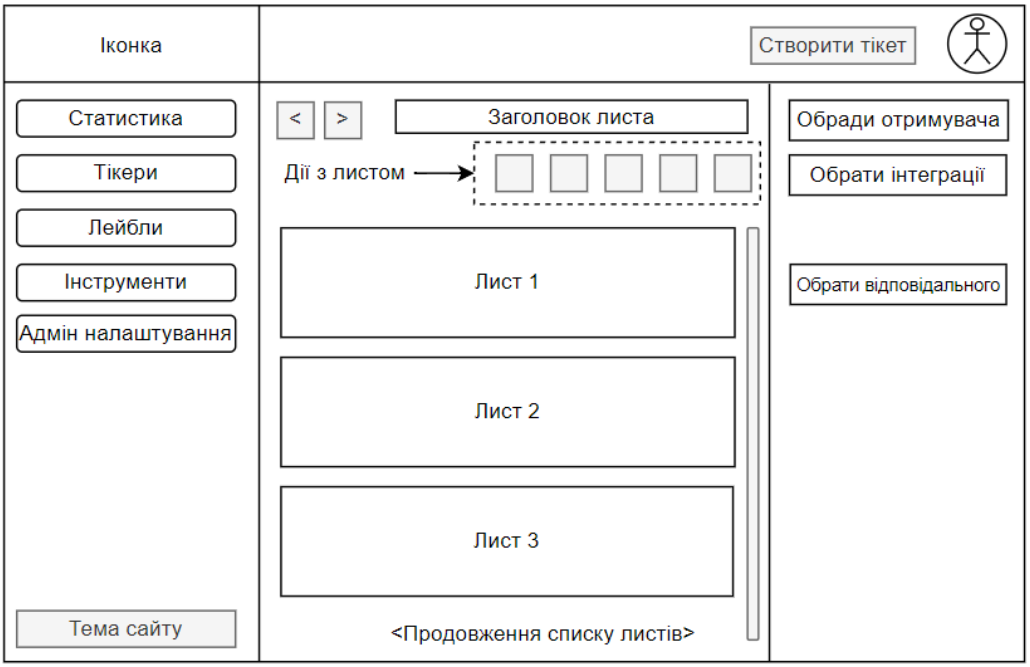


Рисунок 2.12 – Перегляд листів

2.5 Висновки

В ході розробки програмного забезпечення було вирішено використовувати архітектуру RESTful API, яка ґрунтується на моделі клієнт-сервер. Це рішення дозволило чітко розділити відповідальності між компонентами, що відповідають за зберігання та обробку даних (сервер), і компонентами, що займаються відображенням даних і реагуванням на дії користувача (клієнт).

Для розробки користувацького інтерфейсу була обрана бібліотека React. Ця бібліотека дозволяє створювати організовані та ефективні інтерфейси за допомогою компонентного підходу.

Для реалізації бекенду було обрано фреймворк ASP.NET Core API. Цей фреймворк забезпечує високу ефективність, надійність і безпеку, що є критичними для вебсервісів та API. Використання Entity Framework та SQL Server дозволить забезпечити ефективну роботу з базами даних.

Наступним важливим етапом було проектування структур даних. Структура реляційних відносин була оптимізована для забезпечення стійкості до змін у даних і забезпечення цілісності даних. Після нормалізації даних були створені зв’язки між таблицями, що дало змогу забезпечити ефективний обмін даними у системі.

Було розроблено мінімалістичний і зручний дизайн інтерфейсу для покращення сприйняття інформації користувачами. Форми для реєстрації та входу, загальний шаблон вебдодатку, компонент для додавання інтеграцій, сторінка для створення і перегляду листів були ретельно спроектовані для забезпечення інтуїтивно зрозумілого та естетично приємного користувацького досвіду.

В результаті вдалося спроектувати ПЗ. Обрана архітектура та технології дозволяють системі бути стійкою до змін, легко масштабованою і зручною в обслуговуванні, що забезпечує довготривалу продуктивність та стабільність роботи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розроблення бази даних

На цьому етапі можна вважати, що підготовчі процеси для подальшої програмної реалізації завершені. Всі основні етапи, такі як аналіз вимог, вибір архітектури та технологій, а також проектування структур даних і користувацького інтерфейсу, успішно завершені. Це створило підґрунтя для подальшої розробки відповідно до зазначених вимог і технічних умов. Тепер можна приступити до фази програмної реалізації, використовуючи попередньо підготовлені матеріали та концепції.

За допомогою EF Core розпочато імплементацію бази даних та написання моделей для нашого проекту.

Entity Framework використовує міграції для керування змінами в схемі бази даних, коли ви оновлюєте модель даних у вашому коді. Міграції дозволяють зберігати історію змін і застосовувати їх до бази даних у різний час.

Було встановлено необхідні інструменти для роботи з міграціями. За допомогою командного рядку, було встановлено пакет за допомогою команди dotnet add package Microsoft.EntityFrameworkCore.Tools.

Далі було створено необхідні сутності для створення контексту даних.

Сутність Account представляє обліковий запис у системі і містить кілька ключових атрибутів та зв'язків з іншими сутностями. Головними атрибутами є Id, унікальний ідентифікатор типу Guid, що забезпечує унікальність кожного облікового запису, та Login, що зберігає логін користувача. Важливі також Password, який містить пароль користувача, та Marketplace, що визначає тип торгової платформи через ShopType. Поле Token, позначене атрибутом [MaxLength(Int32.MaxValue)], зберігає токен для доступу до системи, а TemporaryToken містить другий тимчасовий токен.Атрибути, що стосуються налаштувань SMTP-сервера, включають SMTPServer, SMTPPort, та прапорець

SMTPUseSSL, який визначає, чи використовувати SSL для з'єднання. Поля ProtocolType, ProtocolAddress, ProtocolPort, та ProtocolUseSSL налаштовують тип протоколу електронної пошти та його параметри. Також зберігаються Name та DisplayName для імені користувача, FailsCount для підрахунку невдалих спроб входу, та прапорці IsBlocked і InProgress для визначення стану облікового запису. Додаткові поля включають FailMessage для збереження повідомлення про помилку, MarketplaceMessagesLastSync для збереження дати останньої синхронізації повідомлень торгової платформи, SendAsName для відображуваного імені відправника, Type для визначення типу інтеграції через IntegrationType, та дати закінчення дії токенів (TokenExpires, TemporaryTokenExpires).Зв'язки з іншими сутностями представлені через навігаційні властивості: Icon вказує на пов'язаний файл типу File, Messages — на колекцію повідомлень, асоційованих з цим обліковим записом, типу ICollection<Message>, а Customers — на колекцію клієнтів, пов'язаних з обліковим записом, типу ICollection<Customer>.

Ця модель забезпечує чітку організацію даних облікових записів у базі даних та дозволяє ефективно керувати інформацією і взаємодією між сутностями у системі.

Код сутності Account.

```
public class Account : IEntity
{
    public Guid Id { get; set; }
    public string Login { get; set; }
    public string Password { get; set; }
    public ShopType Marketplace { get; set; }
    [MaxLength(Int32.MaxValue)]
    public string Token { get; set; }
    public string TemporaryToken { get; set; }
    public string SMTPServer { get; set; }
    public int? SMTPPort { get; set; }
    public bool? SMTPUseSSL { get; set; }
    public EmailProtocolType ProtocolType { get; set; }
    public string ProtocolAddress { get; set; }
    public int? ProtocolPort { get; set; }
    public bool? ProtocolUseSSL { get; set; }
    public string Name { get; set; }
    public string DisplayName { get; set; }
    public int FailsCount { get; set; }
    public bool? IsBlocked { get; set; }
    public bool? InProgress { get; set; }
```

```
        public string FailMessage { get; set; }
        public DateTime? MarketplaceMessagesLastSync { get; set;
        public string SendAsName { get; set; }
        public IntegrationType Type { get; set; }
        public DateTime? TokenExpires { get; set; }
        public DateTime? TemporaryTokenExpires { get; set; }
        public File Icon { get; set; }
        public ICollection<Message> Messages { get; set; }
        public ICollection<Customer> Customers { get; set; }
    }
```

Сутність Message представляє повідомлення у системі і має кілька ключових атрибутів та зв'язків з іншими сутностями.

Головні атрибути включають Id, унікальний ідентифікатор типу Guid, що забезпечує унікальність кожного повідомлення, і Text, текст повідомлення, позначений атрибутом [Searchable] для можливості пошуку. Важливим також є ThreadId, який вказує на потік повідомлень, до якого належить це повідомлення, та MessageDate, що зберігає дату і час створення повідомлення.

Код сутності Message.

```
public class Message : IEntity
{
    public Guid Id { get; set; }
    [Searchable]
    public string Text { get; set; }
    public Guid ThreadId { get; set; }
    public DateTime? MessageDate { get; set; }
    [Searchable]
    public MessageStatus Status { get; set; }
    [Searchable]
    public string Title { get; set; }
    public bool? IsFromCompany { get; set; }
    public Guid? UserId { get; set; }
    public bool? Unread { get; set; }
    public Guid AccountId { get; set; }
    public long? Uid { get; set; }
    [Searchable]
    public string HtmlBody { get; set; }
    public DateTime? SendingDate { get; set; }
    public bool IsDelayed { get; set; }
    public Guid? TemplateId { get; set; }
    public Guid? AutoresponseId { get; set; }
    public Account Account { get; set; }
    public ICollection<Tag> Tags { get; set; }
    [Searchable("Subject", "Folder.Name")]
    public Thread Thread { get; set; }

}
```

Статус повідомлення (Status) та заголовок (Title) також є ключовими атрибутами, які можна шукати. Останній з головних атрибутів — HtmlBody, що

містить HTML-версію тіла повідомлення і теж доступний для пошуку. Сутність має кілька важливих зв'язків. Навігаційна властивість Account вказує на пов'язаний обліковий запис, а Thread вказує на потік повідомлень, до якого належить це повідомлення, з можливістю пошуку за заголовком та іменем папки через атрибут [Searchable("Subject", "Folder.Name")]. Крім цього, є колекція тегів (Tags), пов'язаних з повідомленням, що дозволяє групувати та організовувати повідомлення за категоріями. Ці основні атрибути та зв'язки забезпечують базову функціональність і організацію даних повідомлень у системі, дозволяючи ефективно керувати інформацією та здійснювати пошук.

Сутність Thread представляє потік повідомлень у системі та містить кілька ключових атрибутів та зв'язків з іншими сутностями. Головні атрибути включають Id, унікальний ідентифікатор типу Guid, та CustomerId, ідентифікатор клієнта, з яким пов'язаний цей потік.

Код сутності Thread.

```
public class Thread : IEntity
{
    public Guid Id { get; set; }
    public Guid? ParentThreadId { get; set; }
    public Guid CustomerId { get; set; }
    public ShopType Marketplace { get; set; }
    public string MarketThreadId { get; set; }
    [Searchable]
    public string Subject { get; set; }
    public bool? InActive { get; set; }
    public DateTime? CreatedDate { get; set; }
    public DateTime? LastUpdated { get; set; }
    public Guid? PriorityId { get; set; }
    public PrioritySla Priority { get; set; }
    public ThreadStatus Status { get; set; }
    public long NumberId { get; set; }
    public DateTime? SnoozeTill { get; set; }
    public TicketTypeEnum Type { get; set; }
    public DateTime? FirstAnswer { get; set; }
    public DateTime? ArchiveDate { get; set; }
    [Searchable("CustomerName", "Email")]
    public Customer Customer { get; set; }
    [Searchable("Text")]

    public ICollection<Message> Messages { get; set; }
    public ICollection<Tag> Tags { get; set; }
    public ICollection<Note> Notes { get; set; }
    public ICollection<Task> Tasks { get; set; }
    public ICollection<ThreadAttachment> Attachments { get; set; }
    public ICollection<ChangesLog> ChangesLogs { get; set; }
}
```

Поле ParentThreadId дозволяє організувати ієрархію потоків, вказуючи на батьківський потік, якщо такий існує. Marketplace визначає тип торгової платформи через ShopType, а MarketThreadId зберігає ідентифікатор потоку на цій платформі.Атрибути, що забезпечують додаткову інформацію про потік, включають Subject, тему потоку, яка позначена атрибутом [Searchable] для можливості пошуку, та InActive, прапорець, що вказує на активність потоку. Поля CreatedDate і LastUpdated зберігають дати створення та останнього оновлення потоку відповідно. PriorityId вказує на пріоритет потоку, а Priority представляє цей пріоритет через сутність PrioritySla. Статус потоку зберігається у полі Status, яке використовує тип ThreadStatus. Поле NumberId позначене як пошукове атрибутом [Searchable] і зберігає унікальний номер потоку. Додаткові атрибути включають SnoozeTill для збереження дати тимчасової зупинки обробки потоку, Type для типу квитка через TicketTypeEnum, FirstAnswer для дати першої відповіді та ArchiveDate для дати архівування.Зв'язки з іншими сутностями представлені через навігаційні властивості: Customer, що позначена атрибутом [Searchable("CustomerName", "Email")], вказує на пов'язаного клієнта, а Messages, позначена атрибутом [Searchable("Text")], — на колекцію повідомлень, асоційованих з цим потоком, типу ICollection<Message>. Додаткові зв'язки включають Tags для колекції тегів, Notes для колекції нотаток, Tasks для колекції завдань, Attachments для колекції прикріплень, та ChangesLogs для колекції журналів змін. Ця модель забезпечує чітку організацію даних потоків повідомлень у базі даних та дозволяє ефективно керувати інформацією і взаємодією між сутностями у системі.

Сутність Customer представляє клієнта у системі та містить кілька ключових атрибутів і зв'язків з іншими сутностями. . Перелік атрибутів можна переглянути на рисунку 3.4. Головні атрибути включають Id, унікальний ідентифікатор типу Guid, та CustomerName, ім'я клієнта, яке позначене атрибутом [Searchable] для можливості пошуку. Важливими є також FirstName та LastName, що зберігають відповідно ім'я та прізвище клієнта, і також доступні для пошуку. Поле MarketCustomerId зберігає ідентифікатор клієнта на торговій платформі, а

Email та Phone дозволяють зберігати контактну інформацію клієнта, всі ці поля також позначені як пошукові атрибутом [Searchable].Атрибут Marketplace визначає тип торгової платформи через ShopType, а CreatedDate зберігає дату створення запису про клієнта. Поле AccountId вказує на обліковий запис, з яким пов'язаний цей клієнт.Зв'язки з іншими сутностями представлені через навігаційні властивості: Avatar, що вказує на пов'язаний файл типу File, який може містити зображення аватара клієнта, та Account, що вказує на пов'язаний обліковий запис типу Account. Колекції Threads та Notes представляють відповідно зв'язки з потоками повідомлень (ICollection<Thread>) та нотатками (ICollection<Note>) асоційованими з цим клієнтом. Ця модель забезпечує чітку організацію даних клієнтів у базі даних та дозволяє ефективно керувати інформацією і взаємодією між сутностями у системі.

Код сутності Customer.

```
public class Customer : IEntity
{
    public Guid Id { get; set; }
    [Searchable]
    public string CustomerName { get; set; }
    [Searchable]
    public string FirstName { get; set; }
    [Searchable]
    public string LastName { get; set; }
    [Searchable]
    public string MarketCustomerId { get; set; }
    [Searchable]
    public string Email { get; set; }
    [Searchable]
    public string Phone { get; set; }
    [Searchable]
    public ShopType Marketplace { get; set; }
    public DateTime? CreatedDate { get; set; }
    public Guid? AccountId { get; set;
    public File Avatar { get; set; }
    public Account Account { get; set; }
    public ICollection<Thread> Threads { get; set; }
    public ICollection<Note> Notes { get; set; }
}
```

Наступним етапом у створенні та налаштуванні бази даних є реалізація посередника у вигляді контексту даних. Контекст даних є основним класом, що забезпечує взаємодію з базою даних та відповідає за ініціалізацію з'єднання з базою, встановлення конфігураційних параметрів тощо.

Клас MainContext є посередником між програмою і базою даних. Код контексту можна переглянути на рисунку 3.5. Він успадковується від класу DbContext, що надає можливість взаємодії з базою даних за допомогою Entity Framework Core. Конструктор цього класу приймає параметр DbContextOptionsMainContext options, який вказує на конфігурацію контексту бази даних та передає його до базового класу за допомогою ключового слова base(options). Крім того, в конструкторі викликається метод Initialize(), який може виконати певні дії під час створення контексту.У цьому контексті визначено віртуальні властивості типу DbSet<T>, які представляють колекції сутностей, що відображаються на таблиці бази даних. Наприклад, DbSet<Account> відображає колекцію облікових записів, DbSet<Message> — колекцію повідомлень, і так далі. Ці властивості дозволяють отримувати доступ до даних і виконувати різні операції з ними, такі як додавання, оновлення, видалення тощо. Контекст даних також містить властивості для кожної таблиці бази даних, які є представленням відповідних сутностей. Наприклад, Account — це властивість типу DbSet<Account>, яка представляє колекцію облікових записів. Усі ці елементи дозволяють зручно та ефективно взаємодіяти з базою даних у програмі, спрощуючи роботу з даними та забезпечуючи їхню консистентність.

Код контексту данних.

```
public class ReplycoContext : DbContext
{

    public ReplycoContext(DbContextOptions<ReplycoContext> options) : base(options)
    {
        this.Initialize();
    }

    public virtual DbSet<Account> Account { get; set; }
    public virtual DbSet<Message> Message { get; set; }
    public virtual DbSet<Tag> Tag { get; set; }
    public virtual DbSet<Note> Note { get; set; }
    public virtual DbSet<Task> Task { get; set; }
    public virtual DbSet<Template> Template { get; set; }
    public virtual DbSet<Thread> Thread { get; set; }
    public virtual DbSet<Autoresponder> Autoresponder { get; set; }
    public virtual DbSet<MessageRule> MessageRule { get; set; }
    public virtual DbSet<Folder> Folder { get; set; }
    public virtual DbSet<UserFolders> UserFolders { get; set; }
    public virtual DbSet<UserNotification> UserNotification { get; set; }
    public virtual DbSet<FolderThreads> FolderThreads { get; set; }
    public virtual DbSet<PrioritySla> PrioritySla { get; set; }
```

```
    public virtual DbSet<ChangesLog> ChangesLogs { get; set; }

    public virtual DbSet<SmartFilter> SmartFilters { get; set; }
    public virtual DbSet<Customer> Customer { get; set; }
}
```

За допомогою контексту даних можна легко виконувати різноманітні операції з базою даних за допомогою методів, що він надає. Наприклад, цей контекст дозволяє зручно створювати нові записи в базі даних, вносити зміни до існуючих та оновлювати дані. Взаємодія контексту даних з контролерами, відповідальними за обробку запитів користувача та маніпулювання даними через моделі, відбувається безпосередньо. Крім цього, контекст даних сприяє забезпеченню цілісності даних та розподілу відповідальності між різними компонентами системи.

Після створення моделей даних та контексту бази даних, настає час для створення міграцій. Міграції є механізмом, який дозволяє автоматично оновлювати схему бази даних відповідно до змін у моделях даних без втрати існуючих даних. Під час створення міграцій, Entity Framework Core аналізує зміни в моделях та генерує відповідні SQL-скрипти, які оновлюють структуру бази даних. Це може включати створення нових таблиць, зміну типів даних, додавання чи видалення стовпців тощо. Створення міграцій дозволяє підтримувати структуру бази даних синхронізованою з моделями даних у коді програми. Це робить процес розробки більш простим і безпечним, оскільки зміни в структурі бази даних можуть бути легко відслідковані, перевірені та застосовані за допомогою міграцій.

Крім того, міграції можуть бути використані для автоматичного розгортання змін на продуктивному середовищі, що спрощує процес управління базою даних у реальному часі і зменшує ризик втрати даних чи несумісності структури.

Щоб виконати міграції необхідно в консольному рядку виконати наступні команди.

dotnet ef migrations add InitialCreate
dotnet ef database update

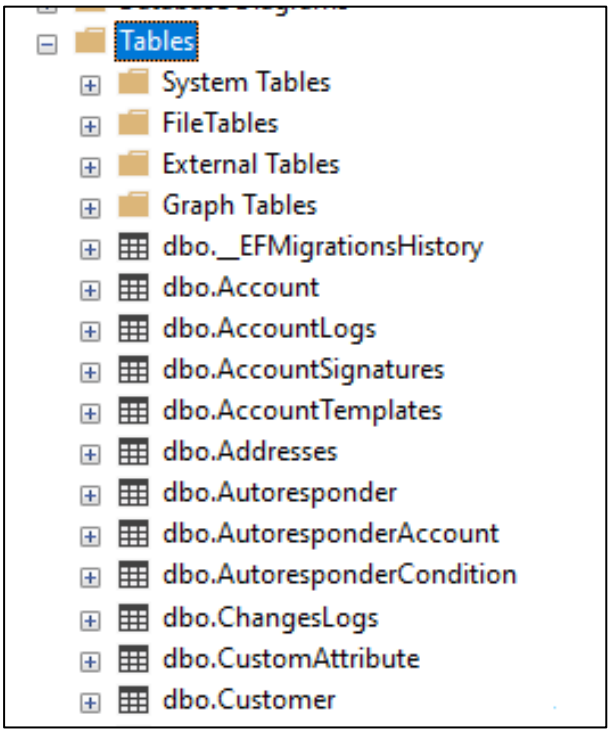


Рисунок 3.1 – Створені таблиці

3.2 Розроблення програмних модулів

На даному етапу розпочато роботу над першим модулем вебзастосунку, який включає функціонал реєстрації та входу користувачів. Цей модуль є критично важливим, оскільки він забезпечує доступ користувачів до системи та управління їхніми обліковими записами.

У попередніх розділах реалізації було визначено основні вимоги до функціоналу та створено загальну архітектуру модуля, яка включає фронтенд та бекенд компоненти.

Перш за все необхідно встановити Node.js та npm (Node Package Manager). Це необхідні інструменти для роботи з React та керування пакетами.

Далі необхідно створити новий проект з Create React App. Це інструмент, що дозволяє швидко налаштувати середовище для розробки React-додатків без потреби в додаткових конфігураціях.

Після створення проекту він матиме наступну структуру файлів і директорій зображену на рисунку 3.2.

Рисунок 3.2 – Початкова структура проекту

Для керування маршрутами в React-додатку використовується бібліотека React Router. Імпортуємо всі необхідні елементи з бібліотеки, а також вказуємо шляхи до компонент застосунку і роути по яких можна отримати певну сторінку. нижче наведений код роутера.

```
class Router extends React.PureComponent {
    static propTypes = {
        isLoggedIn: PropTypes.bool
    };

    static defaultProps = {
        isLoggedIn: false
    };

    constructor(props) {
        super(props);
    }

    get routes() {
        return [{
            route: ROUTES.signIn(),
            component: SignIn
        }, {
            route: ROUTES.signInWithToken(),
            component: SignIn,
            key: 't'
        }, {
            route: ROUTES.signUp(),
            component: SignUp
        }, {
            route: ROUTES.forgotPassword(),
```

```
                component: ForgotPassword
            }, {
                route: ROUTES.resetPassword(),
                component: ResetPassword,
                key: 'accessToken'
            }, {
                route: ROUTES.adminSignIn(),
                component: AdminSignIn,
                key: 'accessToken'
            }, {
                route: ROUTES.verifyCompany(),
                component: VerifyCompany,
                key: 'token'
            }];
        }

        get content() {
            const { showChangeBrowser } = this.state;
            const { isLoggedIn } = this.props;
            if (isMobile) {
                return (
                    <Switch>
                        {_.map(this.mobileRoutes, Shared.renderCustomRoute)}
                        <Redirect to={ROUTES.signUp()} />
                    </Switch>
                );
            }

            if (!isChrome && !isFirefox && showChangeBrowser) {
                return (
                    <ChangeBrowserProposition onClose={this.closeChangeBrowser} />
                );
            }


            return !isLoggedIn ? (
                <Switch>
                    {_.map(this.routes, Shared.renderCustomRoute)}
                    <Redirect to={ROUTES.signIn() + history.location.search} />
                </Switch>
            ) : (
                <Route
                    path="*"
                    render={this.renderAuthorizedRoute}
                />
            );
        }
```

Далі створюємо глобальний стан застосунку за допомогою Redux оголошуючи стор і редюсери. Код стору наведено нижче.

```
const history = createBrowserHistory();
const routeMiddleware = routerMiddleware(history);
let middleware = [thunk, routeMiddleware];

const composeEnhancers = isDevelopmentMode &&
window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ ?
    window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ : compose;
```

```
const store = createStore(
    rootReducer,
    composeEnhancers(applyMiddleware(...middleware))
);

const persistor = persistStore(store, null);

export { store, persistor, history };
```

Код рут редюсеру наведено нижче.

```
const persistConfig = {
    key: 'root',
    storage: ReduxPersistStorage,
    blacklist: [
        routerConstants.NAME,
        coreConstants.NAME,
        authConstants.NAME,
        threadsConstants.NAME,
        toolsConstants.NAME,
        integrationsConstants.NAME,
        accountConstants.NAME,
        messagesConstants.NAME,
        adminConstants.NAME,
        notificationsConstants.NAME
        dashboardConstants.NAME,
        notesConstants.NAME,
    ]
};


const rootReducer = combineReducers({
    [routerConstants.NAME]: routerReducer,
    [coreConstants.NAME]: coreReducer,
    [authConstants.NAME]: authReducer,
    [threadsConstants.NAME]: threadsReducer,
    [toolsConstants.NAME]: toolsReducer,
    [integrationsConstants.NAME]: integrationsReducer,
    [accountConstants.NAME]: accountReducer,
    [messagesConstants.NAME]: messagesReducer,
    [adminConstants.NAME]: adminReducer,
    [notificationsConstants.NAME]: notificationsReducer,
    [dashboardConstants.NAME]: dashboardReducer,
    [notesConstants.NAME]: notesReducer
});

export default persistReducer(persistConfig, rootReducer)
```

Створивши маршрутизацію і глобальний стан застосунку можна перейти до реалізації основних сторінок додатку. Першими реалізовано сторінки для авторизації користувача в системі, а сама реєстрація і вхід. Було створено верстку сторінок і створено обробку подій на сторінках, а саме взаємодія з API.

Код сторінки реєстрації наведено нижче.

```
class SignUp extends React.PureComponent {

    constructor(props) {
        super(props);
        this.useShopifyBilling = !!localStorage.getItem('shopifyPreinstall');
        const { country, appLanguage, region, phonePrefix } =
AppSettingsService.getDefaultAppOnAuth();
        this.state = {
            email: props?.emailFromUrl || '',
            password: '',
            fullName: '',
            companyName: '',
            workPhone: '',
            country,
            appLanguage,
            phonePrefix,
            region,
            errors: {},
            isVisibleModalAfterSocialSinIn: false
        };
        this.onChangeAppLanguage(appLanguage);
    }

   return (
            <Core.SpinnerWrapper isLoading={isRegistering}
tip={i18n.t('auth.registeringYourWorkspace')}>
                <AuthTemplate hasChangedOrder={true} hasCloseButton={true}>
                    <AuthBlock
                        title={i18n.t('auth.signUp')}
                        subtitle={i18n.t('auth.signUpDescription')}
                        footer={(
                            <>
                                {`${i18n.t('auth.alreadyHaveAnAccount')} `}
                                <Core.Link
                                    text={i18n.t('auth.signInHere')}
                                    route={ROUTES.home()}
                                />
                            </>
                        )}>
                        <Core.SelectSecondary
                            name={FIELDS.country}
                            data={this.countryList}
                            value={country}
                            placeholder={i18n.t('settings.company.country')}
                            onChange={this.onChangeCountry}
                            hasRoundBorder={true}
                            hasMargins={true}
                        />
                        <Core.InputSecondary
                            placeholder={i18n.t('auth.fullName')}
                            value={fullName}
                            name={FIELDS.fullName}
                            onChange={this.onChangeField}
                            autoFocus={true}
                            id="signup_fullname"
                            errorMessage={errors[FIELDS.fullName]}
                            onHideError={this.onHideError}
                        />
                                                {this.passwordInput}
                        <Core.InputPhoneNumberSecondary
```

```
                                placeholder={i18n.t('auth.phone')}
                                value={workPhone}
                                name={FIELDS.workPhone}
                                onChange={this.onChangeField}
                                id="signup_phone"
                                errorMessage={errors[FIELDS.workPhone]}
                                onHideError={this.onHideError}
                                country={phonePrefix}
                            />
                            <Core.Modal
                                isVisible={isVisibleModalAfterSocialSinIn}
                                footer={null}
                                onClose={this.onCloseModalAfterSocialSignIn}>
                                <div className={css(styles.modal)}>

                                    <p
className={css(styles.modalHeader)}>{i18n.t('auth.enterPasswordAfterSocialSingIn')
}</p>
                                    {this.passwordInput}
                                    {this.confirmSignUpButton}
                                </div>
                            </Core.Modal>
                        </AuthBlock>
                        {this.hasEnteredData(this.state) ?
                            <Prompt
                                message={() => i18n.t('accounts.confirmLeave')}
                            /> : null
                        }
                    </AuthTemplate>
                </Core.SpinnerWrapper>
            );
        }
    }
```

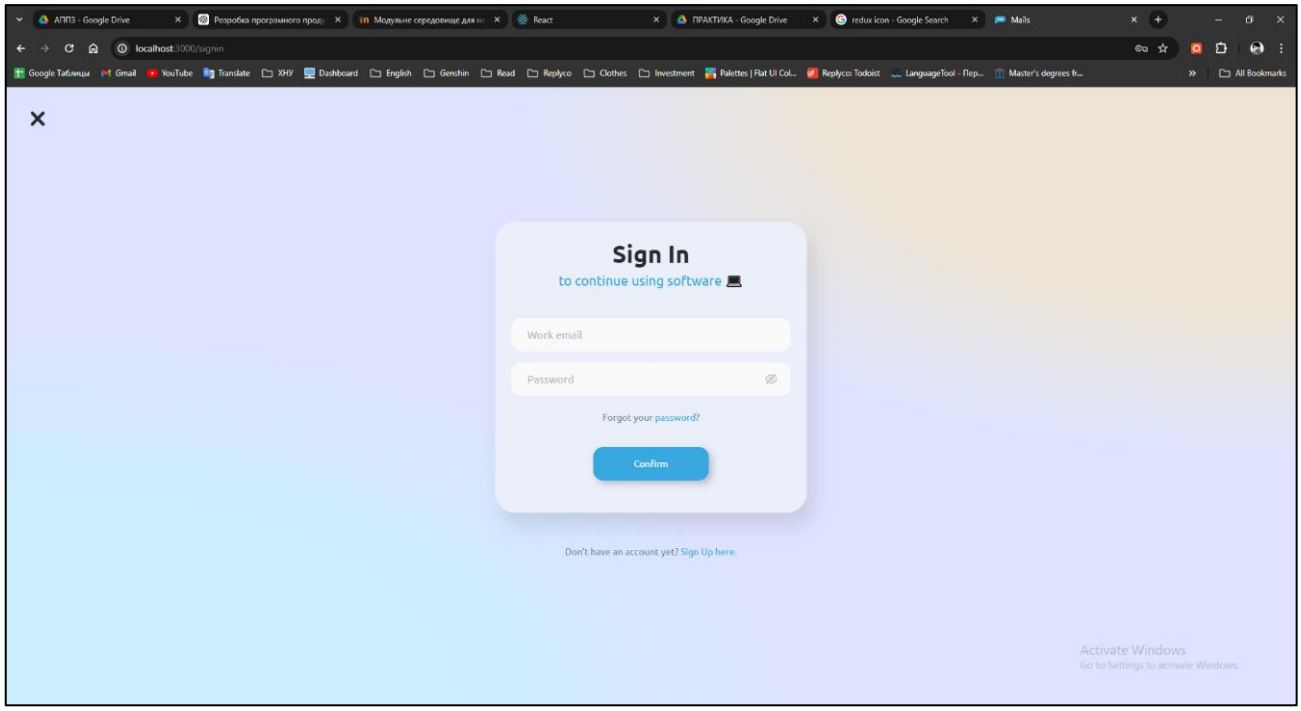


Рисунок 3.3 – Сторінка входу

Код сторінки входу наведено нижче.

```
class SignIn extends React.PureComponent {
    constructor(props) {
        super(props);
        this.state = {
            email: '',
            password: '',
            code: '',
            redirectingToShopify: false,
            errors: {}
        };
        const parsed = queryString.parse(props.location.search);
        const shop = _.get(parsed, 'shop');
        if (_.get(parsed, 'hmac') && shop && _.get(parsed, 'timestamp') &&
!_.get(parsed, 'code') && !_.get(parsed, 'session')) {
            this.state.redirectingToShopify = true;
            localStorage.setItem("shopifyPreinstall", props.location.search);

window.location.replace(`https://${shop}/admin/oauth/authorize?client_id=8fa1a2606b00063
cda5a1fb123e45f3d&scope=read_customers,read_orders,read_products,read_draft_orders,read_
product_listings&redirect_uri=https://app.replyco.com/integrations/shopify/success`);
        }
        /*const linnworksToken = _.get(parsed, 'token');
        if (linnworksToken) {
            this.checkLinnworksToken(linnworksToken);
        }*/ // disabled auto login temporarily as per support team request
        this.setAppLanguage();
    }

        return (
            <SpinnerWrapper isLoading={this.state.redirectingToShopify}
tip={i18n.t('auth.authorizingWithShopify')}>
                <AuthTemplate hasCloseButton={!token}>
                    {token ?
                        <SignInFromInvite
                            token={token}
                        /> :
                        <AuthBlock
                            title={i18n.t('auth.logInToReplyco')}
                            subtitle={i18n.t('auth.signInDescription')}
                            footer={
                                <>
                                    {`${i18n.t('auth.doNotHaveAccountYet')} `}
                                    <Link
                                        text={i18n.t('auth.signUpHere')}
                                        route={ROUTES.signUp()}
                                    />
                                </>
                            }>
                            {!authenticatorToken?.length ?
                                <>
                                    <InputSecondary
                                        placeholder={i18n.t('auth.email')}
                                        value={email}
                                        name={FIELDS.email}
                                        onChange={this.onChangeField}
                                        autoFocus={true}
                                        errorMessage={errors[FIELDS.email]}
                                        onHideError={this.onHideError}
```

```
                                    />
                                    <PasswordInputSecondary
                                        placeholder={i18n.t('auth.password')}
                                        value={password}
                                        name={FIELDS.password}
                                        onChange={this.onChangeField}
                                        onPressEnter={this.onSignIn}
                                        errorMessage={errors[FIELDS.password]}
                                        onHideError={this.onHideError}
                                    />
                                    <p className={css(styles.text)}>
                                        {i18n.t('auth.forgotYourPassword.part1')}
                                        <Link

text={i18n.t('auth.forgotYourPassword.part2')}
                                            route={ROUTES.forgotPassword()}
                                        />
                                        {i18n.t('auth.forgotYourPassword.part3')}
                                    </p>

                                    <ButtonSecondary
                                        label={i18n.t('confirm')}
                                        onPress={this.onSignIn}
                                        isLoading={isAuthorizing}
                                        isDisabled={isAuthorizing}
                                        className={styles.button}
                                    />

                                    {/* <SocialSignIn
                                        onSuccess={this.onSocialLogin}
                                        separatorText={i18n.t('auth.orSignInWith')}
                                    /> */}
                                </> :
                                <>
                                    <InputSecondary
                                        placeholder={i18n.t('auth.verificationCode')}
                                        value={code}
                                        name={FIELDS.code}
                                        onChange={this.onChangeField}
                                        autoFocus={true}
                                        errorMessage={errors[FIELDS.code]}
                                        onHideError={this.onHideError}
                                        onPressEnter={this.onVerifyCode}
                                    />
                                    <ButtonSecondary
                                        label={i18n.t('confirm')}
                                        onPress={this.onVerifyCode}
                                        isLoading={isVerifyingSignInCode}
                                        isDisabled={isVerifyingSignInCode}
                                        className={styles.button}
                                    />
                                </>
                            }
                        </AuthBlock>
                    }
                </AuthTemplate>
            </SpinnerWrapper>
        );
    }
}
```

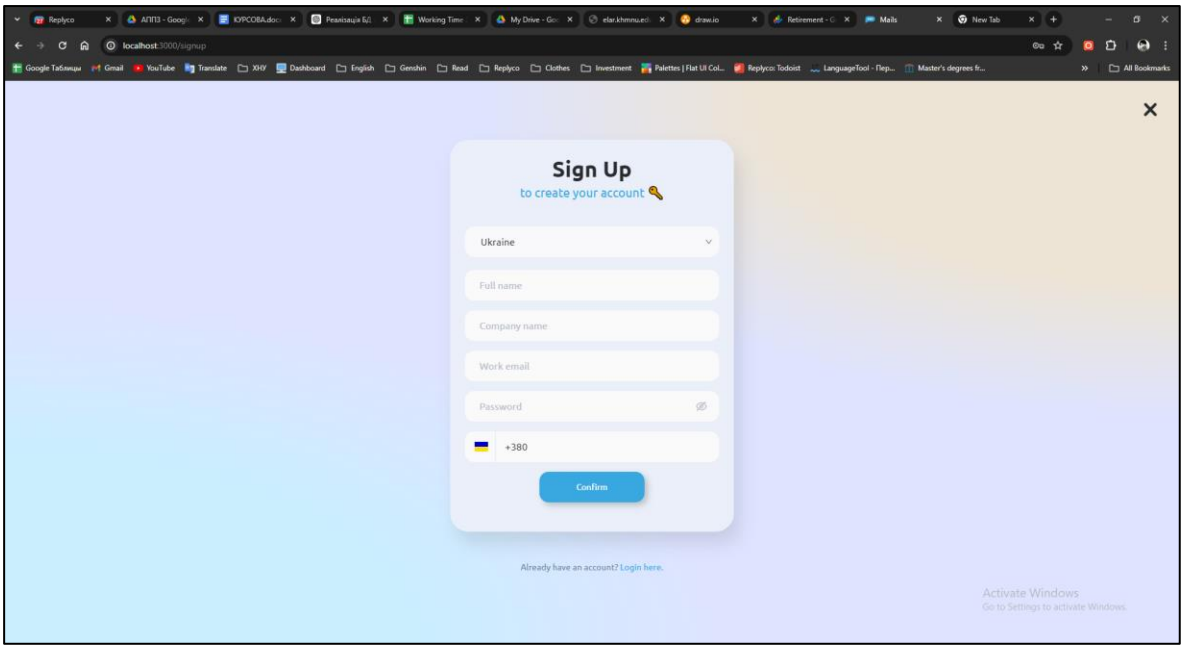


Рисунок 3.4 – Сторінка реєстрації

Бекенд частина застосунку була також детально опрацьована. Було створено необхідні моделі даних та налаштовано контекст бази даних для управління обліковими записами користувачів. Реалізовано основні API-методи для реєстрації, входу, виходу та відновлення пароля. Для забезпечення безпеки використовуються сучасні методи шифрування паролів та аутентифікації користувачів. Вся логіка взаємодії між фронтендом і бекендом ретельно тестується для забезпечення надійної роботи модуля та захисту даних користувачів.

Нижче наведено контролер для обробки запитів на API для авторизації користувача в системі.

```
public class AuthController : Controller
{

    [HttpPost("registration")]
    public async Task<IActionResult> Registration([FromBody]
CompanyRegistrationModel registrationModel)
    {
        var result = await _authManager.RegisterCompany(registrationModel);
        if (result.Success)
        {
            var companyId = result.User.CompanyId;
```

```
                    await _signUpService.CreateDbUser(companyId, registrationModel.Region,
!result.IsUsedPreparedDb);



                    try
                    {
                        await
_verifyCompanyTicketManager.CreateVerifyTicketAsync(result.User);
                    }
                    catch (Exception e) {
                        _logger.LogError($"Error when creating verify email: {e.Message} -
------- {result.User.CompanyId}");
                    }

                    var res = GetResult(result.User.Id, result.Identity.Claims);
                    Response.Cookies.Append(AppConstants.EmailCookieName,
$"{result.User.Email}");
                    var principal = new ClaimsPrincipal(result.Identity);
                    await
HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, principal);
                    res.RefreshToken = result.User.RefreshToken.Find(x => x.DeviceId ==
registrationModel.DeviceId)?.RefreshToken;
                    return Json(res);
                }
                return BadRequest();
            }


        [HttpPost("token")]
           [ApiExplorerSettings(GroupName = "integration")]
           public async Task<IActionResult> Token([FromBody] TokenSignInModel model)
           {
               if (model == null)
               {
                   return BadRequest("Please fill login form.");
               }
               AuthModel authModel;
               switch (model.GrantType)
               {

                   case "password":
                       {
                           authModel = await _authManager.SignIn(model);

                           break;
                       }
                   case "refresh_token":
                       {
                           authModel = await _authManager.RefreshToken(model);
                           break;
                       }
                   default:
                       {
                           return BadRequest("Cannot find this type of login.");
                       }
               }

               if (authModel.Success)
               {
                   var user = await _userManager.GetAsync(authModel.User.Id);
```

```
                    if (!string.IsNullOrEmpty(model.ip))
                    {
                        var location = await IPLocationService.GetIpLocation(model.ip);
                        if (location != null && (location.country_code2 == "RU" ||
location.country_code2 == "BY"))
                        {

                    if (model.GrantType == "password")
                    {
                        var response2FA = await VerifyUserOn2FA(user);
                        if (response2FA != null)
                            return Json(response2FA);
                    }



                    return await ProcessUserSignIn(user.Id, authModel, model.DeviceId,
model.ip);
                }
                return BadRequest(authModel.Message);
            }
        }
```

Було розроблено головний макет замтосунку, який забезпечує зручну навігацію та доступ до основних функцій вебзастосунку. Головне меню містить кілька основних пунктів, кожен з яких відповідає за певну частину системи. Перший пункт меню — "Dashboard" — надає загальний огляд системи, включаючи ключові показники та статистику, що дозволяє користувачам швидко отримати необхідну інформацію.

Другий пункт меню — "Tickets" — містить кілька підпунктів для кращої організації та доступу до різних типів тікетів.

Серед підпунктів є "Not resolved (Не вирішені)", "My Tickets (Мої тікети)", "Unassigned (Без відповідального користувача)", "Resolved (Вирішені)", "All Tickets (Всі тікети)", "Blocked (Заблоковані)", "Delayed (Привідкладені)", "Snozed (Заморожені)" та "Archive (Архів)". Така структура дозволяє користувачам легко знаходити та управляти тікетами в залежності від їхнього статусу, що значно покращує ефективність роботи з системою.

Третій пункт меню — "Labels (Папки)" — призначений для організації та зберігання різних документів та файлів у структурованому вигляді. Ця функція дозволяє користувачам створювати та управляти папками, що полегшує доступ до необхідних даних і забезпечує зручне їх групування.

Четвертий пункт меню — "Tools (Інструменти)" — містить кілька підпунктів, таких як "Automation (Автоматизація)", "Customers (Клієнти)", "Labels (Папки)" та "Smart Filters (Розумні фільтри)". Ці інструменти надають користувачам можливість налаштовувати автоматизовані процеси, управляти інформацією про клієнтів, працювати з папками та створювати розумні фільтри для швидкого пошуку інформації. Останній пункт меню — "Адмін" — призначений для адміністраторів системи, надаючи їм доступ до різних налаштувань та функцій управління системою.

Код розмітка головного макету наведено нижче.

```
class Sidebar extends React.PureComponent {
        render() {
        const { isCollapsedMenu, hasNoCompletedOnboardingChecklist } = this.props;
        return (
            <Layout.Sider
                trigger={null}
                collapsible={true}
                collapsed={isCollapsedMenu}
                width={LAYOUT.SIDEBAR_WIDTH}
                collapsedWidth={LAYOUT.SIDEBAR_WIDTH_COLLAPSED}
                className={css(common.printDisabled, styles.container)}>
                <SidebarLogo />
            </Layout.Sider>
        );
    }
}

class SidebarMenu extends React.PureComponent {
    render() {
        const { menuOpenKeys, isCollapsedMenu } = this.props;
        return (
            <div className={css(styles.mainContainer)}>
                <Menu
                    mode={isCollapsedMenu ? "vertical" : "inline"}
                    openKeys={menuOpenKeys}
                    inlineCollapsed={isCollapsedMenu}
                    triggerSubMenuAction="click"
                    className={css(styles.container)}
                    subMenuCloseDelay={0}>
                    <SidebarDashboardMenu />
                    <SidebarTicketsMenu />
                    <SidebarLabelsMenu />
                    <SidebarToolsMenu />
                    <SidebarAdminMenu />
                </Menu>
            </div>
        );
    }
}
)(SidebarMenu));
```

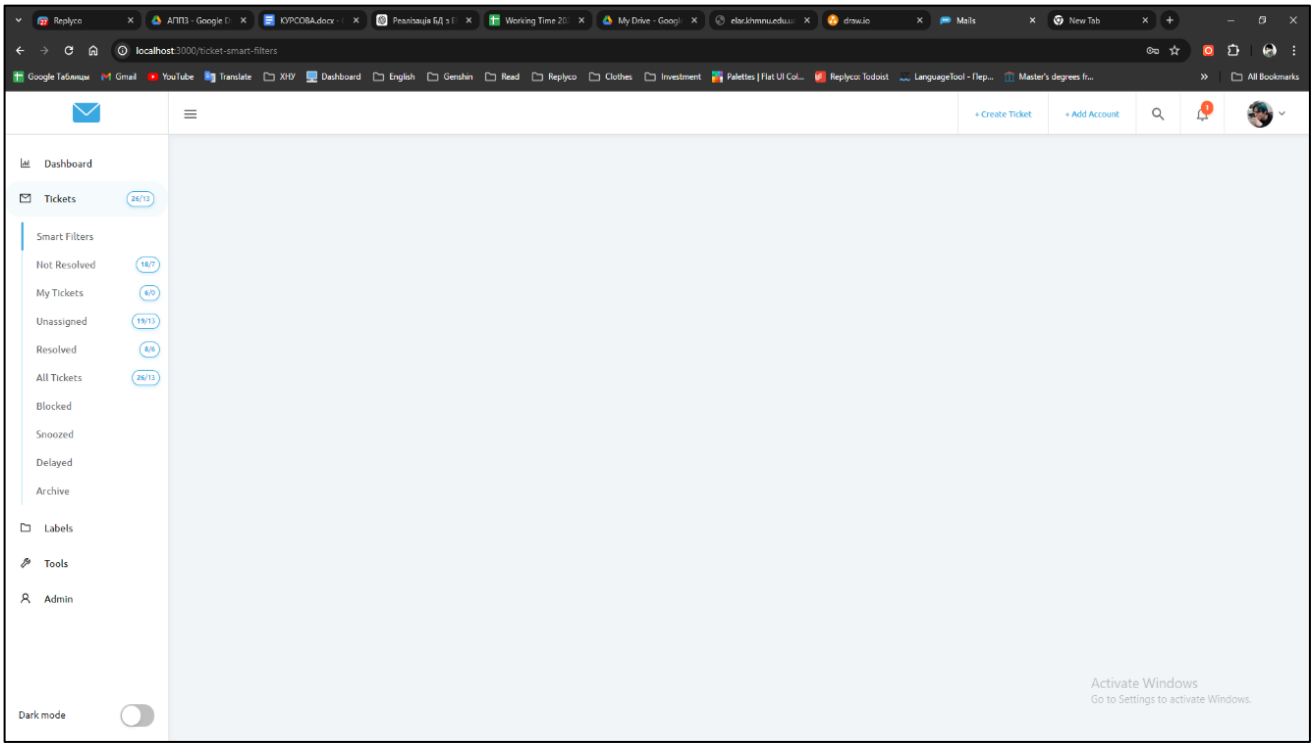


Рисунок 3.5 – Загальний макет сторінки

Було розроблено таблицю тікетів, яка забезпечує зручний перегляд та управління всіма тікетами в системі. Таблиця містить кілька важливих колонок, що дозволяють швидко знайти необхідну інформацію про тікети.

Основні колонки включають "Ticket ID" (ID Тікета), "Label" (Мітка), "Source" (Джерело), "Account" (Обліковий запис), "Customer" (Клієнт), "Message Number" (Номер повідомлення), "Subject" (Тема), "Assigned User" (Призначений користувач), "Respond Till" (Час відповіді), "Resolve Till" (Час вирішення), "Created Date" (Дата створення), "Last Message Date" (Дата останнього повідомлення) та "Language" (Мова).

Кожна з цих колонок відіграє важливу роль у забезпеченні ефективного управління тікетами. Колонка "ID Тікета" дозволяє легко ідентифікувати кожен тікет, тоді як "Мітка" допомагає класифікувати тікети за певними категоріями. "Джерело" вказує на те, звідки надійшов тікет, а "Обліковий запис" показує, до якого облікового запису він належить. Колонка "Клієнт" містить інформацію про клієнта, що створив тікет. "Тема" надає короткий опис змісту тікета, а "Призначений користувач" показує, хто відповідає за його обробку.

Для зручності користувачів було реалізовано функцію пагінації, яка дозволяє переглядати тікети по 10 елементів на сторінку, що забезпечує легкий та швидкий доступ до необхідної інформації.

Код розмітки сторінки подано нижче.

```
class ThreadTable extends React.PureComponent {
    state = {
        contextMenuThreadId: null,
        contextMenuX: null,
        contextMenuY: null,
        mouseOverThreadId: null
    };

    get columns() {
        const columns = [];
        const { i18n, sorter, threadsColumns, isVisibleSnoozeTillColumn,
isVisibleDelayedUntillColumn,
            hasAiFeature, companyAiCategoriesEnabled } = this.props;
        if (threadsColumns[THREAD_TABLE_COLUMNS_ENUM.numberId]) {
            columns.push(
                {
                    title: i18n.t('threads.tableColumns.numberId'),
                    key: THREAD_TABLE_COLUMNS.numberId,
                    sorter: true,
                    dataIndex: THREAD_TABLE_COLUMNS.numberId,
                    render: this.renderItem,
                    sortOrder: sorter && sorter.columnKey ===
THREAD_TABLE_COLUMNS.numberId && sorter.order
                }
            );
        }
        if (threadsColumns[THREAD_TABLE_COLUMNS_ENUM.folder]) {
            columns.push(
                {
                    title: i18n.t('threads.tableColumns.label'),
                    key: THREAD_TABLE_COLUMNS.folder,
                    render: this.renderFolder
                }
            );
        }
        if (threadsColumns[THREAD_TABLE_COLUMNS_ENUM.aiCategories] && hasAiFeature
&& companyAiCategoriesEnabled) {
            columns.push(
                {
                    title: i18n.t('threads.tableColumns.aiCategories'),
                    key: THREAD_TABLE_COLUMNS.aiCategories,
                    render: this.renderAiCategories
                }
            );
        }
        if (threadsColumns[THREAD_TABLE_COLUMNS_ENUM.marketplace]) {
            columns.push(
                {
                    title: i18n.t('threads.tableColumns.source'),
                    key: THREAD_TABLE_COLUMNS.marketplace,
                    render: this.renderImage
```

```
                    }
                );
            }
            if (threadsColumns[THREAD_TABLE_COLUMNS_ENUM.lastMessageDate]) {
                columns.push(
                    {
                        title: i18n.t('threads.tableColumns.lastMessageDate'),
                        key: THREAD_TABLE_COLUMNS.lastMessageDate,
                        sorter: true,
                        dataIndex: THREAD_TABLE_COLUMNS.lastMessageDate,
                        render: toLocalTime.toUserFriendlyFormat,
                        sortOrder:      sorter      &&      sorter.columnKey      ===
THREAD_TABLE_COLUMNS.lastMessageDate && sorter.order
                    }
                );
            }
            if                    (isVisibleDelayedUntillColumn                    &&
threadsColumns[THREAD_TABLE_COLUMNS_ENUM.delayedDate]) {
                columns.push(
                    {
                        title: i18n.t('threads.tableColumns.delayedUntil'),
                        key: THREAD_TABLE_COLUMNS.delayedDate,
                        render: this.renderDelayDate
                    }
                );
            }
            return columns;
        }

        render() {
            const  {  threads,  className,  initialColumnsWidth,  initialColumnsOrder,
isResetInitialColumnSettings,
                getPaginationModel, hasAiFeature, ...props } = this.props;
            const { contextMenuThreadId, contextMenuX, contextMenuY } = this.state;
            return (
                <>
                    <TableResizable
                        {...props}
                        onPressRow={this.onMouseClick}
                        columns={this.columns}
                        data={threads}
                        className={className || styles.table}
                        rowClassName={this.generateTableRowStyles}
                        initialColumnsWidth={initialColumnsWidth}
                        saveColumnsWidth={this.saveColumnsWidth}
                        initialColumnsOrder={initialColumnsOrder}
                        defaultColumnsOrder={THREAD_DEFAULT_COLUMNS_ORDER}
                        saveColumnsOrder={this.saveColumnsOrder}
                        resetInitialColumnSettings={isResetInitialColumnSettings}
                        onContextMenu={this.onContextMenu}
                        onMouseEnter={hasAiFeature && this.onMouseOver}
                        onMouseLeave={hasAiFeature && this.onMouseLeave}
                    />
                    {contextMenuThreadId ?
                        <ThreadTableContextItem
                            threadId={contextMenuThreadId}
                            positionX={contextMenuX}
                            positionY={contextMenuY}
                            onClose={this.onCloseContextMenu}
                            threadsParams={getPaginationModel()}
                        /> : null
```

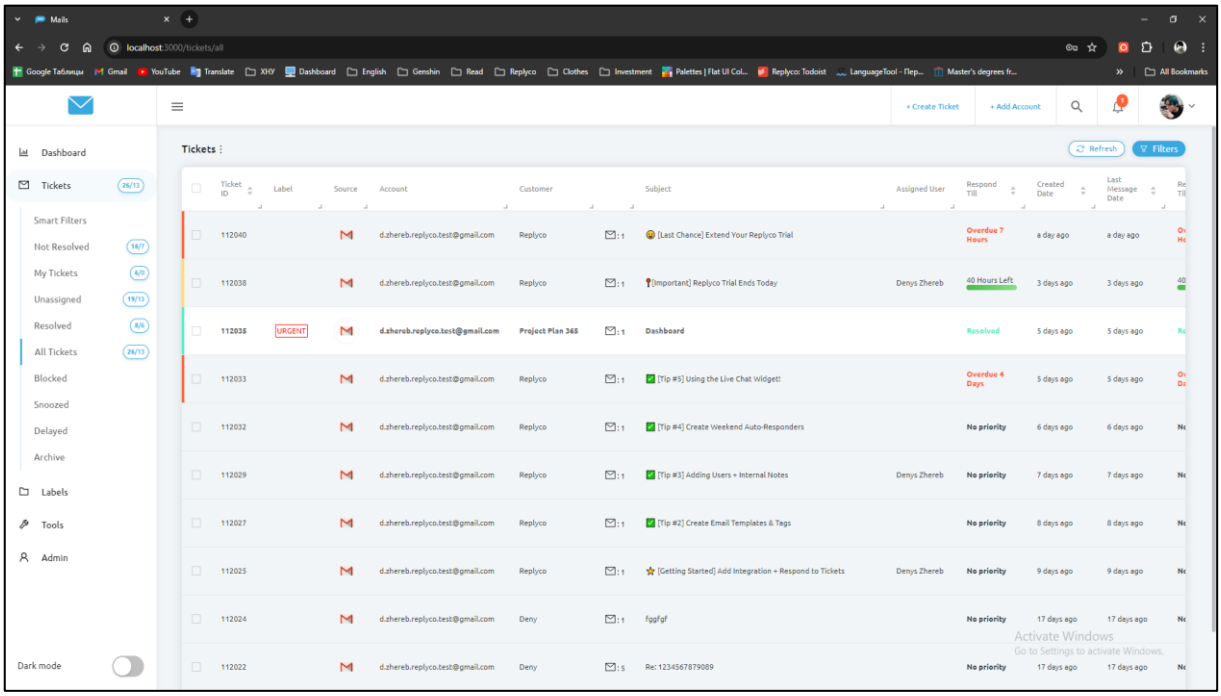


Рисунок 3.6 – Таблиця тікетів

Було розроблено сторінку перегляду тікету, яка дозволяє детально ознайомитися з усією інформацією, представленою у таблиці тікетів. На цій сторінці користувач може побачити повну інформацію про тікет, включаючи "ID Тікета", "Мітку", "Джерело", "Обліковий запис", "Клієнта", "Номер повідомлення", "Тему", "Призначеного користувача", "Час відповіді", "Час вирішення", "Дату створення", "Дату останнього повідомлення" та "Мову". Ця інформація дозволяє користувачам отримати повне уявлення про поточний стан та історію тікета.

На сторінці перегляду тікету також реалізована можливість написати відповідь на лист, що дозволяє оперативно реагувати на запити клієнтів. Форма для відповіді забезпечує всі необхідні функції для зручного створення та відправки повідомлень, включаючи форматування тексту та додавання вкладень. Це значно полегшує процес комунікації з клієнтами та забезпечує високий рівень обслуговування.

Крім того, на даній сторінці користувач може виконати певні дії з тікетом. Наприклад, можна призначити відповідального користувача для обробки тікету, що допомагає розподіляти роботу серед членів команди. Також можна заморозити

тікет, якщо його обробка тимчасово призупинена, або встановити та змінити SLA (Service Level Agreement), що визначає часові рамки для вирішення тікета. Ці функції допомагають ефективно керувати робочими процесами та забезпечувати виконання зобов'язань перед клієнтами.

На сторінці перегляду тікету також можна переглянути детальну інформацію про клієнта, що створив тікет. Ця інформація включає контактні дані клієнта, історію попередніх запитів та взаємодій, що допомагає краще розуміти потреби клієнта та забезпечувати більш персоналізоване обслуговування. Наявність повної інформації про клієнта безпосередньо на сторінці тікета дозволяє швидко і ефективно вирішувати його питання, підвищуючи загальний рівень задоволеності клієнтів.

Код сторінки наведено нижче.

```
class ThreadDetails extends React.PureComponent {
        constructor(props) {
        super(props);
        this.browser = detect();
        this.endContainerRef = React.createRef();
        this.scrollContainerRef = React.createRef();
        this.state = {
            isVisibleComposeMail: false,
            isScrolledToBottom: false,
            orderByAscending:                                _.has(props.currentUser,
'orderOfMessagesByAscending') ?
                !!props.currentUser?.orderOfMessagesByAscending : null,
            replyMessage: null,
            isVisibleAddNote: false,
            isVisibleOnlyNotes: false,
        };
        if (!this.props.thread) {
            this.props.actions.getThreadSimplified(this.props.threadId);
        } else {
            this.initData();
        }
    }

        render() {
        const      {      thread,      threadId,      isThreadDetailsOnboarding,
isThreadDetailsRespondOnboarding } = this.props;
        const { isVisibleComposeMail, isScrolledToBottom } = this.state;
        return !thread ? (
            <SkeletonTheme isInversed={true}>
                <div className={css(styles.skeletonContainer)}>
                    {this.renderMessagesSkeletons}
                    <div className={css(styles.skeletonContainerRight)}>
                        {_.times(4, this.renderRightPanelSkeletonItem)}
                    </div>
                </div>
```

```
                </SkeletonTheme>
            ) : (
                <LayoutWithRightPanel                    right={this.renderRightPanel}
left={this.renderLeftPanel}>
                    <ThreadDetailsHeader
                        threadId={threadId}
                        onClickResponce={this.onClickResponse}
                        onRefresh={this.onRefresh}
                    />
                    {this.renderMessagesContent}
                    {isThreadDetailsOnboarding ?
                        <OnboardingThreadDetailsTutorial /> : null
                    }

                </LayoutWithRightPanel>
            );
        }
    }
```

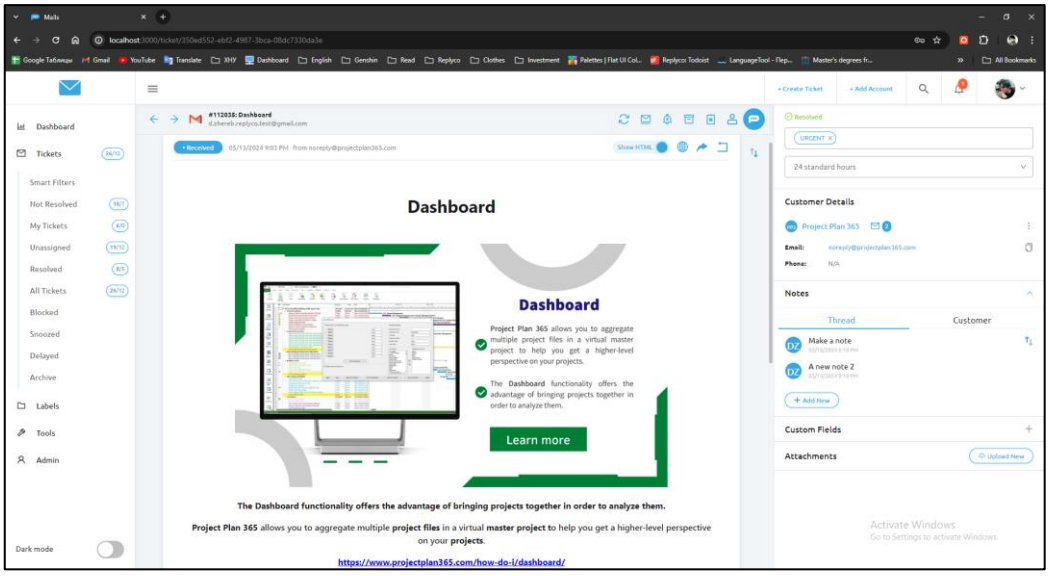


Рисунок 3.7 – Перегляд тікету

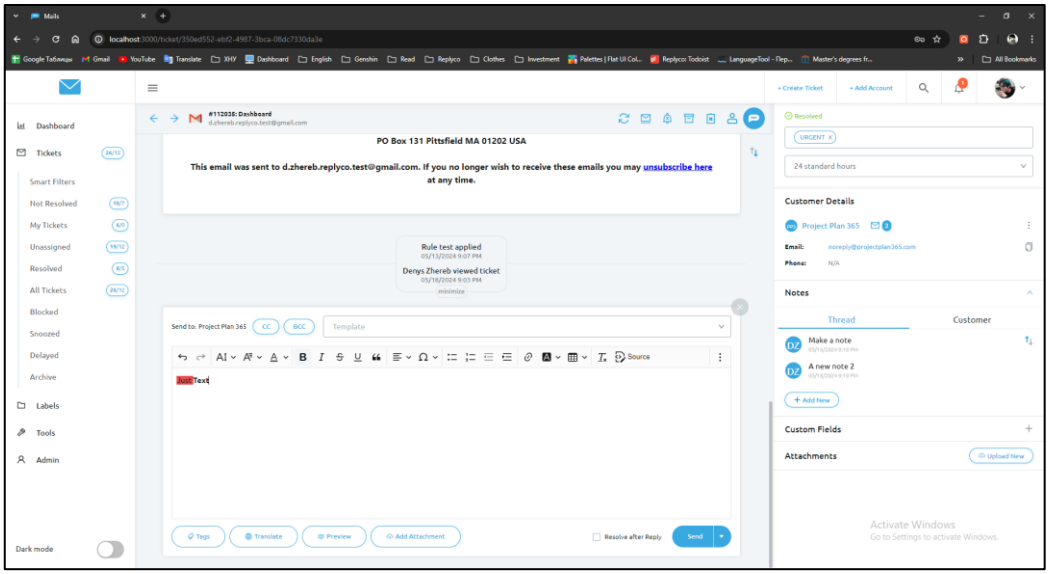


Рисунок 3.8 – Створення листа

Для взаємодії з бекендом було використано бібліотеку axios. Було створено загальну конфігурацію для всіх запитів, а також апі модуль з запитами на бекенд.

```
const configAxios = store => {
    axios.defaults.baseURL = coreConstants.BASE_URL;
    axios.defaults.headers.common['Content-Type'] = 'application/json;
charset=utf-8';
    //axios.defaults.headers.common.Pragma = 'no-cache';

    axios.interceptors.request.use(async config => {
        let updatedConfig = updateThreadContextHeader(config, store.getState());
        updatedConfig = { ...updatedConfig, cancelToken:
cancelTokenService.getCancelToken(updatedConfig.url) };
        return updateAuthorizationHeader(updatedConfig, accessToken);
    }, error => Promise.reject(error));

    axios.interceptors.response.use(response => {
        cancelTokenService.clearCancelToken(response.config.url);
        return response;
    },
    error => {
        if (error && _.get(error, "response.status") === 401 && error.config.baseURL
!== LINNWORKS_APPLICATION_FULL_NAME_URL) {
            store.dispatch(actions.signOut());
            throw new Error(coreConstants.ERROR_TYPES.UNAUTHORIZED);
        }
        throw error;
    }
    );
};
```

Код виконання запитів на бекенд.

```
import axios from 'axios';

export const endpoints = {
    signIn: () => 'auth/token',
    verifySignInCode: () => 'auth/verifyCode',
    signUp: () => 'auth/registration',
    forgotPassword: () => 'restore/token',
    restorePassword: () => 'restore/restorePassword',
    checkResetPasswordTicket: () => 'restore/check',
    signOut: () => 'auth/logout',

    sendCompanyVerification: () => 'verify/token',
    verifyCompany: token => `verify/verify/${token}`
};

export const signIn = params => {
    return axios.post(endpoints.signIn(), { grantType: 'password', ...params });
};

export const refreshToken = params => {
    return axios.post(endpoints.signIn(), { grantType: 'refresh_token', ...params
});
};
```

```
export const oAuthSignIn = data => {
    return axios.post(endpoints.oAuthSignIn(), data);
};

export const linnworksSignIn = (linnworksToken, deviceId) => {
    return axios.post(endpoints.linnworksSignIn(linnworksToken, deviceId));
};

export const verifySignInCode = data => {
    return axios.post(endpoints.verifySignInCode(), data);
};

export const signUp = data => {
    return axios.post(endpoints.signUp(), data);
};

export const restorePassword = data => {
    return axios.post(endpoints.restorePassword(), data);
};
```

Щоб підвищити зручність інтерфейсу було додано більше варіантів налаштування інтерфейсу для користувача, в також декілька тем інтерфейсу (нічна та світла теми).

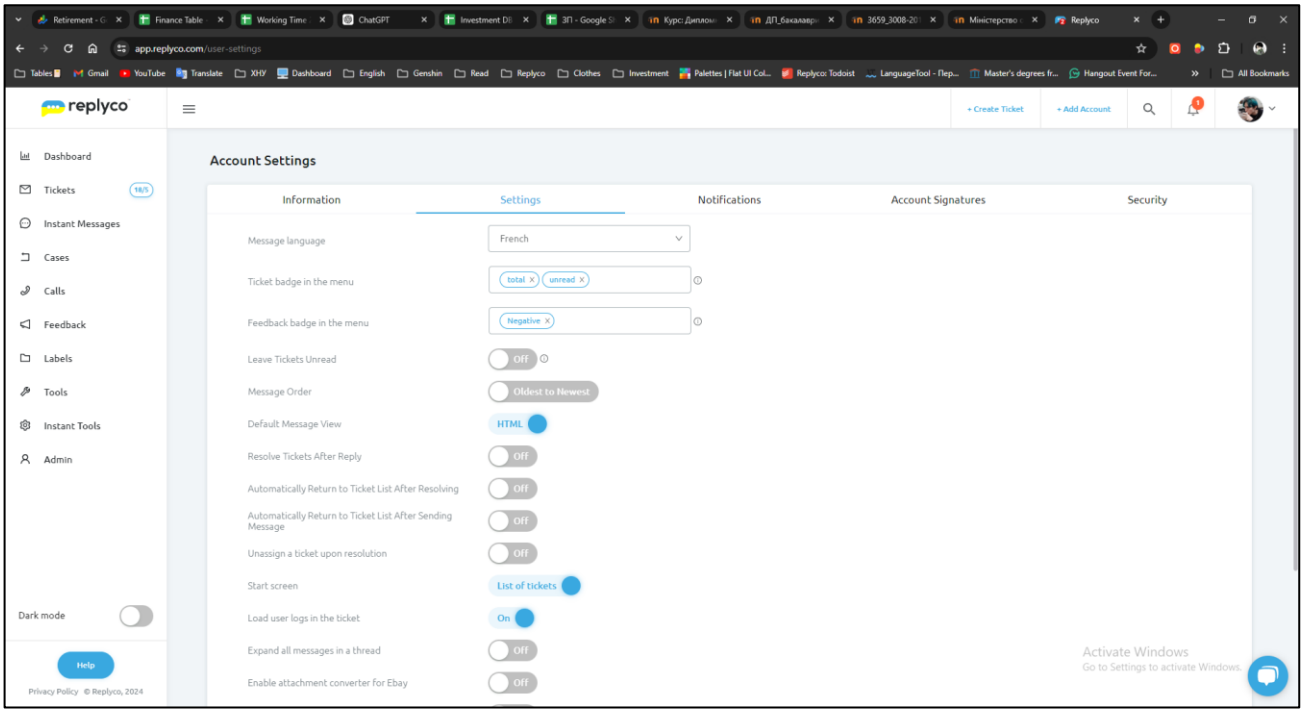


Рисунок 3.9 – Налаштування користувацького інтерфейсу

Код компоненти налаштування інтефейсу.

```
class Settings extends React.PureComponent {
    static propTypes = {
        i18n: PropTypes.object.isRequired,
```

```
        actions: PropTypes.object.isRequired
    }

    constructor(props) {
        super(props);
        this.props.actions.getCurrentUser();
    }


    get tabs() {
        const { i18n } = this.props;
        return [{
            tabProps: {
                key: TABS.information,
                tab: i18n.t('users.tabs.information')
            },
            component: this.renderInformation
        }, {
            tabProps: {
                key: TABS.notifications,
                tab: i18n.t('users.tabs.notifications')
            },
            component: this.renderNotifications
        }, {
            tabProps: {
                key: TABS.integrationsSignatures,
                tab: i18n.t('users.tabs.accountsSignatures')
            },
            component: this.renderIntegrationsSignatures
        }, {
            tabProps: {
                key: TABS.security,
                tab: i18n.t('settings.tabs.security')
            },
            component: this.renderSecurity
        }];
    }


    render() {
        const { i18n } = this.props;
        return (
            <Layout>
                <Title
                    title={i18n.t('shared.userSettings')}
                />

                <Tabs
                    defaultActiveKey={TABS.information}
                    tabs={this.tabs}
                    className={common.tabs}
                />
            </Layout>
        );
    }
}


const mapDispatchToProps = dispatch => ({
    actions: bindActionCreators({ getCurrentUser }, dispatch)
});
```

3.3 Тестування вебзастосунку

В процесі розробки програмного забезпечення необхідно, звичайно ж, проводити ретельне тестування проєкту, щоб запобігти виникненню неочікуваних помилок та забезпечити високу якість кінцевого продукту. Тестування програмного забезпечення є складним і важливим процесом, який включає перевірку відповідності заявлених до продукту вимог і реально реалізованої функціональності.

Це здійснюється шляхом спостереження за роботою програмного забезпечення в штучно створених ситуаціях та на обмеженому наборі тестів, які обираються певним чином для охоплення найважливіших сценаріїв використання.

Для тестування створеного API було використано Postman, який є одним з найпотужніших і найпопулярніших інструментів тестування API серед розробників. Postman допомагає створювати та тестувати неймовірні API, а також значно покращує продуктивність праці розробників. Цей додаток використовується більш ніж мільйоном розробників по всьому світу, і це число постійно зростає, що свідчить про його ефективність та популярність у спільноті розробників. Postman дозволяє зручно створювати та виконувати HTTP-запити, перевіряти відповіді сервера, автоматизувати тестування та документувати API, що робить його незамінним інструментом у процесі розробки.

Тестування API проводиться, ґрунтуючись на бізнес-логіці програмного продукту, що забезпечує відповідність функціональності реальним потребам користувачів. Тестування API відноситься до інтеграційного тестування, тобто воно допомагає виявити помилки взаємодії між різними модулями системи або між різними системами. Це особливо важливо для сучасних складних програмних рішень, де кілька компонентів повинні працювати разом бездоганно.

Під час тестування API використовуються спеціальні інструменти, за допомогою яких можна відправити вхідні дані в запиті та перевірити точність

вихідних даних. Одним з таких інструментів, якраз, і є Postman.

Postman дозволяє не тільки виконувати окремі запити, але й створювати колекції запитів для автоматизованого тестування, що значно спрощує процес тестування на різних етапах розробки. З його допомогою можна симулювати різні сценарії використання API, перевіряти обробку помилок і граничні значення, а також виконувати стрес-тестування для оцінки продуктивності API під навантаженням.

Використання Postman в процесі розробки допомагає забезпечити високу якість кінцевого продукту, зменшуючи кількість помилок і підвищуючи стабільність роботи системи. Це, в свою чергу, сприяє задоволенню користувачів та успіху програмного продукту на ринку.

Створений API обробляє наступні види запитів:

– get;

– post;

– put;

– delete.

Приклад тестування GET запиту на створений API зображено на рисунку 3.10.

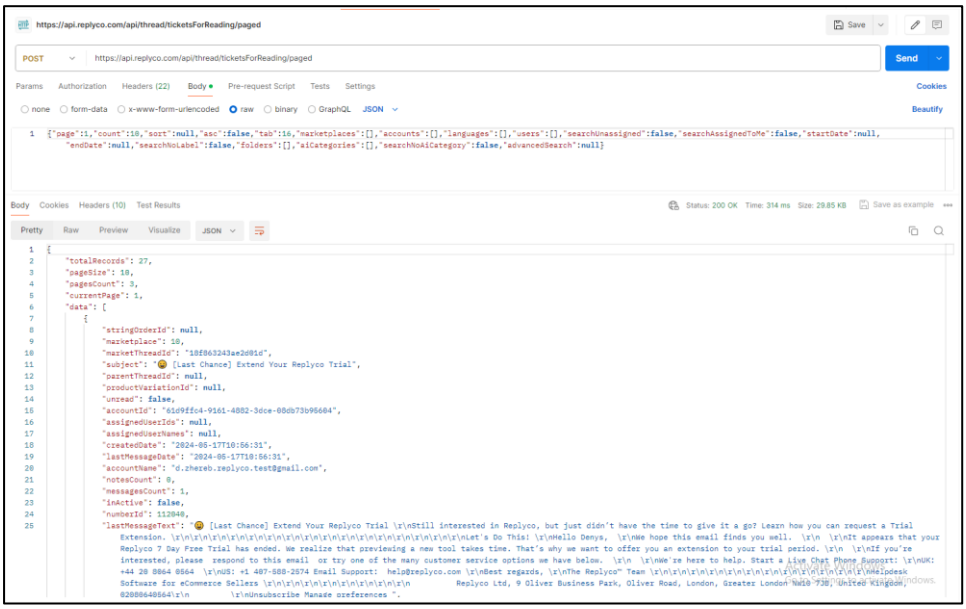


Рисунок 3.10 – Тестування GET запиту

Приклад тестування POST запиту на створений API зображено на рисунку 3.11.

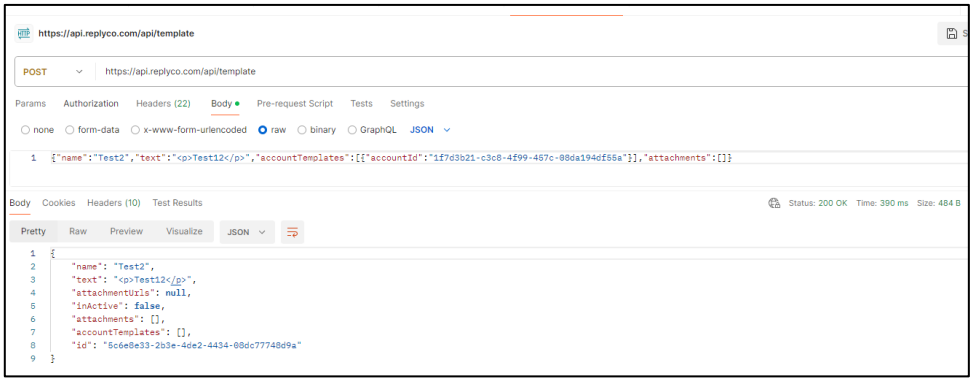


Рисунок 3.11 – Тестування POST запиту

Приклад тестування PUT запиту на створений API зображено на рисунку 3.12.

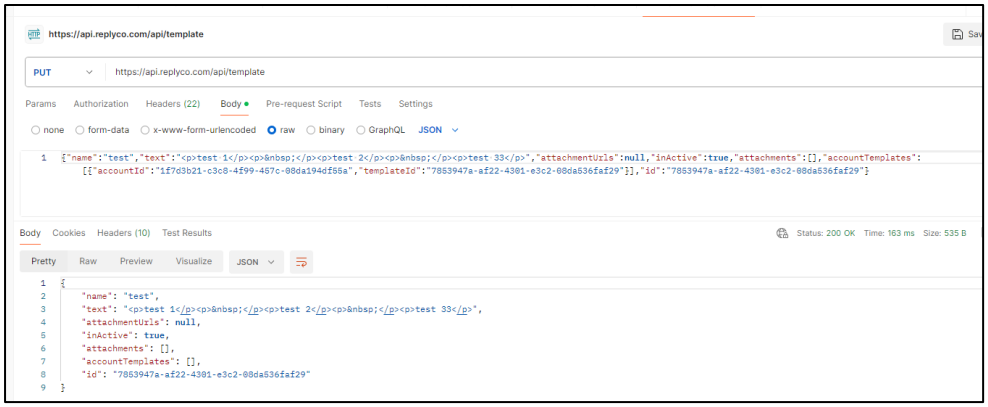


Рисунок 3.12 – Тестування PUT запиту

Приклад тестування DELETE запиту на створений API зображено на рисунку 3.13.

Рисунок 3.13 – Тестування PUT запиту

Як видно з рисунків всі запити було виконано успішно і помилок не було виявлено. Всі запити на сервер були протестовані і помилки виявлено. Всі виявлені помилки були виправленні, а в всі критичні місця сервера було додано оператори обробки виключень. Завдяки такому підходу сервер не буде завершувати свою роботу при типових помилках, які визначенні програмним кодом.

Окрім тестування API було також виконано ряд різних ручних тестів, що забезпечить додаткову якість розробленого ПЗ. Ручне тестування є важливим етапом у процесі забезпечення якості програмного забезпечення, оскільки воно дозволяє виявити помилки та проблеми, які можуть бути пропущені автоматизованими тестами. Ручне тестування включає в себе кілька видів тестування, кожен з яких має свою специфіку і цілі. Це дозволяє перевірити різні аспекти програмного забезпечення та забезпечити його відповідність вимогам та очікуванням користувачів.

Функціональне тестування є одним з ключових видів ручного тестування, яке було виконано для перевірки роботи всіх функцій та можливостей програмного забезпечення.

Це тестування спрямоване на перевірку того, що кожна функція програмного забезпечення працює відповідно до специфікацій. Під час тестування вручну перевірили різні сценарії використання, виконували операції з даними та перевіряли правильність їх обробки. Це включає тестування таких функцій, як створення та редагування записів, аутентифікація користувачів, обробка запитів та інші ключові можливості системи.

Тестування інтерфейсу було проведено для забезпечення того, що користувацький інтерфейс програми є зручним, інтуїтивно зрозумілим і функціональним. Під час цього тестування перевірено розташування елементів інтерфейсу, відповідність дизайну до вимог, коректність відображення даних та реакцію інтерфейсу на дії користувачів. Особлива увага приділялася перевірці доступності інтерфейсу для користувачів з різними потребами та на різних

пристроях. Це допомагає забезпечити, що всі користувачі, незалежно від своїх технічних навичок, можуть легко взаємодіяти з програмою.

Тестування сумісності включало перевірку роботи програмного забезпечення на різних платформах, браузерах та пристроях. Це тестування було необхідне для забезпечення того, що програма працює коректно незалежно від операційної системи, браузера чи типу пристрою, який використовують користувачі. Вручну перевірено функціональність та відображення інтерфейсу на різних комбінаціях платформ та браузерів, що дозволило виявити та виправити проблеми сумісності до випуску продукту на ринок. Тестування проходило в операційних системах Windows 8, Windows 10, Ubuntu Linux. Тестування проходило в браузерах Google Chrome, Firefox, Opera, Edge.

Тестування користувачем проводилося для перевірки програми з точки зору кінцевих користувачів. Під час цього тестування реальні користувачі виконували свої звичні завдання з використанням програмного забезпечення, надаючи зворотний зв'язок про його зручність, ефективність та можливі проблеми. Це тестування допомагає виявити проблеми, які могли бути не помічені під час функціонального та інтерфейсного тестування, та забезпечує впевненість у тому, що продукт відповідає потребам і очікуванням своїх користувачів.

Для перевірки основного функціоналу користувачам, які брали участь у тестуванні, було надано список основних завдань, які вони мали виконати. Основні пункти цього списку включали реєстрація користувача, перегляд тікету і його даних, написання відповіді на листа, назначення корситувача як відповідального за тікет, створення автовідповідача, вихід із профілю користувача.

Після виконання завдань кожному користувачеві було поставлено ряд запитань, пов'язаних з їхнім досвідом використання програмного продукту. Ці запитання були призначені для збору відгуків про їхній досвід взаємодії з програмним продуктом. Користувачі могли оцінити його зручність, ефективність

та інтуїтивність. Крім того, вони мали можливість поділитися своїми рекомендаціями та пропозиціями щодо можливих покращень програми.

Цей етап збору відгуків від користувачів був надзвичайно цінним, оскільки він дозволяє отримати реальні враження та пропозиції від людей, які використовують програмний продукт у реальних умовах. Ця інформація є надзвичайно важливою для подальшого вдосконалення програми, оскільки вона відображає реальні потреби та очікування користувачів.

Отримані відгуки можуть бути використані для покращення функціональності програми та підвищення її загальної якості.

Врахування пропозицій користувачів допомагає створити продукт, який краще відповідає їхнім потребам, що, у свою чергу, сприяє підвищенню задоволеності та лояльності користувачів.

За результатами тестування програмного продукту всі виявленні помилки були вирішено. Фронтенд та бекенд частини були ретельно протестовані. Всі рекомендації було запроваджено.

Програмний продукт готовий для експлуатації та має працездатний стан завдяки виконаним тестам.

3.4 Висновки

На цьому етапі можна вважати, що процеси програмної реалізації завершені.

На даному етапі завершено роботу над фронтенд модулем вебзастосунку з використанням технологій React та Redux. Всі стоірнки на форми були розроблені згідно спроектованого макету.

Бекенд частина застосунку була також детально опрацьована. Було створено необхідні моделі даних та налаштовано контекст бази даних для управління даними користувачів. Реалізовано основні API-методи для взаєможії бекенд та фронтед модулей. Для забезпечення безпеки використовуються сучасні

методи шифрування паролів та аутентифікації користувачів.

Вся логіка взаємодії між фронтендом і бекендом ретельно була протестована за допомогою Postman для забезпечення надійної роботи модуля та захисту даних користувачів.

Окрім тестування API було також виконано ряд різних ручних тестів, що забезпечить додаткову якість розробленого ПЗ.

Функціональне тестування було виконано для перевірки роботи всіх функцій та можливостей програмного забезпечення. Тестування інтерфейсу було проведено для забезпечення того, що користувацький інтерфейс програми є зручним, інтуїтивно зрозумілим і функціональним. Тестування сумісності включало перевірку роботи програмного забезпечення на різних платформах, браузерах та пристроях. Тестування користувачем проводилося для перевірки програми з точки зору кінцевих користувачів.

Усі етапи сприяли успішному впровадженню та розгортанню програмного продукту, забезпечивши його всебічну перевірку за різними параметрами. Враховуючи виконану роботу, можна сказати, що програмний продукт повністю реалізовано відповідно до всіх поствлених вимог.

ВИСНОВКИ

На основі проведеного дослідження предметної області, вивчення існуючих рішень, проектування програмного забезпечення та його реалізації, можна зробити висновок про успішне виконання кваліфікаційної роботи. На першому етапі було здійснено детальний аналіз різних поштових сервісів, що дозволило глибше зрозуміти їхню структуру та функціонал. Аналіз існуючих рішень дозволив виділити ключові характеристики, переваги та недоліки, що допомогло уникнути включення непотрібного функціоналу та покращити якість програмного продукту.

Наступним етапом було проектування програмного забезпечення. Обрана архітектура RESTful API, бібліотека React для користувацького інтерфейсу та фреймворк ASP.NET Core API для бекенду забезпечили високу ефективність та надійність системи. Використання Entity Framework та SQL Server дозволило забезпечити ефективну роботу з базами даних. Проектування структур даних і зв’язків між таблицями сприяло стійкості та цілісності даних. Розроблений мінімалістичний і зручний дизайн інтерфейсу покращив сприйняття інформації користувачами, забезпечуючи інтуїтивно зрозумілий користувацький досвід.

На завершальному етапі була реалізована фронтенд і бекенд частина застосунку, а також проведено детальне тестування. Всі сторінки та форми були розроблені згідно спроектованого макету, а API-методи забезпечили надійну взаємодію між фронтендом і бекендом. Тестування декількома методами високу якість розробленого програмного забезпечення. Усі етапи сприяли успішному впровадженню та розгортанню програмного продукту, забезпечивши його відповідність поставленим функціональним та нефункціональним вимогам.

Завдяки чіткому дотриманню визначених етапів, розроблений програмний продукт відповідає усім встановленим вимогам, є стійким до змін, легко масштабованим та зручним в обслуговуванні. Це забезпечує його довготривалу продуктивність та стабільність роботи, що підтверджує успішне виконання кваліфікаційної роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is email (electronic mail)? [Електронний ресурс] // Вікіпедія. – Режим доступу: https://www.techtarget.com/whatis/definition/e-mail-electronic-mail-or-email

2. Стаття електронної енциклопедії про email [Електронний ресурс] // Wiki. – Режим доступу: https://en.wikipedia.org/wiki/Email

3. Gunkies [Електронний ресурс] // gunkies. – Режим доступу: https://gunkies.org/wiki/Email

4. What is email? | Email definition [Електронний ресурс] // cloudflare – Режим доступу: https://mind.ua/publications/20222353-igrova-industriya-v-cifrah-skilki-ukrayinci-vitrachayut-na-videoigri

5. Email (electronic mail) [Електронний ресурс] // zoho. – Режим доступу: https://itc.ua/ua/novini/igrova-industriya-u-2022-roczi-obyem-200-milyardiv-dolariv-ponad-3-mlrd-gravcziv-i-istorichna-peremoga-ssha-nad-kitayem-prognoz-newzoo/

6. NINJAONE TICKETING The Helpdesk Solution for IT Teams [Електронний ресурс] // Веб-сайт NINJAONE TICKETING. – https://www.ninjaone.com/ticketing-software/

7. Airbnb [Електронний ресурс] // Вікіпедія. – Режим доступу: https://en.wikipedia.org/wiki/Airbnb

8. The Web & Communication [Електронний ресурс] // Веб-сайт SENET. – Режим доступу: https://senet.cloud/?utm_source=google& utm_medium=cpc& utm_campaign=searc h_cis_brand_tier2& utm_adgroup=SENET_Exact_Match& utm_term=SENET_Exact_Match&utm_content=senet&gclid=CjwKCAjwiOCgBhAgEiwAjv5whMdIxpycMfyqhsyfo94R8HADddwO-RGLqwzmxN2UQ5XfO7FqXEd-QxoCuNgQAvD_BwE#

9. E-mail, messages transmitted and received by digital computers through a network [Електронний ресурс] // Веб-сайт britannica. – Режим доступу: https:// britannica.ru/sistema-upravleniya-kompyuternym-klubom-locker-programma-dlya/

10. React Документаці [Електронний ресурс] // Офіційна документація React.

– Режим доступу: https://react.dev/

11. Redux Документаці [Електронний ресурс] // Офіційна документація Redux. – Режим доступу: https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow

12. React Router Документаці [Електронний ресурс] // Офіційна документація React Router. – Режим доступу: https://reactrouter.com/en/main

13. Підручник: створення веб-API за допомогою ASP.NET Core [Електронний ресурс] // Microsoft документація. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-8.0&tabs=visual-studio

14. Entity Framework Core [Електронний ресурс] // Microsoft документація. – Режим доступу: https://learn.microsoft.com/en-us/ef/core/

15. SQL Server [Електронний ресурс] // Microsoft документація. – Режим доступу: https://www.microsoft.com/en-us/sql-server/

16. C# Tutorial [Електронний ресурс] // Сайт w3schools. – Режим доступу: https://www.w3schools.com/cs/index.php

17. What is a REST API? [Електронний ресурс] // Сайт IBM. – Режим доступу: [https://www.ibm.com/topics/rest-apis#:~:text=A%20REST%20API%20(also%20called,transfer%20(REST)%20architectural%20style](https://www.ibm.com/topics/rest-apis#:~:text=A%20REST%20API%20(also%20called,transfer%20(REST)%20architectural%20style.)

18. Pehlivanov V. ASP.NET Core vs ASP.NET MVC: Which .NET Framework is better for You? [Електронний ресурс] // Сайт resolutesoftware. – Режим доступу: https://www.resolutesoftware.com/blog/asp-net-mvc-vs-asp-net-core/

19. Основні принципи створення інтерфейсу [Електронний ресурс] // Сайт um.co. – Режим доступу: http://um.co.ua/8/8-10/8-109690.html#google_vignette

20. Огляд видів тестування. [Електронний ресурс] // Сайт qatestlab. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/

ДОДАТОК А

(обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

A.1 – Розмітка сторінки SignUp

```
import React from 'react';
import PropTypes from 'prop-types';
import queryString from "query-string";
import { Prompt, withRouter } from 'react-router-dom';
import { connect } from 'react-redux';
import { bindActionCreators } from "redux";
import _ from "lodash";
import { selectors as authSelectors, actions as authActions } from
"../../base/auth";
import { ROUTES, translate, URLS, validator, actions as coreActions,
LocationService, AppSettingsService } from '../../base/core';
import * as Core from "../core";
import { SocialSignIn, AuthTemplate, AuthBlock } from "./common";
import { colors, css, fonts, spacing, StyleSheet } from "../../styles";

const { signUp } = authActions;
const { changeLanguage } = coreActions;

const FIELDS = {
  email: 'email',
  password: 'password',
  fullName: 'fullName',
  companyName: 'companyName',
  workPhone: 'workPhone',
  country: 'country'
};

class SignUp extends React.PureComponent {
  static propTypes = {
    i18n: PropTypes.object.isRequired,
    location: PropTypes.object.isRequired,
    actions: PropTypes.object.isRequired,
    emailFromUrl: PropTypes.string,
    campaign_id: PropTypes.string,
    source: PropTypes.string,
    isRegistering: PropTypes.bool
  };

  static defaultProps = {
    emailFromUrl: undefined,
    campaign_id: undefined,
    source: undefined,
    isRegistering: false
  };

  constructor(props) {
    super(props);
    this.useShopifyBilling = !!localStorage.getItem('shopifyPreinstall');
    const { country, appLanguage, region, phonePrefix } =
AppSettingsService.getDefaultAppOnAuth();
    this.state = {
      email: props?.emailFromUrl || '',
      password: '',
      fullName: '',
      companyName: '',
      workPhone: '',
      country,
      appLanguage,
      phonePrefix,
      region,
      errors: {},
      isVisibleModalAfterSocialSinIn: false
    };
    this.onChangeAppLanguage(appLanguage);
  }

  componentDidUpdate(prevProps, prevState) {
    if (!this.hasEnteredData(prevState) && this.hasEnteredData(this.state)) {
      window.addEventListener("beforeunload", this.onLeavePage);
    }
  }

  componentWillUnmount() {
    window.removeEventListener("beforeunload", this.onLeavePage);
  }

  get countryList() {
    return _.map(LocationService.COUNTRIES, country => ({
      id: country.name,
      name: country.name,
      code: country.code
    }));
  }

  get passwordInput() {
    const { i18n } = this.props;
    const { password, errors } = this.state;
    return (
      <Core.PasswordInputSecondary
        placeholder={i18n.t('auth.password')}
        value={password}
        name={FIELDS.password}
        onChange={this.onChangeField}
        id="signup_password"
        errorMessage={errors[FIELDS.password]}
        onHideError={this.onHideError}
      />
    );
  }

  get confirmSignUpButton() {
    const { i18n, isRegistering } = this.props;
    return (
      <>
        {/* <p className={css(styles.text)}>
          {i18n.t('auth.termsAndPrivacy.part1')}
          <a
            target="_blank"
            href={URLS.TERMS_AND_CONDITIONS}
            rel="noopener noreferrer"
            className={css(styles.link)}>
            {i18n.t('auth.termsAndPrivacy.part2')}
          </a>
          {i18n.t('auth.termsAndPrivacy.part3')}
          <a
            target="_blank"
            href={URLS.TERMS_AND_CONDITIONS}
            rel="noopener noreferrer"
            className={css(styles.link)}>
            {i18n.t('auth.termsAndPrivacy.part4')}
          </a>
        </p> */}
        <Core.ButtonSecondary
          label={i18n.t('confirm')}
          onPress={this.onSignUp}
          isLoading={isRegistering}
          isDisabled={isRegistering}
          className={styles.button}
        />
      </>
    );
  }

  hasEnteredData = state => {
    const { email, password, fullName, companyName, workPhone } = state;
    return email?.length || password?.length || fullName?.length ||
companyName?.length || workPhone?.length;
  };

  getCountryCode = country => _.get(_.find(this.countryList, item =>
item.name === country), 'code');

  onChangeField = (value, field) => this.setState({ [field]: value });

  onChangeCountry = (value, field) => {
    this.setState({
      [field]: value,
      region: AppSettingsService.getAppRegionByCountry(value),
      phonePrefix: _.toLower(this.getCountryCode(value))
    });

this.onChangeAppLanguage(AppSettingsService.getLanguageByCountry(value
));
  };

  onChangeAppLanguage = appLanguage => {
    this.setState({ appLanguage });

this.props.actions.changeLanguage(AppSettingsService.getAppLanguageLocal(
appLanguage));
  };

  onHideError = field => {
    this.setState(prevState => ({
      errors: _.omit(prevState.errors, [field])
    }));
  };

  onSocialLogin = socialUser => {
```

```
        const fullName = `${socialUser.profile.firstName}
${socialUser.profile.lastName}`;
        this.setState({
            fullName,
            email: socialUser.profile.email,
            companyName: fullName,
            isVisibleModalAfterSocialSinIn: true
        });
    }

    onSignUp = () => {
        const { i18n } = this.props;
        const { email, password, fullName, companyName, workPhone, country,
appLanguage, region, isVisibleModalAfterSocialSinIn } = this.state;
        const useShopifyBilling = this.useShopifyBilling;
        //region registration data
        const errors = {};
        if (!email || !validator.validateEmail(email)) {
            errors[FIELDS.email] = i18n.t('auth.enterCorrectEmail');
        }
        if (!password) {
            errors[FIELDS.password] = i18n.t('auth.enterCorrectPassword');
        }
        if (!fullName) {
            errors[FIELDS.fullName] = i18n.t('auth.enterYourName');
        }
        const trimmedPhone = !isVisibleModalAfterSocialSinIn ?
workPhone.replace(/\s+/g, '') : null;
        if (!isVisibleModalAfterSocialSinIn && (!trimmedPhone ||
trimmedPhone.length < 10)) {
            errors[FIELDS.workPhone] = i18n.t('auth.enterYourPhone');
        }
        if (!companyName) {
            errors[FIELDS.companyName] =
i18n.t('auth.enterYourCompanyName');
        }

        if (!_.isEmpty(errors)) {
            this.setState({ errors });
            return;
        }

        const names = fullName.split(' ');
        const firstName = names?.[0] || '';
        const lastName = names?.[1] || '';
        const data = {
            email,
            password,
            useShopifyBilling,
            workPhone: trimmedPhone,
            companyName,
            firstName,
            lastName,
            region,
            country,
            appLanguage
        };
        //endregion

        //region hubspot data
        // eslint-disable-next-line camelcase
        const { campaign_id, source } = this.props;
        const allCookies = document.cookie.split('; ');
        let trackingCode = allCookies.find(x => x.startsWith('tracking_code'));
        if (trackingCode) {
            trackingCode = trackingCode.replace('tracking_code=', '');
        }
        let lmref = allCookies.find(x => x.startsWith('lmref_code'));
        if (lmref) {
            lmref = lmref.replace('lmref_code=', '').replace('?', '').replace('lmref=', '');
        }
        const params = queryString.parse(trackingCode ? trackingCode :
this.props.location.search);
        const customerData = {
            campaign_id,
            source,
            email,
            phone: trimmedPhone,
            companyName,
            firstName,
            lastName,
            ...params,
            referral_partner: lmref || 'no_ref'
        };
        //endregion

        this.props.actions.signUp(data, customerData);
        this.onCloseModalAfterSocialSignIn();
    };

    onLeavePage = e => {
        e.returnValue = this.props.i18n.t('accounts.confirmLeave');
    };

    onCloseModalAfterSocialSignIn = () => this.setState({
isVisibleModalAfterSocialSinIn: false });

    render() {
        const { i18n, isRegistering } = this.props;
        const { email, fullName, companyName, workPhone, errors, country,
phonePrefix, isVisibleModalAfterSocialSinIn } = this.state;
        return (
            <Core.SpinnerWrapper isLoading={isRegistering}
tip={i18n.t('auth.registeringYourWorkspace')}>
                <AuthTemplate hasChangedOrder={true} hasCloseButton={true}>
                    <AuthBlock
                        title={i18n.t('auth.signUp')}
                        subtitle={i18n.t('auth.signUpDescription')}
                        footer={(
                            <>
                                {`${i18n.t('auth.alreadyHaveAnAccount')} `}
                                <Core.Link
                                    text={i18n.t('auth.signInHere')}
                                    route={ROUTES.home()}
                                />
                            </>
                        )}>
                        <Core.SelectSecondary
                            name={FIELDS.country}
                            data={this.countryList}
                            value={country}
                            placeholder={i18n.t('settings.company.country')}
                            onChange={this.onChangeCountry}
                            hasRoundBorder={true}
                            hasMargins={true}
                        />
                        <Core.InputSecondary
                            placeholder={i18n.t('auth.fullName')}
                            value={fullName}
                            name={FIELDS.fullName}
                            onChange={this.onChangeField}
                            autoFocus={true}
                            id="signup_fullname"
                            errorMessage={errors[FIELDS.fullName]}
                            onHideError={this.onHideError}
                        />
                        <Core.InputSecondary
                            placeholder={i18n.t('auth.companyName')}
                            value={companyName}
                            name={FIELDS.companyName}
                            onChange={this.onChangeField}
                            id="signup_company"
                            errorMessage={errors[FIELDS.companyName]}
                            onHideError={this.onHideError}
                        />
                        <Core.InputSecondary
                            placeholder={i18n.t('auth.email')}
                            value={email}
                            name={FIELDS.email}
                            onChange={this.onChangeField}
                            id="signup_email"
                            errorMessage={errors[FIELDS.email]}
                            onHideError={this.onHideError}
                        />
                        {this.passwordInput}
                        <Core.InputPhoneNumberSecondary
                            placeholder={i18n.t('auth.phone')}
                            value={workPhone}
                            name={FIELDS.workPhone}
                            onChange={this.onChangeField}
                            id="signup_phone"
                            errorMessage={errors[FIELDS.workPhone]}
                            onHideError={this.onHideError}
                            country={phonePrefix}
                        />
                        {this.confirmSignUpButton}
                        {/* <SocialSignIn
                            onSuccess={this.onSocialLogin}
                            separatorText={i18n.t('auth.orSignUpWith')}
                        /> */}
                        <Core.Modal
                            isVisible={isVisibleModalAfterSocialSinIn}
                            footer={null}
                            onClose={this.onCloseModalAfterSocialSignIn}>
                            <div className={css(styles.modal)}>
```

```
                        <p
className={css(styles.modalHeader)}>{i18n.t('auth.enterPasswordAfterSocial
SingIn')}</p>
                        {this.passwordInput}
                        {this.confirmSignUpButton}
                      </div>
                    </Core.Modal>
                  </AuthBlock>
                  {this.hasEnteredData(this.state) ?
                    <Prompt
                      message={() => i18n.t('accounts.confirmLeave')}
                    /> : null
                  }
                </AuthTemplate>
              </Core.SpinnerWrapper>
            );
          }
}

const mapStateToProps = (state, ownProps) => {
  const params = queryString.parse(ownProps.location.search);
  return {
    campaign_id: _.get(ownProps.match.params, 'utm_campaign', ''),
    source: _.get(ownProps.match.params, 'utm_source', ''),
    emailFromUrl: params?.email,
    isRegistering: authSelectors.isRegistering(state)
  };
};

const mapDispatchToProps = dispatch => ({
  actions: bindActionCreators({ signUp, changeLanguage }, dispatch)
});
});
```

```
export default withRouter(connect(mapStateToProps,
mapDispatchToProps)(translate()(SignUp)));

const styles = StyleSheet.create({
  text: {
    ...fonts.smRegular,
    textAlign: 'center',
    color: colors.authText2,
    marginTop: spacing.s5,
    marginBottom: spacing.s8,
    '@media(max-width: 500px)': {
      marginTop: spacing.s2,
      marginBottom: spacing.s3
    }
  },
  link: {
    color: colors.primary
  },
  button: {
    margin: `0 auto ${spacing.s5}px`
  },
  modal: {
    textAlign: 'center',
    padding: `${spacing.s8}px ${spacing.s4}px ${spacing.s2}px
${spacing.s4}px`
  },
  modalHeader: {
    ...fonts.lgMedium,
    color: colors.primary,
    paddingBottom: spacing.s3
  }
```

A.2 – Розмітка сторінки SignIn

```
import React from 'react';
import { withRouter } from 'react-router-dom';
import { connect } from 'react-redux';
import { bindActionCreators } from "redux";
import PropTypes from 'prop-types';
import _ from 'lodash';
import queryString from "query-string";
import { selectors as authSelectors, actions as authActions } from
"../../base/auth";
import { actions as coreActions, translate, ROUTES, TICKET_TYPES,
destinationPath, TableStorageService, validator, Toast, TOAST_TYPES,
AppSettingsService } from '../../base/core';
import { Link, ButtonSecondary, SpinnerWrapper, InputSecondary,
PasswordInputSecondary } from '../core';
import { SocialSignIn, AuthTemplate, AuthBlock } from './common';
import SignInFromInvite from './SignInFromInvite';
import { StyleSheet, css, spacing, fonts, colors } from '../../styles';

const { changeLanguage } = coreActions;
const { signIn, oAuthSignIn, linnworksSignIn, verifySignInCode } =
authActions;

const FIELDS = {
  email: 'email',
  password: 'password',
  code: 'code'
};

class SignIn extends React.PureComponent {
  static propTypes = {
    i18n: PropTypes.object.isRequired,
    actions: PropTypes.object.isRequired,
    history: PropTypes.object.isRequired,
    location: PropTypes.object.isRequired,
    isLoggedIn: PropTypes.bool,
    isAuthorizing: PropTypes.bool,
    token: PropTypes.string,
    authenticatorToken: PropTypes.string,
    isVerifyingSignInCode: PropTypes.bool
  };

  static defaultProps = {
    isLoggedIn: false,
    isAuthorizing: false,
    token: undefined,
    authenticatorToken: null,
    isVerifyingSignInCode: false
  };

  constructor(props) {
    super(props);
    this.state = {
```

```
      email: '',
      password: '',
      code: '',
      redirectingToShopify: false,
      errors: {}
    };
    const parsed = queryString.parse(props.location.search);
    const shop = _.get(parsed, 'shop');
    if (_.get(parsed, 'hmac') && shop && _.get(parsed, 'timestamp') &&
!_.get(parsed, 'code') && !_.get(parsed, 'session')) {
      this.state.redirectingToShopify = true;
      localStorage.setItem("shopifyPreinstall", props.location.search);

window.location.replace(`https://${shop}/admin/oauth/authorize?client_id=8fa
1a2606b00063cda5a1fb123e45f3d&scope=read_customers,read_orders,read_p
roducts,read_draft_orders,read_product_listings&redirect_uri=https://app.reply
co.com/integrations/shopify/success`);
    }
    /*const linnworksToken = _.get(parsed, 'token');
    if (linnworksToken) {
      this.checkLinnworksToken(linnworksToken);
    }*/ // disabled auto login temporarily as per support team request
    this.setAppLanguage();
  }

  componentDidMount() {
    if (this.props.isLoggedIn) {
      this.props.history.push(destinationPath.getDestinationsPath() ||
ROUTES.home());
    }
  }

  componentDidUpdate(prevProps) {
    if (!prevProps.isLoggedIn && this.props.isLoggedIn) {
      this.props.history.push(destinationPath.getDestinationsPath() ||
ROUTES.home());
    }
    if (prevProps.isVerifyingSignInCode &&
!this.props.isVerifyingSignInCode && !this.props.authenticatorToken &&
!this.props.isLoggedIn) {
      this.setState({ password: '' });
    }
  }

  setAppLanguage = () => {
    try {
      const { appLanguage } = AppSettingsService.getDefaultAppOnAuth();

this.props.actions.changeLanguage(AppSettingsService.getAppLanguageLocal(
appLanguage));
    }
    catch {
```

```
            // just ignore
        }
    };

    checkLinnworksToken = linnworksToken =>
this.props.actions.linnworksSignIn(linnworksToken);

    onChangeField = (value, field) => this.setState({ [field]: value });

    onHideError = field => {
        this.setState(prevState => ({
            errors: _.omit(prevState.errors, [field])
        }));
    };

    onSignIn = () => {
        const { i18n } = this.props;
        const { email, password } = this.state;
        const errors = {};
        if (!email || !validator.validateEmail(email)) {
            errors[FIELDS.email] = i18n.t('auth.enterCorrectEmail');
        }
        if (!password) {
            errors[FIELDS.password] = i18n.t('auth.enterCorrectPassword');
        }
        if (!_.isEmpty(errors)) {
            this.setState({ errors });
        } else {
            this.props.actions.signIn({ email, password }, true);
            _.map(TICKET_TYPES, ticket => {
                TableStorageService.removeFilters(ticket);
                TableStorageService.removeSorting(ticket);
                TableStorageService.removePagination(ticket);
            });
        }
    };

    onSocialLogin = user => {
        this.props.actions.oAuthSignIn({ ...user, email: user.profile.email }, () =>
{
            this.props.history.push(ROUTES.signUp());
        });
    };

    onVerifyCode = () => {
        const { authenticatorToken } = this.props;
        const { code } = this.state;
        if (code?.length !== 6)
        {
            Toast.error({ type: TOAST_TYPES.invalidVerificationCode });
            return;
        }
        this.props.actions.verifySignInCode({ token: authenticatorToken, code });
    };

    render() {
        const { i18n, isAuthorizing, token, authenticatorToken,
isVerifyingSignInCode } = this.props;
        const { errors, email, password, code } = this.state;
        return (
            <SpinnerWrapper isLoading={this.state.redirectingToShopify}
tip={i18n.t('auth.authorizingWithShopify')}>
                <AuthTemplate hasCloseButton={!token}>
                    {token ?
                        <SignInFromInvite
                            token={token}
                        /> :
                        <AuthBlock
                            title={i18n.t('auth.logInToReplyco')}
                            subtitle={i18n.t('auth.signInDescription')}
                            footer={
                                <>
                                    {`${i18n.t('auth.doNotHaveAccountYet')} `}
                                    <Link
                                        text={i18n.t('auth.signUpHere')}
                                        route={ROUTES.signUp()}
                                    />
                                </>
                            }>
                            {!authenticatorToken?.length ?
                                <>
                                    <InputSecondary
                                        placeholder={i18n.t('auth.email')}
                                        value={email}
                                        name={FIELDS.email}
                                        onChange={this.onChangeField}
                                        autoFocus={true}
                                        errorMessage={errors[FIELDS.email]}
                                        onHideError={this.onHideError}
                                    />
                                    <PasswordInputSecondary
                                        placeholder={i18n.t('auth.password')}
                                        value={password}
                                        name={FIELDS.password}
                                        onChange={this.onChangeField}
                                        onPressEnter={this.onSignIn}
                                        errorMessage={errors[FIELDS.password]}
                                        onHideError={this.onHideError}
                                    />
                                    <p className={css(styles.text)}>
                                        {i18n.t('auth.forgotYourPassword.part1')}
                                        <Link
                                            text={i18n.t('auth.forgotYourPassword.part2')}
                                            route={ROUTES.forgotPassword()}
                                        />
                                        {i18n.t('auth.forgotYourPassword.part3')}
                                    </p>
                                    <ButtonSecondary
                                        label={i18n.t('confirm')}
                                        onPress={this.onSignIn}
                                        isLoading={isAuthorizing}
                                        isDisabled={isAuthorizing}
                                        className={styles.button}
                                    />
                                    {/* <SocialSignIn
                                        onSuccess={this.onSocialLogin}
                                        separatorText={i18n.t('auth.orSignInWith')}
                                    /> */}
                                </> :
                                <>
                                    <InputSecondary
                                        placeholder={i18n.t('auth.verificationCode')}
                                        value={code}
                                        name={FIELDS.code}
                                        onChange={this.onChangeField}
                                        autoFocus={true}
                                        errorMessage={errors[FIELDS.code]}
                                        onHideError={this.onHideError}
                                        onPressEnter={this.onVerifyCode}
                                    />
                                    <ButtonSecondary
                                        label={i18n.t('confirm')}
                                        onPress={this.onVerifyCode}
                                        isLoading={isVerifyingSignInCode}
                                        isDisabled={isVerifyingSignInCode}
                                        className={styles.button}
                                    />
                                </>
                            }
                        </AuthBlock>
                    }
                </AuthTemplate>
            </SpinnerWrapper>
        );
    }
}

const mapStateToProps = (state, ownProps) => ({
    isLoggedIn: authSelectors.isLoggedIn(state),
    isAuthorizing: authSelectors.isAuthorizing(state),
    token: _.get(ownProps.match, 'params.t'),
    authenticatorToken: authSelectors.getSignInAuthenticatorToken(state),
    isVerifyingSignInCode: authSelectors.isVerifyingSignInCode(state)
});

const mapDispatchToProps = dispatch => ({
    actions: bindActionCreators({ changeLanguage, signIn, oAuthSignIn,
linnworksSignIn, verifySignInCode }, dispatch)
});

export default withRouter(connect(mapStateToProps,
mapDispatchToProps)(translate()(SignIn)));

const styles = StyleSheet.create({
    text: {
        ...fonts.smRegular,
        textAlign: 'center',
        color: colors.authText2,
        marginTop: spacing.s5,
        marginBottom: spacing.s8,
        '@media(max-width: 500px)': {
            marginTop: spacing.s2,
            marginBottom: spacing.s3
        }
    },
    button: {
```

```
    margin: `0 auto ${spacing.s5}px`
  }
```

A.3 – Код контролера AuthController

```
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Microsoft.IdentityModel.Tokens;
using Newtonsoft.Json;
using Replyco.API.Helpers.Attributes;
using Replyco.API.Helpers.Services.Implementations;
using Replyco.Core.Entities.Mongo;
using Replyco.Core.Enums;
using Replyco.Core.Helpers;
using Replyco.Core.Helpers.Configurations;
using Replyco.Domain.Core.Models.DTO.Auth;
using Replyco.Domain.Managers.Interfaces.Auth;
using Replyco.Domain.Providers;
using Replyco.Domain.Services;
using Replyco.Domain.Services.AuthLinnworks;
using Replyco.Domain.Services.SignUp;
using Replyco.Models.Domain.Auth;
using Replyco.Models.DTO;
using Replyco.Models.DTO.Auth;
using Replyco.Models.DTO.Integrations;
using Replyco.MongoDatabase.Services;

namespace Replyco.API.Controllers.Auth;

[Route("api/[controller]")]
[ApiExplorerSettings(GroupName = "v1")]
public class AuthController : Controller
{
    private readonly IAuthManager _authManager;
    private readonly GoogleAuthProvider _googleAuthProvider;
    private readonly FacebookAuthProvider _facebookAuthProvider;
    private readonly IUsersActivityManager _usersActivityManager;
    private readonly IUserManager _userManager;
    private readonly UserProvider _userProvider;
    private readonly AuthLinnworksService _authLinnworksService;
    private readonly IVerifyCompanyTicketManager
_verifyCompanyTicketManager;
    private readonly IUserAuthenticatorTicketManager
_userAuthenticatorTicketManager;
    private readonly ILogger<AuthController> _logger;
    private readonly SignUpService _signUpService;

    public AuthController(IAuthManager authManager,
        GoogleAuthProvider googleAuthProvider,
        FacebookAuthProvider facebookAuthProvider,
        IUsersActivityManager usersActivityManager,
        IUserManager userManager,
        UserProvider userProvider,
        AuthLinnworksService authLinnworksService,
        IVerifyCompanyTicketManager verifyCompanyTicketManager,
        IUserAuthenticatorTicketManager userAuthenticatorTicketManager,
        ILogger<AuthController> logger,
        SignUpService signUpService)
    {
        _authManager = authManager;
        _googleAuthProvider = googleAuthProvider;
        _facebookAuthProvider = facebookAuthProvider;
        _usersActivityManager = usersActivityManager;
        _userManager = userManager;
        _userProvider = userProvider;
        _authLinnworksService = authLinnworksService;
        _verifyCompanyTicketManager = verifyCompanyTicketManager;
        _userAuthenticatorTicketManager = userAuthenticatorTicketManager;
        _logger = logger;
        _signUpService = signUpService;
    }

    [HttpPost("registration")]
    public async Task<IActionResult> Registration([FromBody]
CompanyRegistrationModel registrationModel)
    {
        var result = await _authManager.RegisterCompany(registrationModel);
        if (result.Success)
        {
            var companyId = result.User.CompanyId;
            await _signUpService.CreateDbUser(companyId,
registrationModel.Region, !result.IsUsedPreparedDb);

            try
            {
                await
_verifyCompanyTicketManager.CreateVerifyTicketAsync(result.User);
            }
            catch (Exception e) {
                _logger.LogError($"Error when creating verify email: {e.Message} --
------ {result.User.CompanyId}");
            }

            var res = GetResult(result.User.Id, result.Identity.Claims);
            Response.Cookies.Append(AppConstants.EmailCookieName,
$"{result.User.Email}");
            var principal = new ClaimsPrincipal(result.Identity);
            await
HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationSchem
e, principal);
            res.RefreshToken = result.User.RefreshToken.Find(x => x.DeviceId ==
registrationModel.DeviceId)?.RefreshToken;
            return Json(res);
        }
        return BadRequest();
    }

    [HttpPost("logout")]
    [Authorize]
    [ApiExplorerSettings(GroupName = "integration")]
    [Token]
    public async Task Logout([FromBody] string deviceId, string token)
    {
        await
_userManager.RemoveUserNotificationTokensByDeviceId(_userProvider.Get
CurrentUserId(), deviceId);
        Response.Cookies.Delete(AppConstants.EmailCookieName);
        var userId = _userProvider.GetCurrentUserId();
        await _usersActivityManager.CreateAsync(new UsersActivityModel {
Action = "Log out", UserId = userId });
        if (token != null) AuthTokensService.RemoveByToken(userId, token);
        await HttpContext.SignOutAsync();
    }

    [HttpPost("token")]
    [ApiExplorerSettings(GroupName = "integration")]
    public async Task<IActionResult> Token([FromBody] TokenSignInModel
model)
    {
        if (model == null)
        {
            return BadRequest("Please fill login form.");
        }
        AuthModel authModel;
        switch (model.GrantType)
        {
            case "password":
                {
                    authModel = await _authManager.SignIn(model);

                    break;
                }
            case "refresh_token":
                {
                    authModel = await _authManager.RefreshToken(model);
                    break;
                }
            default:
                {
                    return BadRequest("Cannot find this type of login.");
                }
        }
        if (authModel.Success)
        {
            var user = await _userManager.GetAsync(authModel.User.Id);

            if (!string.IsNullOrEmpty(model.ip))
            {
                var location = await IPLocationService.GetIpLocation(model.ip);
                if (location != null && (location.country_code2 == "RU" ||
location.country_code2 == "BY"))
});
```

```
            {
                try
                {
                    var slackService = new
SlackService(ConfigurationsManager.GetSlackBlacklistChannelUri());
                    slackService.PostMessage(
                        $"CompanyId - {user.CompanyId}{Environment.NewLine}"
+
                        $"UserId - {user.Id}{Environment.NewLine}" +
                        $"IP - {model.ip}{Environment.NewLine}" +
                        $"Data - {JsonConvert.SerializeObject(location)}"
                    );
                }
                catch
                {
                    _logger.LogError("Error during sending slack blacklist
message");
                }
            }
        }

        if (model.GrantType == "password")
        {
            var response2FA = await VerifyUserOn2FA(user);
            if (response2FA != null)
                return Json(response2FA);
        }

        return await ProcessUserSignIn(user.Id, authModel, model.DeviceId,
model.ip);
    }
    return BadRequest(authModel.Message);
  }

  [HttpPost("byLinnToken/{linnworksToken}/{deviceId}")]
  [ApiExplorerSettings(GroupName = "integration")]
  public async Task<IActionResult> ByLinnworksToken(string
linnworksToken, string deviceId)
  {
    if (Guid.Parse(linnworksToken) == Guid.Empty)
    {
        return BadRequest("Bad linnworks Guid id.");
    }

    var authModel = await CheckLinnworksTempDB(linnworksToken,
deviceId);
    if (authModel.Success)
    {
        var response2FA = await VerifyUserOn2FA(authModel.User);
        if (response2FA != null)
            return Json(response2FA);
        return await ProcessUserSignIn(authModel.User.Id, authModel,
deviceId);
    }

    var authLinnworksData = await
_authLinnworksService.GetLinnworksAuthData(linnworksToken);
    if (authLinnworksData != null &&
!string.IsNullOrEmpty(authLinnworksData.Email))
    {
        var user = await
_userManager.FindByEmail(authLinnworksData.Email);
        if (user != null)
        {
            authModel = await _authManager.SignIn(user, deviceId);
            if (authModel.Success)
            {
                var response2FA = await VerifyUserOn2FA(authModel.User);
                if (response2FA != null)
                    return Json(response2FA);
                return await ProcessUserSignIn(user.Id, authModel, deviceId);
            }
        }
    }

    return BadRequest("Something went wrong!");
  }

  [HttpPost("oauth")]
  public async Task<IActionResult> OAuth([FromBody] OAuthSignInModel
model)
  {
    if (model == null)
    {
        return Unauthorized();
    }

    AuthModel authModel;
    switch (model._Provider)
    {
        case "facebook":
            {
                authModel = await _facebookAuthProvider.Authorize(model);
                break;
            }
        case "google":
            {
                authModel = await _googleAuthProvider.Authorize(model);
                break;
            }
        default:
            {
                return Unauthorized();
            }
    }

    if (authModel.Success)
    {
        var user = await _userManager.GetAsync(authModel.User.Id);

        var response2FA = await VerifyUserOn2FA(user);
        if (response2FA != null)
            return Json(response2FA);

        return await ProcessUserSignIn(user.Id, authModel, model.DeviceId);
    }

    return Unauthorized();
  }
}
```

A.4 – Розмітка сторінки ThreadTable

```
import React from 'react';
import { connect } from 'react-redux';
import _ from 'lodash';
import { withRouter } from 'react-router-dom';
import PropTypes from 'prop-types';
import moment from 'moment';
import { TableResizable, Tooltip, ProgressLine, Tag, Popover, Divider } from
'../../core';
import { colors, css, fonts, spacing, StyleSheet } from '../../../styles';
import { ROUTES, translate, toLocalTime, htmlParser, TableStorageService,
getObjectValue } from '../../../base/core';
import { valueBuilder, THREAD_TABLE_COLUMNS,
  THREAD_TABLE_COLUMNS_ENUM,
THREAD_DEFAULT_COLUMNS_ORDER,
THREAD_DEFAULT_COLUMNS_WIDTH,
  SlaHelper, NAME } from '../../../base/threads';
import { selectors as accountSelectors } from '../../../base/account';
import ThreadTableStatusCell from './ThreadTableStatusCell';
import ThreadTableLanguage from './ThreadTableLanguage';
import { MarketplaceIcon } from '../../common';
import ThreadTableContextItem from './ThreadTableContextItem';
import GradientTag from '../../core/GradientTag';

const MAX_HOVERED_MESSAGE_LENGTH = 250;

class ThreadTable extends React.PureComponent {
  static propTypes = {
    i18n: PropTypes.object.isRequired,
    history: PropTypes.object.isRequired,
    threads: PropTypes.array,
    sorter: PropTypes.object,
    threadsColumns: PropTypes.object,
    isResetInitialColumnSettings: PropTypes.bool,
    isVisibleSnoozeTillColumn: PropTypes.bool,
    isVisibleDelayedUntillColumn: PropTypes.bool,
    className: PropTypes.object,
    initialColumnsWidth: PropTypes.object,
    initialColumnsOrder: PropTypes.array,
    getPaginationModel: PropTypes.func.isRequired,
    hasAiFeature: PropTypes.bool,
    companyHasAutoAiSummary: PropTypes.bool,
    companyAiCategoriesEnabled: PropTypes.bool,
    isAiSummaryEnabled: PropTypes.bool
  }

  static defaultProps = {
    threads: [],
    sorter: null,
    threadsColumns: {},
    isVisibleSnoozeTillColumn: false,
```

```
      isResetInitialColumnSettings: false,
      isVisibleDelayedUntillColumn: false,
      className: undefined,
      initialColumnsWidth: TableStorageService.getColumnsWidths(NAME,
THREAD_DEFAULT_COLUMNS_WIDTH),
      initialColumnsOrder: TableStorageService.getColumnsOrder(NAME) ||
THREAD_DEFAULT_COLUMNS_ORDER,
      hasAiFeature: false,
      companyHasAutoAiSummary: false,
      companyAiCategoriesEnabled: false,
      isAiSummaryEnabled: false
    }

    state = {
      contextMenuThreadId: null,
      contextMenuX: null,
      contextMenuY: null,
      mouseOverThreadId: null
    };

    get columns() {
      const columns = [];
      const { i18n, sorter, threadsColumns, isVisibleSnoozeTillColumn,
isVisibleDelayedUntillColumn,
        hasAiFeature, companyAiCategoriesEnabled } = this.props;
      if (threadsColumns[THREAD_TABLE_COLUMNS_ENUM.numberId])
{
        columns.push(
          {
            title: i18n.t('threads.tableColumns.numberId'),
            key: THREAD_TABLE_COLUMNS.numberId,
            sorter: true,
            dataIndex: THREAD_TABLE_COLUMNS.numberId,
            render: this.renderItem,
            sortOrder: sorter && sorter.columnKey ===
THREAD_TABLE_COLUMNS.numberId && sorter.order
          }
        );
      }
      if (threadsColumns[THREAD_TABLE_COLUMNS_ENUM.folder]) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.label'),
            key: THREAD_TABLE_COLUMNS.folder,
            render: this.renderFolder
          }
        );
      }
      if
(threadsColumns[THREAD_TABLE_COLUMNS_ENUM.aiCategories] &&
hasAiFeature && companyAiCategoriesEnabled) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.aiCategories'),
            key: THREAD_TABLE_COLUMNS.aiCategories,
            render: this.renderAiCategories
          }
        );
      }
      if
(threadsColumns[THREAD_TABLE_COLUMNS_ENUM.marketplace]) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.source'),
            key: THREAD_TABLE_COLUMNS.marketplace,
            render: this.renderImage
          }
        );
      }
      if
(threadsColumns[THREAD_TABLE_COLUMNS_ENUM.accountName]) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.account'),
            key: THREAD_TABLE_COLUMNS.accountName,
            dataIndex: THREAD_TABLE_COLUMNS.accountName,
            render: this.renderItem
          }
        );
      }
      if
(threadsColumns[THREAD_TABLE_COLUMNS_ENUM.customerName]) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.customer'),
            key: THREAD_TABLE_COLUMNS.customerName,
            dataIndex: THREAD_TABLE_COLUMNS.customerName,
            render: this.renderItem
          }
        );
      }
      if
(threadsColumns[THREAD_TABLE_COLUMNS_ENUM.messagesCount]) {
        columns.push(
          {
            key: THREAD_TABLE_COLUMNS.messagesCount,
            render: this.renderMessagesCount
          }
        );
      }
      if (threadsColumns[THREAD_TABLE_COLUMNS_ENUM.subject]) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.subject'),
            key: THREAD_TABLE_COLUMNS.subject,
            render: this.renderItemSubject
          }
        );
      }
      if
(threadsColumns[THREAD_TABLE_COLUMNS_ENUM.assignedUserName
s]) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.assignedUser'),
            key: THREAD_TABLE_COLUMNS.assignedUserNames,
            dataIndex: THREAD_TABLE_COLUMNS.assignedUserNames,
            render: this.renderAssignedUserName
          }
        );
      }
      if (isVisibleSnoozeTillColumn &&
threadsColumns[THREAD_TABLE_COLUMNS_ENUM.snoozeTill]) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.snoozeTill'),
            key: THREAD_TABLE_COLUMNS.snoozeTill,
            dataIndex: THREAD_TABLE_COLUMNS.snoozeTill,
            render: this.renderSnoozeTill
          }
        );
      }
      if
(threadsColumns[THREAD_TABLE_COLUMNS_ENUM.respondWithin]) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.respondTill'),
            key: THREAD_TABLE_COLUMNS.respondWithin,
            render: this.renderRespondWithin,
            sorter: true,
            sortOrder: sorter && sorter.columnKey ===
THREAD_TABLE_COLUMNS.respondWithin && sorter.order
          }
        );
      }
      if
(threadsColumns[THREAD_TABLE_COLUMNS_ENUM.resolveWithin]) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.resolveTill'),
            key: THREAD_TABLE_COLUMNS.resolveWithin,
            render: this.renderResolveWithin,
            sorter: true,
            sortOrder: sorter && sorter.columnKey ===
THREAD_TABLE_COLUMNS.resolveWithin && sorter.order
          }
        );
      }
      if
(threadsColumns[THREAD_TABLE_COLUMNS_ENUM.createdDate]) {
        columns.push(
          {
            title: i18n.t('threads.tableColumns.createdDate'),
            key: THREAD_TABLE_COLUMNS.createdDate,
            sorter: true,
            dataIndex: THREAD_TABLE_COLUMNS.createdDate,
            render: toLocalTime.toUserFriendlyFormat,
            sortOrder: sorter && sorter.columnKey ===
THREAD_TABLE_COLUMNS.createdDate && sorter.order
          }
        );
      }
      if
(threadsColumns[THREAD_TABLE_COLUMNS_ENUM.lastMessageDate])
{
        columns.push(
```

```
            {
                title: i18n.t('threads.tableColumns.lastMessageDate'),
                key: THREAD_TABLE_COLUMNS.lastMessageDate,
                sorter: true,
                dataIndex: THREAD_TABLE_COLUMNS.lastMessageDate,
                render: toLocalTime.toUserFriendlyFormat,
                sortOrder: sorter && sorter.columnKey ===
THREAD_TABLE_COLUMNS.lastMessageDate && sorter.order
            }
        );
    }
    if (isVisibleDelayedUntillColumn &&
threadsColumns[THREAD_TABLE_COLUMNS_ENUM.delayedDate]) {
        columns.push(
            {
                title: i18n.t('threads.tableColumns.delayedUntil'),
                key: THREAD_TABLE_COLUMNS.delayedDate,
                render: this.renderDelayDate
            }
        );
    }
    if (threadsColumns[THREAD_TABLE_COLUMNS_ENUM.language]) {
        columns.push(
            {
                title: i18n.t('threads.tableColumns.language'),
                key: THREAD_TABLE_COLUMNS.language,
                render: this.renderLanguage,
                align: 'center'
            }
        );
    }
    return columns;
  }

  renderItem = value => (
    <div>{_.truncate(value, { length: 100 })}</div>
  )

  renderItemSubject = item => {
    const { mouseOverThreadId } = this.state;
    const { hasAiFeature, isAiSummaryEnabled } = this.props;

    const isSnoozedBefore = _.get(item, 'snoozeTill') &&
moment.utc(_.get(item, 'snoozeTill')).local().isBefore(moment());
    let hoveredMessage = item?.lastMessageText || item?.subject || '...';
    hoveredMessage =
hoveredMessage.replace(htmlParser.getHtmlFullTagWithContent(hoveredMess
age, 'style'), '');
    hoveredMessage = (new
DOMParser()).parseFromString(hoveredMessage,
"text/html").documentElement.textContent;
    hoveredMessage = hoveredMessage.replace(/\t| {2}/g, '');

    if (hoveredMessage && hoveredMessage.length >
MAX_HOVERED_MESSAGE_LENGTH) {
      hoveredMessage = (hoveredMessage.substring(0,
MAX_HOVERED_MESSAGE_LENGTH).concat("..."));
    }

    if (hasAiFeature && isAiSummaryEnabled && item?.summary) {
      return (
        <Popover
          trigger="hover"
          visible={mouseOverThreadId && item?.id ===
mouseOverThreadId}
          placement="bottom"
          overlayClassName={styles.summaryPopover}
          content={this.summaryPopoverContent(item?.summary,
hoveredMessage)}>
          <div className={css(styles.summaryPopoverContainer)}>
            {this.renderItem(getObjectValue(item, 'subject'))}
          </div>
        </Popover>);
    }

    return (
      <Tooltip title={hoveredMessage} className={[styles.tooltip,
styles.tooltipText]} insecure={true}>
        <div className={css(styles.subjectInnerCell)}>
          <p className={css(styles.snoozedAgo)}>
            {isSnoozedBefore ?
              valueBuilder.getSnoozedValue(item.snoozeTill,
this.props.i18n) : null
            }
          </p>
          {this.renderItem(getObjectValue(item, 'subject'))}
        </div>
      </Tooltip>
    );
  }

  renderImage = marketplace => (
    <MarketplaceIcon
      accountId={marketplace.accountId}
      marketplace={marketplace.marketplace}
      size={spacing.s4}
    />
  )

  displayUserName = (name, index) => (
    <div key={name.concat(index)}>{_.truncate(name, { length: 100
})}</div>
  );

  renderAssignedUserName = value => {
    if (value)
    {
      try {
        const names = JSON.parse(value);
        return names ? (
          _.map(names, this.displayUserName)
        ) : null;
      }
      catch (e) {
        //ignoring parse error
      }
    }
    return null;
  };

  renderMessagesCount = object => (
    <ThreadTableStatusCell threadId={object.id} />
  )

  renderFolderItem = (folder, index) => (
    <Tag
      key={folder.Name.concat(index)}
      color={folder.Color}
      text={folder.Name}
    />
  );

  renderFolder = object => {
    try {
      const folders = object.folder ? JSON.parse(object.folder) : null;
      return (
        folders && folders.length ? (
          folders.map(this.renderFolderItem)
        ) : null
      );
    }
    catch (e) {
      console.warn(e);
      return null;
    }
  };

  renderAiCategoryItem = (aiCategory, index) => (
    <GradientTag
      key={aiCategory.Name.concat(index)}
      gradientColors={[aiCategory.ColorGradientFrom,
aiCategory.ColorGradientTo ?? aiCategory.ColorGradientFrom]}
      text={aiCategory.ShortName ?? aiCategory.Name}
    />
  );

  renderAiCategories = object => {
    try {
      const aiCategories = object.aiCategories ?
JSON.parse(object.aiCategories) : null;
      return (
        aiCategories && aiCategories.length ? (
          aiCategories.map(this.renderAiCategoryItem)
        ) : null
      );
    }
    catch (e) {
      console.warn(e);
      return null;
    }
  };

  renderRespondWithin = object => {
    const { i18n } = this.props;
    const value = SlaHelper.getRespondTillValue(object, i18n);
    if (SlaHelper.isVisibleRespondProgress(object)) {
```

```
            const progress = SlaHelper.getRespondProgress(object);
            const [startColor, endColor] =
SlaHelper.getProgressGradient(progress);
            return (
                <>
                    <p>{value}</p>
                    <ProgressLine
                        progress={progress}
                        startColor={startColor}
                        endColor={endColor}
                    />
                </>
            );
        }
        let style;
        if (object.status === 3) {
            style = styles.resolvedCell;
        }
        else if (object.priorityId && object.lastMessageIsFromCompany) {
            style = styles.respondedCell;
        }
        else if (SlaHelper.isOverdueRespond(object)) {
            style = styles.overdueCell;
        }
        else if (object.unread) {
            style = styles.defaultCell;
        }
        return (
            <p className={css(styles.boldText, style)}>
                {value}
            </p>
        );
    }

    renderResolveWithin = object => {
        const { i18n } = this.props;
        const value = SlaHelper.getResolveTillValue(object, i18n);
        if (SlaHelper.isVisibleResolveProgress(object)) {
            const progress = SlaHelper.getResolveProgress(object);
            const [startColor, endColor] =
SlaHelper.getProgressGradient(progress);
            return (
                <>
                    <p>{value}</p>
                    <ProgressLine
                        progress={progress}
                        startColor={startColor}
                        endColor={endColor}
                    />
                </>
            );
        }
        let style;
        if (object.unread) {
            style = styles.defaultCell;
        }
        if (SlaHelper.isOverdueResolve(object)) {
            style = styles.overdueCell;
        }
        if (object.status === 3) {
            style = styles.resolvedCell;
        }
        return (
            <p className={css(styles.boldText, style)}>
                {value}
            </p>
        );
    }

    renderSnoozeTill = value => {
        const snoozedValue = valueBuilder.getSnoozedValue(value,
this.props.i18n);
        if (snoozedValue && moment.utc(value).local().isAfter(moment())) {
            return (
                <p>{snoozedValue}</p>
            );
        }
        return null;
    };

    renderDelayDate = object => (
        object.isDelayed && object.delayedDate ?
valueBuilder.formatDate(toLocalTime.toLocalDateTimeSeconds(object.delaye
dDate)) : ""
    );

    renderLanguage = thread => <ThreadTableLanguage threadId={thread.id}
/>;

    generateTableRowStyles = record => {
        const { contextMenuThreadId } = this.state;
        let classNames = [];
        if (record.unread) {
            classNames = [...classNames, css(styles.unreadRow)];
        } else {
            classNames = [...classNames, css(styles.readRow)];
        }

        if (record.status === 3) {
            classNames = [...classNames, css(styles.coloredRow,
styles.resolvedRow)];
        } else if (record.priorityId && record.lastMessageIsFromCompany) {
            classNames = [...classNames, css(styles.coloredRow,
styles.respondedRow)];
        } else if (record.priority && record.respondWithin) {
            const respondTime = moment(record.respondWithin);
            const diff = moment().diff(respondTime, "minutes");
            if (diff > 0) {
                classNames = [...classNames, css(styles.coloredRow,
styles.overdueRow)];
            } else {
                classNames = [...classNames, css(styles.coloredRow,
styles.notOverdueRow)];
            }
        }

        if (record.id === contextMenuThreadId) {
            classNames = [...classNames, css(styles.selectedRow)];
        }
        return classNames.length ? _.join(classNames, ' ') : '';
    };

    onMouseClick = (thread, e) => {
        if (e) {
            if (e.ctrlKey || e.metaKey) {
                window.open(ROUTES.ticketDetails(thread.id), "_blank");
            }
            else {
                this.props.history.push(ROUTES.ticketDetails(thread.id));
            }
        }
    }

    onContextMenu = (thread, e) => {
        e.preventDefault();
        this.setState({
            contextMenuThreadId: thread.id,
            contextMenuX: e.clientX,
            contextMenuY: e.clientY
        });
    }

    onCloseContextMenu = () => {
        this.setState({
            contextMenuThreadId: null,
            contextMenuX: null,
            contextMenuY: null
        });
    }

    onMouseOver = (thread, e) => {
        e.preventDefault();
        this.setState({
            mouseOverThreadId: thread.id
        });
    }

    onMouseLeave = () => {
        this.setState({
            mouseOverThreadId: null
        });
    }

    saveColumnsWidth = columnsWidth => {
        TableStorageService.setColumnsWidths(NAME, columnsWidth,
THREAD_TABLE_COLUMNS);
    }

    saveColumnsOrder = columnsOrder => {
        TableStorageService.setColumnsOrder(NAME, columnsOrder);
    }

    summaryFormattedContent = summary => {
        const summaryBulletPoints = summary.split("\r\n");
        return summaryBulletPoints.map((line, index) => {
```

```
        return (<li className={css(styles.li, index ===
summaryBulletPoints.length - 1 && styles.liLast)}>{line}</li>);
      });
  }

  summaryPopoverContent = (summary, hoveredMessage) => {
    const { companyHasAutoAiSummary } = this.props;

    return (
      <div className={css(styles.border)}>
        <div className={css(styles.scrollInsideBorder)}>
          <div className={css(styles.popoverContent)}>
            <h4
className={css(styles.popoverTitles)}>{this.props.i18n.t('threads.aiAssistant.s
ummary')}</h4>
            <ul className={css(styles.ul)}>
              {this.summaryFormattedContent(summary)}
            </ul>
          </div>
          {!companyHasAutoAiSummary ?
            <>
              <Divider />
              <h4 className={css(styles.popoverTitles)}>Last
message</h4>
              <div
className={css(styles.popoverContentLastMessageContent)}>
                {hoveredMessage}
              </div>
            </> : null}
        </div>
      </div>
    );
  };

  render() {
    const { threads, className, initialColumnsWidth, initialColumnsOrder,
isResetInitialColumnSettings,
      getPaginationModel, hasAiFeature, ...props } = this.props;
    const { contextMenuThreadId, contextMenuX, contextMenuY } =
this.state;
    return (
      <>
        <TableResizable
          {...props}
          onPressRow={this.onMouseClick}
          columns={this.columns}
          data={threads}
          className={className || styles.table}
          rowClassName={this.generateTableRowStyles}
          initialColumnsWidth={initialColumnsWidth}
          saveColumnsWidth={this.saveColumnsWidth}
          initialColumnsOrder={initialColumnsOrder}

defaultColumnsOrder={THREAD_DEFAULT_COLUMNS_ORDER}
          saveColumnsOrder={this.saveColumnsOrder}
          resetInitialColumnSettings={isResetInitialColumnSettings}
          onContextMenu={this.onContextMenu}
          onMouseEnter={hasAiFeature && this.onMouseOver}
          onMouseLeave={hasAiFeature && this.onMouseLeave}
        />
        {contextMenuThreadId ?
          <ThreadTableContextItem
            threadId={contextMenuThreadId}
            positionX={contextMenuX}
            positionY={contextMenuY}
            onClose={this.onCloseContextMenu}
            threadsParams={getPaginationModel()}
          /> : null
        }
      </>
    );
  }
}

const mapStateToProps = state => ({
  threadsColumns:
accountSelectors.getCurrentUserThreadTableColumnsMapped(state),
  isResetInitialColumnSettings:
accountSelectors.isResettingThreadTableColumnsSetting(state),
  hasAiFeature: accountSelectors.hasAiFeature(state),
  companyHasAutoAiSummary:
accountSelectors.companyHasAutoAiSummary(state),
  companyAiCategoriesEnabled:
accountSelectors.companyAiCategoriesEnabled(state),
  isAiSummaryEnabled: accountSelectors.isAiSummaryEnabled(state)
});

export default
withRouter(connect(mapStateToProps)(translate()(ThreadTable)));
const styles = StyleSheet.create({
  table: {
    marginTop: spacing.s4
  },
  unreadRow: {
    backgroundColor: colors.tableUnreadRow,
    '& div': {
      fontWeight: fonts.mdBold.fontWeight,
      color: colors.tableTextBrighter
    }
  },
  readRow: {
    backgroundColor: colors.tableReadRow
  },
  coloredRow: {
    '& td:nth-of-type(1)': {
      position: 'relative',
      height: '100%'
    },
    '& td:nth-of-type(1):after': {
      content: '" "',
      position: 'absolute',
      left: 0,
      top: 0,
      bottom: 0
    }
  },
  resolvedRow: {
    '& td:nth-of-type(1):after': {
      borderLeft: `${spacing.s0}px solid ${colors.green}`
    }
  },
  overdueRow: {
    '& td:nth-of-type(1):after': {
      borderLeft: `${spacing.s0}px solid ${colors.red}`
    }
  },
  notOverdueRow: {
    '& td:nth-of-type(1):after': {
      borderLeft: `${spacing.s0}px solid ${colors.yellow}`
    }
  },
  respondedRow: {
    '& td:nth-of-type(1):after': {
      borderLeft: `${spacing.s0}px solid ${colors.yellow}`
    }
  },
  selectedRow: {
    transform: 'scale(1)',
    boxShadow: `0 1px ${6}px ${6}px ${colors.shadow}`,
    backgroundColor: colors.white
  },
  resolvedCell: {
    color: `${colors.green} !important`
  },
  overdueCell: {
    color: `${colors.red} !important`
  },
  respondedCell: {
    color: `${colors.yellow} !important`
  },
  boldText: {
    ...fonts.xsBold
  },
  defaultCell: {
    color: colors.tableText
  },
  snoozedAgo: {
    ...fonts.xsBold,
    color: `${colors.red} !important`
  },
  tooltip: {
    maxWidth: '50vw',
    '@media (max-width: 800px)': {
      maxWidth: '90vw'
    },
    fontWeight: `${fonts.smRegular.fontWeight} !important`
  },
  tooltipText: {
    color: `${colors.whiteDefault} !important`
  },
  subjectInnerCell: {
    margin: `-${spacing.s3}px !important`,
    padding: `${spacing.s3}px !important`
  },
  popoverContent: {
```

```
    whiteSpace: 'pre-line',
    padding: '10px 10px 0px 10px',
    width: '300px !important'
  },
  popoverContentLastMessageContent: {
    padding: '0px 10px 10px 20px',
    width: '300px !important'
  },
  popoverTitles: {
    color: colors.text,
    paddingLeft: '30px',
    paddingBottom: '5px'
  },
  summaryPopoverContainer: {
    margin: `-${spacing.s3}px !important`,
    padding: `${spacing.s3}px !important`
  },
  summaryPopover: {
    position: 'fixed !important',
    left: '50% !important'
  },
  li: {
    marginBottom: '10px'
  },
  liLast: {
    marginBottom: '0px'
  },
  ul: {
    paddingLeft: `${spacing.s4}px !important`
  },
  border: {
    padding: '4px',
    border: '2px transparent',
    borderRadius: '5px',
    backgroundImage: `linear-gradient(${colors.white}, ${colors.white}),
linear-gradient(-45deg, #ffc800B3, #ff02f0B3, #8930fdB3, #49ccf9B3)`,
    backgroundOrigin: 'border-box',
    backgroundClip: 'content-box, border-box'
  },
  scrollInsideBorder: {
    maxHeight: '600px',
    overflowY: 'auto',
    scrollbarWidth: 'thin'
  }
});
```

A.5 – Код контролера Thread

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Replyco.API.Controllers.Base;
using Replyco.API.Helpers.Attributes;
using Replyco.Core.Entities.DTO.Auth;
using Replyco.Core.Entities.DTO.Filters;
using Replyco.Core.Entities.DTO.Thread;
using Replyco.Core.Enums;
using Replyco.Domain.Core.Models.DTO;
using Replyco.Domain.Managers.Interfaces.App;
using Replyco.Domain.Managers.Interfaces.App.TablesForReading;
using Replyco.Models.Domain.App;
using Replyco.Models.DTO;
using static Replyco.Core.Helpers.AppConstants;
using TaskStatus = Replyco.Core.Enums.TaskStatus;

namespace Replyco.API.Controllers.App;

[Authorize]
public class ThreadController : BaseController<ThreadModel>
{
  private IThreadManager _threadManager;
  private readonly IThreadForReadingManager _threadForReadingManager;

  public ThreadController(IThreadManager manager,
    IThreadForReadingManager threadForReadingManager) : base(manager)
  {
    _threadManager = manager;
    _threadForReadingManager = threadForReadingManager;
  }

  [HttpPost]
  [ApiExplorerSettings(GroupName = "integration")]
  public override async Task<IActionResult> Create([FromBody]
ThreadModel model)
  {
    return Ok(await _threadManager.CreateThread(model));
  }

  [HttpPost("create")]
  [ApiExplorerSettings(GroupName = "integration")]
  public async Task<IActionResult> Create([FromBody]
CreateThreadModelDTO model)
  {
    return Ok(await _threadManager.CreateThread(model));
  }

  [HttpGet("paged")]
  [ApiExplorerSettings(GroupName = "integration")]
  public override async Task<IActionResult> GetAsync(int? page = 1, int?
count = DefaultPageSize, string term = null, string sort = null, bool asc = true)
  {
    return Ok(await Manager.GetAsync(page, count, term, sort, asc));
  }

  [HttpGet("simplifiedThread/{id}")]
  public async Task<IActionResult> GetSimplifiedThreadAsync(Guid id)
  {
    return Ok(await _threadManager.GetThreadSimplified(id));
  }

  [HttpGet("parentThread/{id}")]
  public async Task<IActionResult> GetParentThread(Guid id)
  {
    return Ok(await _threadManager.GetThreadParent(id));
  }

  [HttpGet("childrenThreads/{id}")]
  [ApiExplorerSettings(GroupName = "integration")]
  public async Task<IActionResult> GetChildrenThreads(Guid id)
  {
    return Ok(await _threadManager.GetChildrenThreads(id));
  }

  [MarkThreadAsRead]
  [ChangesLog(ChangesLogActionEnum.Resolve)]
  [HttpGet("markResolved")]
  [ApiExplorerSettings(GroupName = "integration")]
  public async Task<IActionResult> MarkThreadAsResolved([FromHeader]
Guid threadId)
  {
    await _threadManager.MarkThreadAsResolved(threadId);
    return Ok();
  }

  [ChangesLog(ChangesLogActionEnum.Resolve, true)]
  [HttpPost("markResolved")]
  [ApiExplorerSettings(GroupName = "integration")]
  public async Task<IActionResult>
MarkThreadAsResolved([FromBody]IEnumerable<Guid> id)
  {
    return Ok(await _threadManager.MarkThreadAsResolved(id));
  }

  [MarkThreadAsRead]
  [ChangesLog(ChangesLogActionEnum.Unresolve)]
  [HttpGet("markUnresolved")]
  [ApiExplorerSettings(GroupName = "integration")]
  public async Task<IActionResult> MarkThreadAsUnresolved([FromHeader]
Guid threadId)
  {
    await _threadManager.MarkThreadAsNotResolved(threadId);
    return Ok();
  }

  [ChangesLog(ChangesLogActionEnum.Unresolve, true)]
  [HttpPost("markUnresolved")]
  [ApiExplorerSettings(GroupName = "integration")]
  public async Task<IActionResult>
MarkThreadAsUnresolved([FromBody]IEnumerable<Guid> id)
  {
    return Ok(await _threadManager.MarkThreadAsNotResolved(id));
  }

  [MarkThreadAsRead]
  [ChangesLog(ChangesLogActionEnum.ArchiveTicket)]
  [HttpGet("remove/{inActive}")]
  public async Task<IActionResult> MarkThreadAsRemoved([FromHeader]
Guid threadId, bool inActive)
  {
```

```
        await _threadManager.MarkThreadAsRemoved(new List<Guid>() {
threadId }, inActive);
        return Ok();
    }

    [ChangesLog(ChangesLogActionEnum.ArchiveTicket, true)]
    [HttpPost("remove/{inActive}")]
    public async Task<IActionResult>
MarkThreadsAsRemoved([FromBody]IEnumerable<Guid> ids, bool inActive)
    {
        await _threadManager.MarkThreadAsRemoved(ids, inActive);
        return Ok();
    }

    [ChangesLog(ChangesLogActionEnum.MergeTickets)]
    [HttpPost("merge")]
    public async Task<IActionResult> MergeThreads([FromBody]
IEnumerable<Guid> ids)
    {
        await _threadManager.MergeThreads(ids);
        return Ok();
    }

    [HttpGet("byProduct/{productVariationId}")]
    public async Task<IActionResult> GetByProductId(Guid
productVariationId)
    {
        return Ok(await _threadManager.GetByProductId(productVariationId));
    }

    [HttpGet("byOrder/{orderId}")]
    public async Task<IActionResult> GetByOrderId(Guid orderId)
    {
        return Ok(await
_threadForReadingManager.GetByOrderIdAsync(orderId));
    }

    [HttpGet("byFeedback/{feedbackId}")]
    public async Task<IActionResult> GetByFeedbackId(Guid feedbackId)
    {
        return Ok(await
_threadForReadingManager.GetByFeedbackIdAsync(feedbackId));
    }

    [HttpGet("allAndUnread")]
    public async Task<IActionResult> GetAllAndUnread()
    {
        return Ok(await _threadManager.GetAllAndUnreadAsync());
    }

    [MarkThreadAsRead]
    [ChangesLog(ChangesLogActionEnum.AssignPriority)]
    [HttpGet("assignpriority/{id}")]
    public async Task<IActionResult> AssignPriority([FromHeader] Guid
threadId, Guid? id)
    {
        return Ok(await _threadManager.AssignPriorityAsync(threadId, id));
    }

    [MarkThreadAsRead]
    [ChangesLog(ChangesLogActionEnum.RemoveOrder)]
    [HttpGet("removeOrder")]
    public async Task<IActionResult> RemoveOrderFromThread([FromHeader]
Guid threadId)
    {
        await _threadManager.RemoveOrderFromThread(threadId);
        return Ok();
    }

    [MarkThreadAsRead]
    [ChangesLog(ChangesLogActionEnum.AssignUser)]
    [HttpGet("assignUser/{userId}")]
    public async Task<IActionResult> AssignUser([FromHeader] Guid threadId,
Guid userId)
    {
        await _threadManager.AssignUserAsync(userId, new List<Guid>()
        {
            threadId
        });
        return Ok();
    }

    [ChangesLog(ChangesLogActionEnum.AssignUser, true)]
    [HttpPost("assignUser/{userId}")]
    public async Task<IActionResult> AssignUser(Guid userId,
[FromBody]IEnumerable<Guid> threadIds)
    {
        await _threadManager.AssignUserAsync(userId, threadIds);
        return Ok();
    }

    [HttpGet("bycontact/{customerId}")]
    public async Task<IActionResult> GetByCustomer(Guid customerId, int?
page = 1, int? count = DefaultPageSize)
    {
        return Ok(await
_threadForReadingManager.GetByCustomerIdAsync(customerId, page,
count));
    }

    [MarkThreadAsRead]
    [ChangesLog(ChangesLogActionEnum.SnoozeTicket)]
    [HttpPost("snoozeThread")]
    public async Task<IActionResult> SnoozeTicket([FromHeader] Guid
threadId, [FromBody] SnoozeTicketsModel model)
    {
        return Ok(await _threadManager.SnoozeTickets(new List<Guid>
        {
          threadId,
        }, model.Date));
    }

    [ChangesLog(ChangesLogActionEnum.SnoozeTicket, true)]
    [HttpPost("snooze")]
    public async Task<IActionResult>
SnoozeTickets([FromBody]SnoozeTicketsModel model)
    {
        return Ok(await _threadManager.SnoozeTickets(model.Ids, model.Date));
    }

    [MarkThreadAsRead]
    [RequiredFeature(Features.CustomFields)]
    [HttpPost("updateCustomFieldsValues")]
    public async Task<IActionResult>
UpdateCustomFieldsValues([FromHeader] Guid threadId, [FromBody]
List<ThreadCustomFieldsValueModel> values)
    {
        return Ok(await _threadManager.UpdateCustomFieldsValues(values,
threadId));
    }

    [HttpPost("csvReport")]
    public async Task<IActionResult> GetCSVReport([FromBody]
ThreadReportToCSVFilter filter)
    {
        return Ok(await _threadManager.GetCSVReport(filter));
    }

    [HttpGet("notesCount/{id}")]
    public async Task<IActionResult> GetNotesCount(Guid id)
    {
        return Ok(await _threadManager.GetNotesCount(id));
    }

    [HttpPost("ticketsForReading/paged")]
    public async Task<IActionResult>
TicketsListForReadingPaged([FromBody]ThreadFilterModel filterModel,
CancellationToken cancellationToken)
    {
        cancellationToken.ThrowIfCancellationRequested();
        return Ok(await _threadManager.GetSimplePageAsync(filterModel,
cancellationToken));
    }

    [HttpPost("nextUnread")]
    public async Task<IActionResult>
GetNextUnreadThread([FromBody]NextUnreadThreadModel
nextUnreadModel)
    {
        return Ok(await
_threadForReadingManager.GetNextUnreadThread(nextUnreadModel));
    }

    /// <summary>
    /// Return all users who assigned to unresolved tickets
    /// </summary>
    [HttpGet("unresolvedTicketsAssignedUsers")]
    public async Task<IActionResult> GetUnresolvedTicketsAssignedUsers()
    {
        return Ok(await _threadForReadingManager.SelectAll(x => x.Status !=
TaskStatus.Resolved && x.UserId != null,
            x => new UserSimplifiedModel { Id = x.UserId, FullName =
x.Username }, true));
    }

    [HttpPost("deleteForever")]
```

```
    public async Task<IActionResult> DeleteThreadsForever([FromBody]
IEnumerable<Guid> ids)
    {
        await _threadManager.DeleteThreadsForever(ids);
        return Ok();
    }
}
```

A.6 – Розмітка сторінки Tools

```
import React from 'react';
import PropTypes from 'prop-types';
import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';
import { withRouter } from "react-router-dom";
import queryString from "query-string";
import { translate } from '../../../base/core';
import { TemplatesList } from '../threadTemplates';
import { AutoresponseRulesList } from '../threadAutoresponseRules';
import Rules from '../rules/Rules';
import FeedbackRules from '../feedbackRules/FeedbackRules';
import { selectors as toolsSelectors, THREAD_AUTOMATION_TABS,
actions as toolsActions } from '../../../base/tools';
import { selectors as accountSelectors } from '../../../base/account';
import { BusinessProcess } from '../../common/Tools';
import { THREAD_MARKETPLACE_WITH_RESPONSES } from
"../../../base/common";

const { addIntegrationsToThreadTemplates,
addIntegrationsToThreadAutoResponseRules } = toolsActions;
const { withSelection } = BusinessProcess;

class ThreadBusinessProcess extends React.PureComponent {
    static propTypes = {
        i18n: PropTypes.object.isRequired,
        actions: PropTypes.object.isRequired,
        isAdding: PropTypes.bool,
        tab: PropTypes.string,
        isAddingIntegrations: PropTypes.bool,
        onChangeSelection: PropTypes.func.isRequired,
        onChangeMultipleSelection: PropTypes.func.isRequired,
        setSelectedRows: PropTypes.func.isRequired,
        selectedRows: PropTypes.array.isRequired,
        hasAutoresponsesFeature: PropTypes.bool
    }

    static defaultProps = {
        isAdding: false,
        tab: null,
        isAddingIntegrations: false,
        hasAutoresponsesFeature: false
    }

    constructor(props) {
        super(props);
        this.state = {
            currentTab: props.tab || THREAD_AUTOMATION_TABS.rules,
            searchText: '',
            isAddingSwitcher: false
        };
        this.props.setSelectedRows({
            [THREAD_AUTOMATION_TABS.templates]: [],
            [THREAD_AUTOMATION_TABS.autoresponseRules]: []
        });
    }

    componentDidUpdate(prevProps) {
        if (prevProps.tab !== this.props.tab && this.props.tab) {
            this.setState({ currentTab: this.props.tab });
        }
    }

    onChangeSelectionTemplates = (id, value) =>
this.props.onChangeSelection(id, value,
THREAD_AUTOMATION_TABS.templates);

    onChangeSelectionAutoresponses = (id, value) =>
this.props.onChangeSelection(id, value,
THREAD_AUTOMATION_TABS.autoresponseRules);

    onChangeSelectionAllTemplates = (ids, addNew) =>
this.props.onChangeMultipleSelection(ids,
THREAD_AUTOMATION_TABS.templates, addNew);

    onChangeSelectionAllAutoresponses = (ids, addNew) =>
this.props.onChangeMultipleSelection(ids,
THREAD_AUTOMATION_TABS.autoresponseRules, addNew);

    onSearch = searchText => this.setState({ searchText });

    onCreateNew = () => this.setState(prevState => ({ isAddingSwitcher:
!prevState.isAddingSwitcher }));

    onChangeTab = currentTab => this.setState({ currentTab });

    get tabs() {
        const { i18n, hasAutoresponsesFeature } = this.props;
        return [{
            tabProps: {
                key: THREAD_AUTOMATION_TABS.rules,
                tab: i18n.t('rules.rules')
            },
            component: this.renderRules
        }, {
            tabProps: {
                key: THREAD_AUTOMATION_TABS.templates,
                tab: i18n.t('templates.templates')
            },
            component: this.renderTemplates
        }, ...(hasAutoresponsesFeature ? [{
            tabProps: {
                key: THREAD_AUTOMATION_TABS.autoresponseRules,
                tab: i18n.t('autoresponse.autoresponseRules')
            },
            component: this.renderAutoresponseRules
        }] : []),
        // {
        //     tabProps: {
        //         key: THREAD_AUTOMATION_TABS.feedbackRules,
        //         tab: i18n.t('feedbackRules.feedbackRules')
        //     },
        //     component: this.renderFeedbackRules
        // }
    ];
    }

    get renderRules() {
        return (
            <Rules
                searchText={this.state.searchText}
                isActive={this.state.currentTab ===
THREAD_AUTOMATION_TABS.rules}
                isAddingSwitcher={this.state.isAddingSwitcher}
            />
        );
    }

    get renderTemplates() {
        return (
            <TemplatesList
                searchText={this.state.searchText}
                isActive={this.state.currentTab ===
THREAD_AUTOMATION_TABS.templates}
                isAddingSwitcher={this.state.isAddingSwitcher}
                onChangeSelection={this.onChangeSelectionTemplates}
                onChangeMultipleSelection={this.onChangeSelectionAllTemplates}

selectedRows={this.props.selectedRows[THREAD_AUTOMATION_TABS.te
mplates]}
            />
        );
    }

    get renderAutoresponseRules() {
        return (
            <AutoresponseRulesList
                searchText={this.state.searchText}
                isActive={this.state.currentTab ===
THREAD_AUTOMATION_TABS.autoresponseRules}
                isAddingSwitcher={this.state.isAddingSwitcher}
                onChangeSelection={this.onChangeSelectionAutoresponses}

onChangeMultipleSelection={this.onChangeSelectionAllAutoresponses}

selectedRows={this.props.selectedRows[THREAD_AUTOMATION_TABS.a
utoresponseRules]}
            />
        );
    }

    get renderFeedbackRules() {
```

```
      return (
        <FeedbackRules
          searchText={this.state.searchText}
          isActive={this.state.currentTab ===
THREAD_AUTOMATION_TABS.feedbackRules}
          isAddingSwitcher={this.state.isAddingSwitcher}
        />
      );
    }

    addIntegrations = selectedIntegrations => {
      const { currentTab } = this.state;
      const { selectedRows, setSelectedRows } = this.props;
      if (currentTab === THREAD_AUTOMATION_TABS.templates) {

this.props.actions.addIntegrationsToThreadTemplates(selectedRows[currentTa
b], selectedIntegrations);
      } else if (currentTab ===
THREAD_AUTOMATION_TABS.autoresponseRules) {

this.props.actions.addIntegrationsToThreadAutoResponseRules(selectedRows[
currentTab], selectedIntegrations);
      }
      setSelectedRows(prevState => ({
        ...prevState,
        [currentTab]: []
      }));
    }

    render() {
      const { isAdding, isAddingIntegrations } = this.props;
      const { currentTab } = this.state;
      return (
        <BusinessProcess.BusinessProcessCore
          isThreadAutomation={true}
          currentTab={currentTab}
          tabs={this.tabs}
          onSearch={this.onSearch}
          onCreateNew={this.onCreateNew}
          onChangeTab={this.onChangeTab}
          addIntegrations={this.addIntegrations}
          isAdding={isAdding}
          isAddingIntegrations={isAddingIntegrations}
          marketplaces={THREAD_MARKETPLACE_WITH_RESPONSES}
        />
      );
    }
}

const mapStateToProps = (state, ownProps) => {
  const params = queryString.parse(ownProps.location.search);
  return {
    isAdding: toolsSelectors.isAddingRule(state) ||
toolsSelectors.isAddingTemplate(state) ||
      toolsSelectors.isAddingAutoresponseRule(state),
    tab: params?.tab,
    isAddingIntegrations:
toolsSelectors.isAddingIntegrationsToThreadAutoresponsesRules(state) ||
      toolsSelectors.isAddingIntegrationsToThreadTemplates(state),
    hasAutoresponsesFeature:
accountSelectors.hasAutoresponsesFeature(state)
  };
};

const mapDispatchToProps = dispatch => ({
  actions: bindActionCreators({ addIntegrationsToThreadTemplates,
addIntegrationsToThreadAutoResponseRules }, dispatch)
});

const WrappedThreadBusinessProcess =
withRouter(withSelection(connect(mapStateToProps,
mapDispatchToProps)(translate()(ThreadBusinessProcess))));

export {
  WrappedThreadBusinessProcess as ThreadBusinessProcess
};
```

A.7 – Код контролера TemplateController

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Replyco.API.Controllers.Base;
using Replyco.API.Helpers.Attributes;
using Replyco.Core.Enums;
using Replyco.Domain.Managers.Interfaces.App;
using Replyco.Models.Domain.App;
using Replyco.Models.DTO.App;

namespace Replyco.API.Controllers.App;

[Authorize]
public class TemplateController : BaseController<TemplateModel>
{
  private readonly ITemplateManager _templateManager;

  public TemplateController(ITemplateManager manager) : base(manager)
  {
    _templateManager = manager;
  }

  [HttpGet]
  public override async Task<IActionResult> Get()
  {
    return Ok(await Manager.GetAsync());
  }

  [Permission(Permissions.BusinessProcess, PermissionMode.Edit)]
  public override Task<IActionResult> Create([FromBody] TemplateModel
model)
  {
    return base.Create(model);
  }

  [Permission(Permissions.BusinessProcess, PermissionMode.Edit)]
  public override Task<IActionResult> Update([FromBody] TemplateModel
model)
  {
    return base.Update(model);
  }

  [Permission(Permissions.BusinessProcess, PermissionMode.Edit)]
  public override Task<IActionResult> Delete(Guid id)
  {
    return base.Delete(id);
  }

  [Permission(Permissions.BusinessProcess, PermissionMode.Edit)]
  [HttpPost("integrations")]
  public async Task<IActionResult>
AddIntegrations([FromBody]AddIntegrationsToAutomationDTO data)
  {
    await _templateManager.AddIntegrations(data);
    return Ok();
  }

  [HttpPost("switch/{templateId}/{inActive}")]
  public async Task<IActionResult> SwitchTemplate(Guid templateId, bool
inActive)
  {
    await _templateManager.SwitchTemplate(templateId, inActive);
    return Ok();
  }

  [HttpGet("term")]
  public async Task<IActionResult> GetByTerm(string term = null)
  {
    return Ok(await _templateManager.GetByTermAsync(term));
  }
}
```

A.8 – Код контролера RuleController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Replyco.API.Controllers.Base;
using Replyco.Domain.Managers.Interfaces.App;
using Replyco.Models.Domain.App;
using Replyco.Models.DTO.App;

namespace Replyco.API.Controllers.App;

[Authorize]
public class MessageRuleController : BaseController<MessageRuleModel>
{
```

```
    private readonly IMessageRuleManager _messageRuleManager;
    public MessageRuleController(IMessageRuleManager manager) :
base(manager)
    {
        _messageRuleManager = manager;
    }

    [HttpGet("term")]
    public async Task<IActionResult> GetByTerm(string term = null)
    {
        return Ok(await _messageRuleManager.GetByTermAsync(term));
    }

    [HttpGet("switch/{ruleId}/{inActive}")]
    public async Task<IActionResult> SwitchRuleId(Guid ruleId, bool inActive)
    {
        await _messageRuleManager.SwitchRuleAsync(ruleId, inActive);
        return Ok();
    }

    [HttpPost]
    public override Task<IActionResult> Create([FromBody]
MessageRuleModel model)
    {
        return base.Create(model);
    }

    [HttpPut]
    public override Task<IActionResult> Update([FromBody]
MessageRuleModel model)
    {
        return base.Update(model);
    }

    [HttpDelete("{id}")]
    public override Task<IActionResult> Delete(Guid id)
    {
        return base.Delete(id);
    }

    [HttpPost("changeRulesPriority")]
    public async Task<IActionResult> ChangeRulesPriority([FromBody]
List<RulesPriority> rulesPriorities)
    {
        var prioritiesList = rulesPriorities.Select(x => x.Priority);
        if (prioritiesList.Count() != prioritiesList.Distinct().Count())
            return BadRequest("Priority should be unique for every rule");

        await _messageRuleManager.UpdateRulesPriorities(rulesPriorities);
        return Ok();
    }
}
```

A.9 – Код контролера FolderController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Replyco.API.Controllers.Base;
using Replyco.Core.Entities.DTO;
using Replyco.Domain.Managers.Interfaces.App;
using Replyco.Models.Domain.App;

namespace Replyco.API.Controllers.App;

[Authorize]
[ApiExplorerSettings(GroupName = "integration")]
public class FolderController : BaseController<FolderModel>
{
    private readonly IFolderManager _folderManager;
    private readonly IUserFoldersManager _userFoldersManager;

    public FolderController(IFolderManager manager, IUserFoldersManager
userFoldersManager) : base(manager)
    {
        _folderManager = manager;
        _userFoldersManager = userFoldersManager;
    }

    [HttpPost("assign/{userId}")]
    public async Task<IActionResult> AssignFolders(Guid userId,
[FromBody]FolderModel[] folders)
    {
        await _userFoldersManager.AssignUser(userId, folders);
        return Ok();
    }

    [HttpGet("byUserId/{userId:guid?}")]
    public async Task<IActionResult> GetByUserId(Guid? userId = null)
    {
        return Ok(await _userFoldersManager.GetByUserId(userId));
    }

    [HttpGet("term")]
    public async Task<IActionResult> GetByTerm(string term = null)
    {
        return Ok(await _folderManager.GetByTermAsync(term));
    }

    [HttpPost("changeFoldersPriority")]
    public async Task<IActionResult> ChangeRulesPriority([FromBody]
List<FolderOrder> folderPriorities)
    {
        if (folderPriorities.Count != folderPriorities.Select(x =>
x.Order).Distinct().ToList().Count)
            return BadRequest("Priority should be unique for every folder");

        await _folderManager.UpdateFolderOrders(folderPriorities);
        return Ok();
    }
}
```

A.10 – Код контролера AutoresponderController

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Replyco.API.Controllers.Base;
using Replyco.API.Helpers.Attributes;
using Replyco.Core.Enums;
using Replyco.Domain.Managers.Interfaces.App;
using Replyco.Models.Domain.App;
using Replyco.Models.DTO.App;

namespace Replyco.API.Controllers.App;

[Authorize]
[RequiredFeature(Features.Autoresponses)]
public class AutoresponderController : BaseController<AutoresponderModel>
{
    private readonly IAutoresponderManager _autoresponderManager;

    public AutoresponderController(IAutoresponderManager manager) :
base(manager)
    {
        _autoresponderManager = manager;
    }

    [HttpGet("term")]
    public async Task<IActionResult> GetByTerm(string term = null)
    {
        return Ok(await _autoresponderManager.GetByTermAsync(term));
    }

    [HttpPost("integrations")]
    public async Task<IActionResult> AddIntegrations([FromBody]
AddIntegrationsToAutomationDTO data)
    {
        await _autoresponderManager.AddIntegrations(data);
        return Ok();
    }

    [HttpPost("switch/{ruleId}/{inActive}")]
    public async Task<IActionResult> SwitchAutoresponder(Guid ruleId, bool
inActive)
    {
        await _autoresponderManager.SwitchAutoresponder(ruleId, inActive);
        return Ok();
    }
}
```

A.11 – Код контролера MessageController

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using FluentValidation.AspNetCore;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Replyco.API.Controllers.Base;
using Replyco.API.Helpers.Attributes;
using Replyco.Domain.Core.Managers.Interfaces;
using Replyco.Domain.Managers.Interfaces.App;
using Replyco.Models.Domain.App;
using static Replyco.Core.Helpers.AppConstants;

namespace Replyco.API.Controllers.App;

[Authorize]
[ApiExplorerSettings(GroupName = "integration")]
public class MessageController : BaseController<MessageModel>
{
    private IMessageManager _manager;
    private readonly IByteFilesManager _byteFilesManager;

    public MessageController(IMessageManager manager,
        IByteFilesManager byteFilesManager) : base(manager)
    {
        _manager = manager;
        _byteFilesManager = byteFilesManager;
    }

    //[ChangesLog(ChangesLogActionEnum.SendMessage)] //NOTE:
Commented this because of memory issue
    [MarkThreadAsRead]
    [HttpPost("send")]
    public async Task<IActionResult> SendMessage([FromBody]
MessageModel messageToSend)
    {
        return Ok(await _manager.SendMessage(messageToSend));
    }

    [HttpPost("preview")]
    public async Task<IActionResult>
GetMessagePreview([CustomizeValidator(Skip=true)][FromBody]
MessageModel messageToSend)
    {
        return Ok(await _manager.GetPreviewMessage(messageToSend));
    }

    [MarkThreadAsRead]
    [HttpGet("resend/{id}")]
    public async Task<IActionResult> ResendMessage(Guid id)
    {
        return Ok(await _manager.ResendMessage(id));
    }

    [HttpGet("threadMessages/{threadId}")]
    public async Task<IActionResult> GetByThreadId(Guid threadId)
    {
        return Ok(await _manager.GetThreadMessages(threadId));
    }

    [HttpGet("pagedThreadMessages")]
    public async Task<IActionResult> GetPagedByThreadId([FromHeader]
Guid threadId, int? page = 1, int? count = DefaultPageSize, bool asc = false)
    {
        return Ok(await _manager.GetThreadMessages(threadId, page, count,
asc));
    }

    [HttpPost("messageAttachmentsPreviews")]
    public async Task<IActionResult>
GetMessageAttachmentsPreviews([FromBody] IEnumerable<Guid> Ids)
    {
        return Ok(await
_byteFilesManager.GetMessageAttachmentsPreviewsByIds(Ids));
    }

    [MarkThreadAsRead]
    [HttpGet("cancelMessage/{messageId}")]
    public async Task<IActionResult> CancelMessage(Guid messageId)
    {
        return Ok(await _manager.CancelMessage(messageId));
    }

    [HttpGet("messageHtmlBody/{messageId}")]
    public async Task<IActionResult> GetMessageHtmlBodyById(Guid
messageId)
    {
        return Ok( await _manager.GetMessageHtmlBodyById(messageId) );
    }
}
```

A.12 – Розмітка сторінки Dashboard

```
import React from 'react';
import PropTypes from 'prop-types';
import { connect } from 'react-redux';
import { bindActionCreators } from "redux";
import { ENDPOINTS_WITH_CANCELING, translate } from
"../../../base/core";
import { DASHBOARD_STRING_TYPES, WIDGETS,
DASHBOARD_DEFAULT_LAYOUT, WIDGETS_LIST, actions as
dashboardActions } from "../../../base/dashboard";
import DashboardContent from '../layout/DashboardContent';
import { TicketsCalculation } from './TicketsCalculation';
import MessageLanguageCount from './MessageLanguageCount';
import RelationCreatedTickets from './RelationCreatedTickets';
import { DueOverdueAverage } from './DueOverdueAverage';
import { InformationAboutMessages } from './InformationAboutMessages';
import AverageTime from './AverageTime';
import BuyerLocations from './BuyerLocations';
import { DueOverduePercentage } from './DueOverduePercentage';
import { cancelTokenService } from '../../../base/shared';

const {
    getOverviewTicketsCalculation,
    getOverviewRelationCreatedTickets,
    getOverviewMessagesCount,
    getOverviewDueOverdue,
    getOverviewDueOverduePercentage,
    getOverviewAverageTime,
    getOverviewMessagesLanguageCount,
    getOverviewNotSentMessages,
    getOverviewBuyerLocations
} = dashboardActions;

const REQUEST_TYPES = {
    ticketsCalculation: 'ticketsCalculation',
    relationCreatedTickets: 'relationCreatedTickets',
    messagesCount: 'messagesCount',
    dueOverdue: 'dueOverdue',
    averageTime: 'averageTime',
    messageLanguageCount: 'messageLanguageCount',
    notSentMessages: 'notSentMessages',
    buyerLocations: 'buyerLocations',
    dueOverduePercentage: 'dueOverduePercentage'
};

class DashboardOverview extends React.PureComponent {
    static propTypes = {
        i18n: PropTypes.object.isRequired,
        actions: PropTypes.object.isRequired
    };

    componentWillUnmount() {
        cancelTokenService.cancelRequests([
            ENDPOINTS_WITH_CANCELING.ticketsCalculation,
            ENDPOINTS_WITH_CANCELING.relationCreatedTickets,
            ENDPOINTS_WITH_CANCELING.messagesCountInfo,
            ENDPOINTS_WITH_CANCELING.dueOverdueCountInfo,
            ENDPOINTS_WITH_CANCELING.dueOverduePercentageInfo,
            ENDPOINTS_WITH_CANCELING.avarageTimeInfo,
            ENDPOINTS_WITH_CANCELING.ticketLanguageCountInfo,
            ENDPOINTS_WITH_CANCELING.notSentMessagesCount,
            ENDPOINTS_WITH_CANCELING.countriesList
        ]);
    }

    get widgets() {
        return [
            { key: WIDGETS.OVERVIEW.TICKETS_CALCULATION,
component: TicketsCalculation },
            { key: WIDGETS.OVERVIEW.RELATION_CREATED_TICKETS,
component: RelationCreatedTickets },
            { key:
WIDGETS.OVERVIEW.INFORMATION_ABOUT_MESSAGES,
component: InformationAboutMessages },
            { key: WIDGETS.OVERVIEW.DUE_OVERDUE_AVERAGE,
component: DueOverdueAverage },
            { key: WIDGETS.OVERVIEW.DUE_OVERDUE_PERCENTAGE,
component: DueOverduePercentage },
            { key: WIDGETS.OVERVIEW.AVERAGE_TIME, component:
AverageTime },
```

```
        { key: WIDGETS.OVERVIEW.MESSAGE_LANGUAGE_COUNT,
component: MessageLanguageCount },
        { key: WIDGETS.OVERVIEW.BUYER_LOCATIONS, component:
BuyerLocations }
      ];
  }

  get requestWidgetsItems() {
    return [
      { request: REQUEST_TYPES.ticketsCalculation, widgets:
[WIDGETS.OVERVIEW.TICKETS_CALCULATION] },
      { request: REQUEST_TYPES.relationCreatedTickets, widgets:
[WIDGETS.OVERVIEW.RELATION_CREATED_TICKETS] },
      { request: REQUEST_TYPES.messagesCount, widgets:
[WIDGETS.OVERVIEW.INFORMATION_ABOUT_MESSAGES] },
      { request: REQUEST_TYPES.dueOverdue, widgets:
[WIDGETS.OVERVIEW.DUE_OVERDUE_AVERAGE] },
      { request: REQUEST_TYPES.dueOverduePercentage, widgets:
[WIDGETS.OVERVIEW.DUE_OVERDUE_PERCENTAGE] },
      { request: REQUEST_TYPES.averageTime, widgets:
[WIDGETS.OVERVIEW.AVERAGE_TIME] },
      { request: REQUEST_TYPES.messageLanguageCount, widgets:
[WIDGETS.OVERVIEW.MESSAGE_LANGUAGE_COUNT] },
      { request: REQUEST_TYPES.notSentMessages, widgets:
[WIDGETS.OVERVIEW.INFORMATION_ABOUT_MESSAGES] },
      { request: REQUEST_TYPES.buyerLocations, widgets:
[WIDGETS.OVERVIEW.BUYER_LOCATIONS] }
    ];
  }

  get requestActions() {
    return [
      { type: REQUEST_TYPES.ticketsCalculation, action:
this.getDashboardTicketsCalculation },
      { type: REQUEST_TYPES.relationCreatedTickets, action:
this.getDashboardRelationCreatedTickets },
      { type: REQUEST_TYPES.messagesCount, action:
this.getDashboardMessagesCount },
      { type: REQUEST_TYPES.dueOverdue, action:
this.getDashboardDueOverdue },
      { type: REQUEST_TYPES.dueOverduePercentage, action:
this.getDashboardDueOverduePercentage },
      { type: REQUEST_TYPES.averageTime, action:
this.getDashboardAverageTime },
      { type: REQUEST_TYPES.messageLanguageCount, action:
this.getDashboardMessagesLanguageCount },
      { type: REQUEST_TYPES.notSentMessages, action:
this.getDashboardNotSentMessages },
      { type: REQUEST_TYPES.buyerLocations, action:
this.getDashboardBuyerLocations }
    ];
  }

  getDashboardTicketsCalculation = params =>
this.props.actions.getOverviewTicketsCalculation(params);

  getDashboardRelationCreatedTickets = params =>
this.props.actions.getOverviewRelationCreatedTickets(params);

  getDashboardMessagesCount = params =>
this.props.actions.getOverviewMessagesCount(params);

  getDashboardDueOverdue = params =>
this.props.actions.getOverviewDueOverdue(params);

  getDashboardDueOverduePercentage = params =>
this.props.actions.getOverviewDueOverduePercentage(params);

  getDashboardAverageTime = params =>
this.props.actions.getOverviewAverageTime(params);

  getDashboardMessagesLanguageCount = params =>
this.props.actions.getOverviewMessagesLanguageCount(params);

  getDashboardNotSentMessages = params =>
this.props.actions.getOverviewNotSentMessages(params);

  getDashboardBuyerLocations = params =>
this.props.actions.getOverviewBuyerLocations(params);

  render() {
    const { i18n } = this.props;
    return (
      <DashboardContent
        type={DASHBOARD_STRING_TYPES.overview}
        widgets={this.widgets}
        widgetsList={WIDGETS_LIST.OVERVIEW}

defaultLayout={DASHBOARD_DEFAULT_LAYOUT.OVERVIEW}
        requestTypes={REQUEST_TYPES}
        requestWidgetsItems={this.requestWidgetsItems}
        requestActions={this.requestActions}
        title={i18n.t('screens.overview')}
        hasDateFilter={true}
        hasMarketplaceFilter={true}
        hasIntegrationFilter={true}
      />
    );
  }
}

const mapDispatchToProps = dispatch => ({
  actions: bindActionCreators({
    getOverviewTicketsCalculation,
    getOverviewRelationCreatedTickets,
    getOverviewMessagesCount,
    getOverviewDueOverdue,
    getOverviewDueOverduePercentage,
    getOverviewAverageTime,
    getOverviewMessagesLanguageCount,
    getOverviewNotSentMessages,
    getOverviewBuyerLocations
  }, dispatch)
});

export default connect(null,
mapDispatchToProps)(translate()(DashboardOverview));
```

A.13 – TicketsCalculation

```
import React, { useCallback } from "react";
import { useSelector } from "react-redux";
import { ICONS, ROUTES, TICKET_TYPES, useTranslation, getObjectValue
} from '../../../base/core';
import { selectors as dashboardSelectors } from '../../../base/dashboard';
import { THREAD_ADDITIONAL_FILTERS, THREAD_FILTER_NAMES }
from '../../../base/threads';
import { TwoRowsMatrix } from "./common";
import { StyleSheet, colors, spacing } from "../../../styles";

export const TicketsCalculation = () => {
  const { t } = useTranslation();
  const ticketsCalculation =
useSelector(dashboardSelectors.getOverviewTicketsCalculation);
  const isLoading =
useSelector(dashboardSelectors.isLoadingOverviewTicketsCalculation);
  const filters = useSelector(dashboardSelectors.getDashboardFilters);

  const getTopItems = useCallback(() => (
    [{
      count: getObjectValue(ticketsCalculation, 'current.createdTicketsCount',
0),
      prevCount: getObjectValue(ticketsCalculation,
'previous.createdTicketsCount', 0),
      text: t('dashboard.createdTickets'),
      tooltipText: t('dashboard.widgetDescriptions.overview.newUnread'),
      iconName: ICONS.plus,
      iconColor: colors.green,
      iconBackground: colors.greenOpacity,
      locationData: {
        pathname: ROUTES.tickets(TICKET_TYPES.all),
        state: filters
      }
    }, {
      count: getObjectValue(ticketsCalculation, 'current.newTicketsCount',
0),
      prevCount: getObjectValue(ticketsCalculation,
'previous.newTicketsCount', 0),
      text: t('dashboard.newTickets'),
      tooltipText:
t('dashboard.widgetDescriptions.overview.newUnreadWithNoResponse'),
      iconName: ICONS.message,
      iconColor: colors.primary,
      iconBackground: colors.primaryLight,
      locationData: {
        pathname: ROUTES.tickets(TICKET_TYPES.all),
        state: {
          [THREAD_FILTER_NAMES.filterTypes]:
[THREAD_ADDITIONAL_FILTERS.new],
          ...filters
        }
      }
```

```
      }, {
         count: getObjectValue(ticketsCalculation,
'current.notResolvedTicketsCount', 0),
         prevCount: getObjectValue(ticketsCalculation,
'previous.notResolvedTicketsCount', 0),
         isReverse: true,
         text: t('dashboard.notResolvedTickets'),
         tooltipText: t('dashboard.widgetDescriptions.overview.notResolved'),
         iconName: ICONS.close,
         iconColor: colors.red,
         iconBackground: colors.redOpacity,
         locationData: {
            pathname: ROUTES.tickets(TICKET_TYPES.all),
            state: {
               [THREAD_FILTER_NAMES.filterTypes]:
[THREAD_ADDITIONAL_FILTERS.notResolved],
               ...filters
            }
         }
      }, {
         count: getObjectValue(ticketsCalculation,
'current.resolvedTicketsCount', 0),
         prevCount: getObjectValue(ticketsCalculation,
'previous.resolvedTicketsCount', 0),
         text: t('dashboard.resolvedTickets'),
         tooltipText: t('dashboard.widgetDescriptions.overview.resolved'),
         iconName: ICONS.check,
         iconColor: colors.green,
         iconBackground: colors.greenOpacity,
         locationData: {
            pathname: ROUTES.tickets(TICKET_TYPES.all),
            state: {
               [THREAD_FILTER_NAMES.filterTypes]:
[THREAD_ADDITIONAL_FILTERS.resolved],
               ...filters
            }
         }
      }]
   ), [filters, ticketsCalculation]);

   const getBottomItems = useCallback(() => (
      [{
         count: getObjectValue(ticketsCalculation, 'current.snoozedCount', 0),
         prevCount: getObjectValue(ticketsCalculation,
'previous.snoozedCount', 0),
         isReverse: true,
         text: t('dashboard.snoozedTickets'),
         tooltipText: t('dashboard.widgetDescriptions.overview.snoozed'),
         iconName: ICONS.pause,
         iconColor: colors.yellow,
         iconBackground: colors.yellowOpacity,
         locationData: {
            pathname: ROUTES.tickets(TICKET_TYPES.snoozed),
            state: filters
         }
      }, {
```

A.14 – Код контролера ReportController

```
using System;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Replyco.API.Helpers.Attributes;
using Replyco.Core.Entities.DTO.Reports.ReportFilters;
using Replyco.Core.Enums;
using Replyco.Domain.Managers.Interfaces.App;

namespace Replyco.API.Controllers.App;

[Authorize]
[Produces("application/json")]
[Route("api/[controller]")]
[ApiExplorerSettings(GroupName = "v1")]
public class ReportingController : Controller
{
   private readonly IReportingManager _manager;

   public ReportingController(IReportingManager manager)
   {
      _manager = manager;
   }


   #region Overview
   [HttpPost("ticketsCalculation")]
   public async Task<IActionResult>
TicketsCalculation([FromBody]BaseReportFilter filter, CancellationToken
cancellationToken)
   {
      return Ok(await _manager.TicketsCalculation(filter, cancellationToken));
   }

   [HttpPost("compareCreatedAnsweredResolved")]
   public async Task<IActionResult>
CompareCreatedAnsweredResolved([FromBody]BaseReportFilter filter,
CancellationToken cancellationToken)
   {
      return Ok(await _manager.CompareCreatedAnsweredResolved(filter,
cancellationToken));
   }

   [HttpPost("messagesCountInfo")]
   public async Task<IActionResult>
MessagesCountInfo([FromBody]BaseReportFilter filter, CancellationToken
cancellationToken = default)
   {
      return Ok(await _manager.MessagesCountInfo(filter, cancellationToken));
   }

   [HttpPost("dueOverdueCountInfo")]
   public async Task<IActionResult>
DueOverdueCountInfo([FromBody]BaseReportFilter filter, CancellationToken
cancellationToken = default)
   {
      return Ok(await _manager.DueOverdueCountInfo(filter,
cancellationToken));
   }

   [HttpPost("dueOverduePercentageInfo")]
   public async Task<IActionResult>
DueOverduePercentageInfo([FromBody]BaseReportFilter filter,
CancellationToken cancellationToken = default)
   {
      return Ok(await _manager.DueOverduePercentageInfo(filter,
cancellationToken));
   }

   [HttpPost("avarageTimeInfo")]
   public async Task<IActionResult>
AvarageTimeInfo([FromBody]BaseReportFilter filter, CancellationToken
cancellationToken = default)
   {
      return Ok(await _manager.AvarageTime(filter, cancellationToken));
   }

   [HttpPost("ticketLanguageCountInfo")]
   public async Task<IActionResult>
TicketLanguageCountInfo([FromBody]BaseReportFilter filter,
CancellationToken cancellationToken = default)
   {
      return Ok(await _manager.TicketLanguageCountInfo(filter,
cancellationToken));
   }

   [HttpPost("notSentMessagesCount")]
   public async Task<IActionResult>
GetNotSentMessagesCount([FromBody]BaseReportFilter filter,
CancellationToken cancellationToken = default)
   {
      return Ok(await _manager.GetNotSentMessagesCount(filter,
cancellationToken));
   }
   [HttpPost("countriesList")]
   public async Task<IActionResult> GetCountriesList([FromBody]
BaseReportFilter filter, CancellationToken cancellationToken = default)
   {
      return Ok(await _manager.GetCountriesList(filter, cancellationToken));
   }
   #endregion


}
```

ДОДАТОК Б

(обов’язковий)

ПРЕЗЕНТАЦІЙНИЙ МАТЕРІАЛ

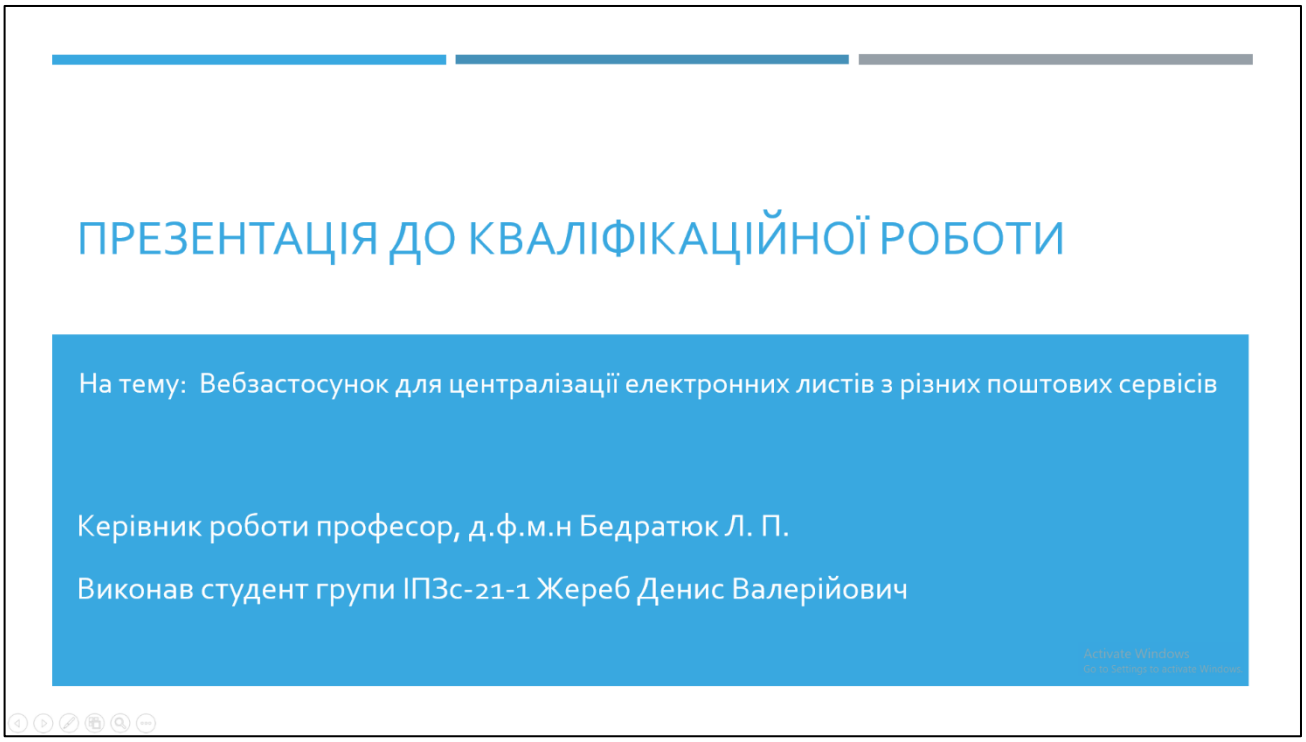


Рисунок Б.1 – Слайд №1

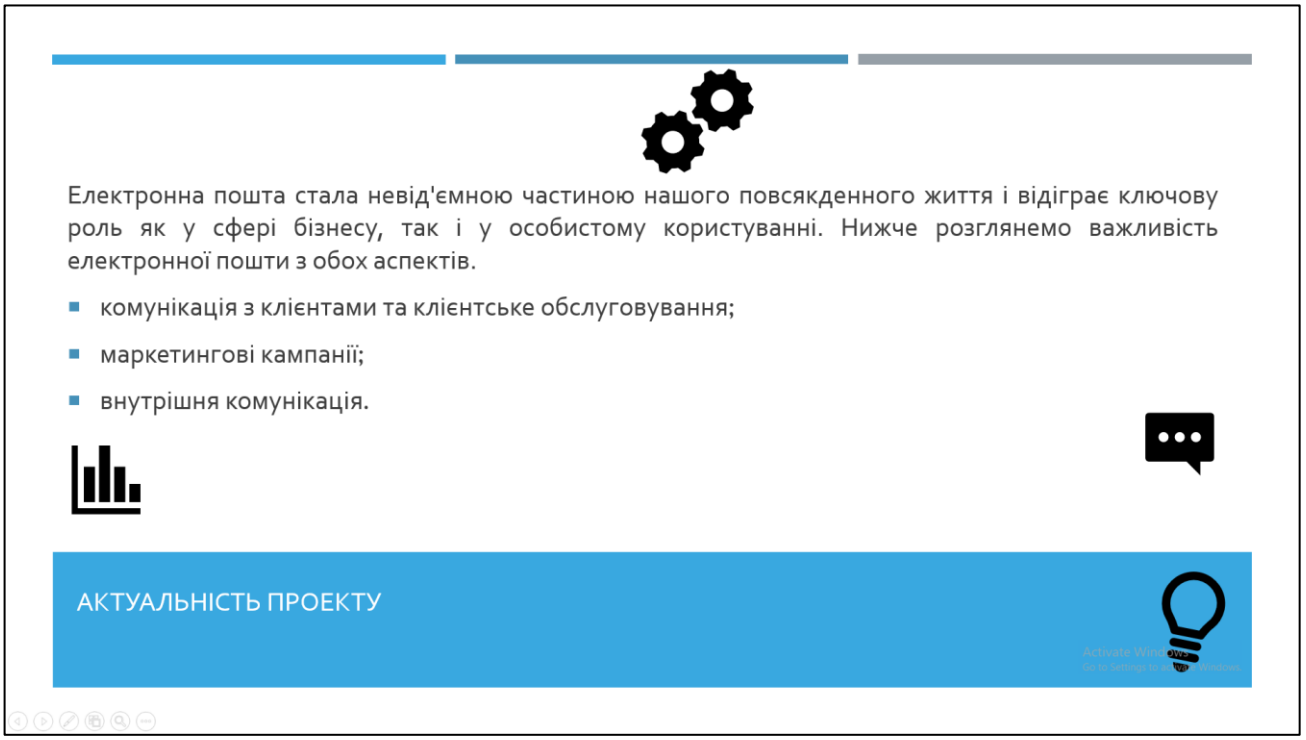


Рисунок Б.2 – Слайд №2

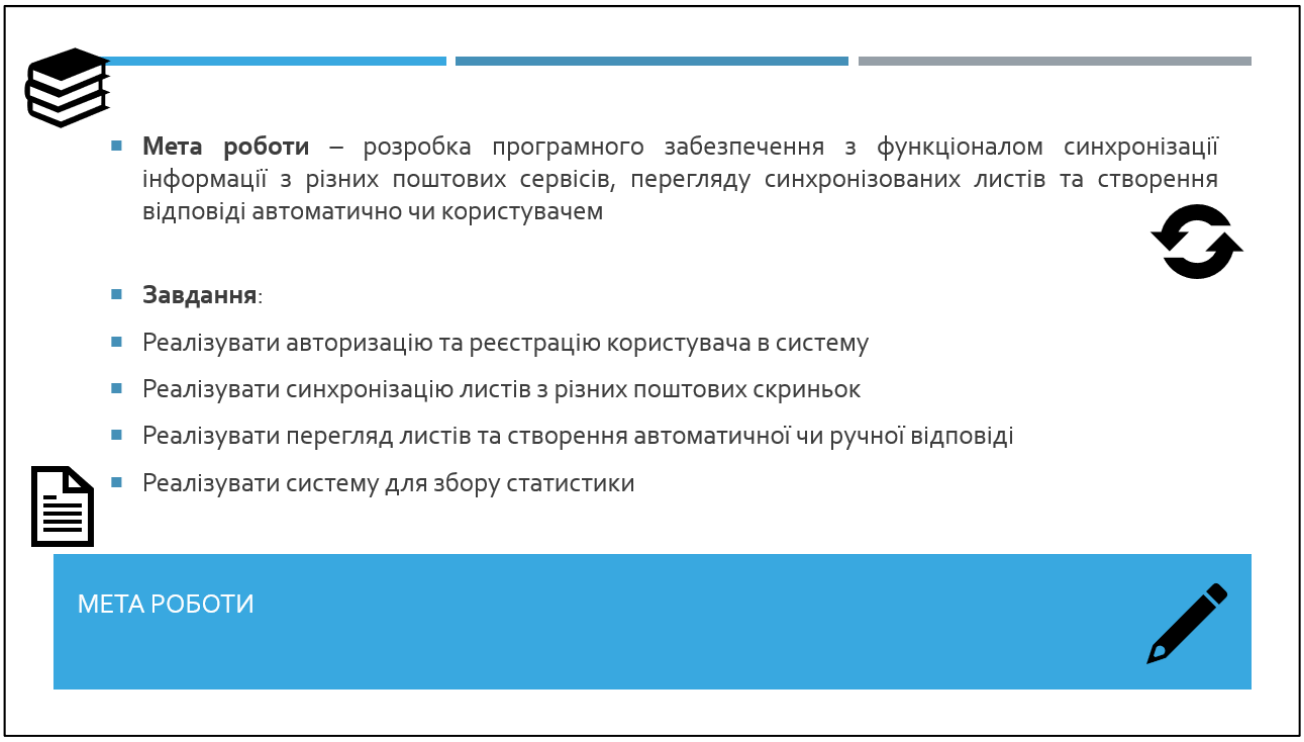


Рисунок Б.3 – Слайд №3

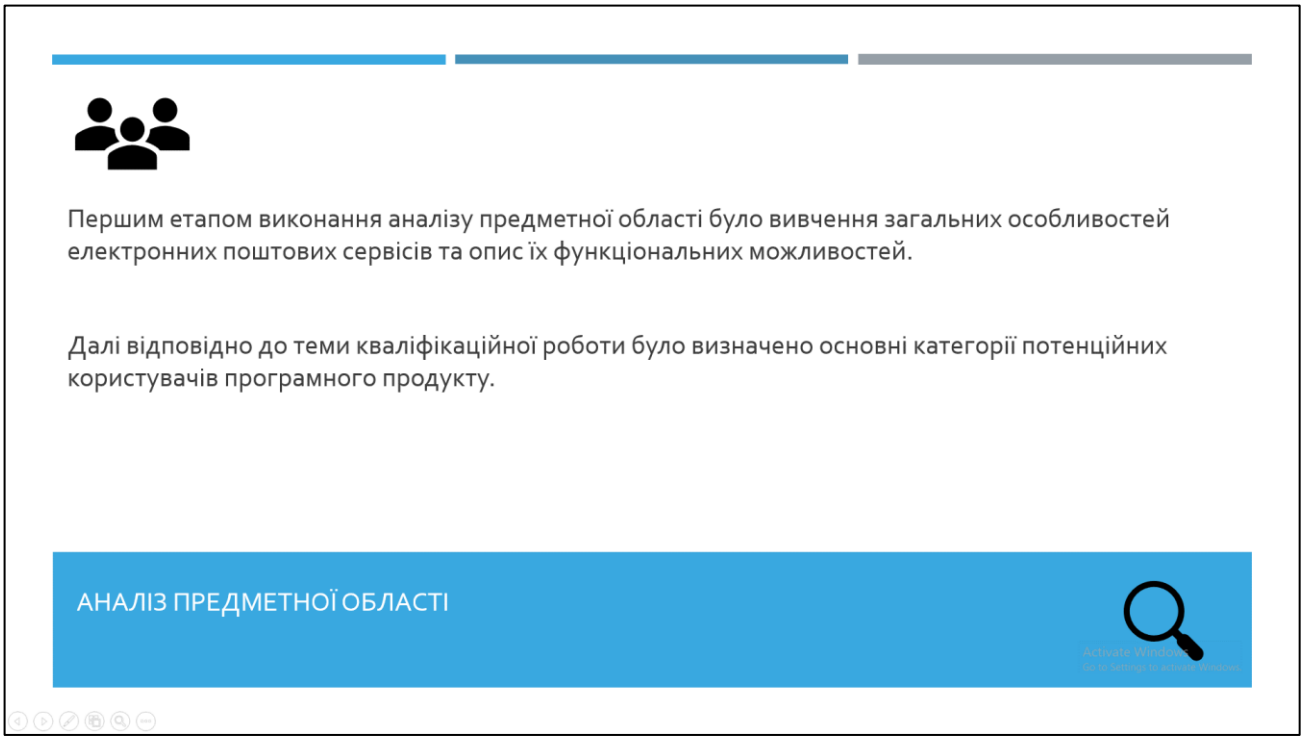
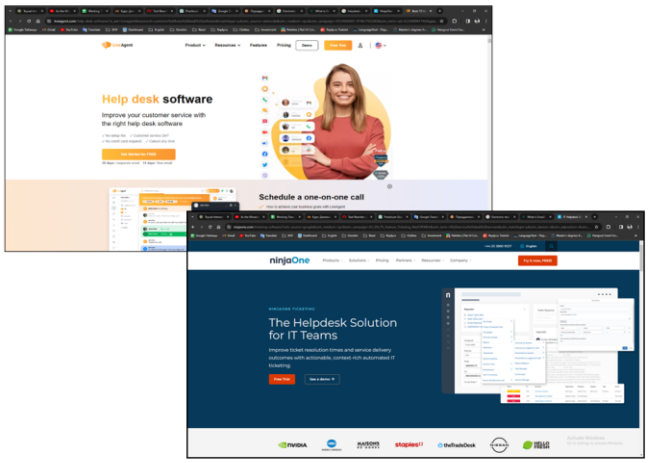


Рисунок Б.4 – Слайд №4

Для кращого розуміння вигляду фінального продукту на етапі проектування було розглянуто вже існуючі рішення.

Це допомогло зрозуміти їх переваги та недоліки, що допомогло встановити кращі вимоги та уникнути зайвого функціоналу

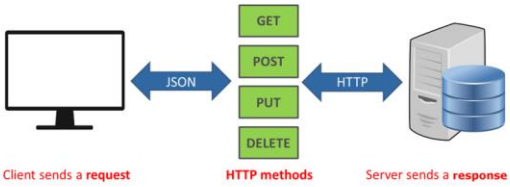


АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Рисунок Б.5 – Слайд №5

- Для реалізації проєкту було використано декілька архітектурних підходів.

Code-First Approach



ВИКОРИСТАНІ АРХІТЕКТУРНІ ПІДХОДИ

Рисунок Б.6 – Слайд №6

Рисунок Б.7 – Слайд №7

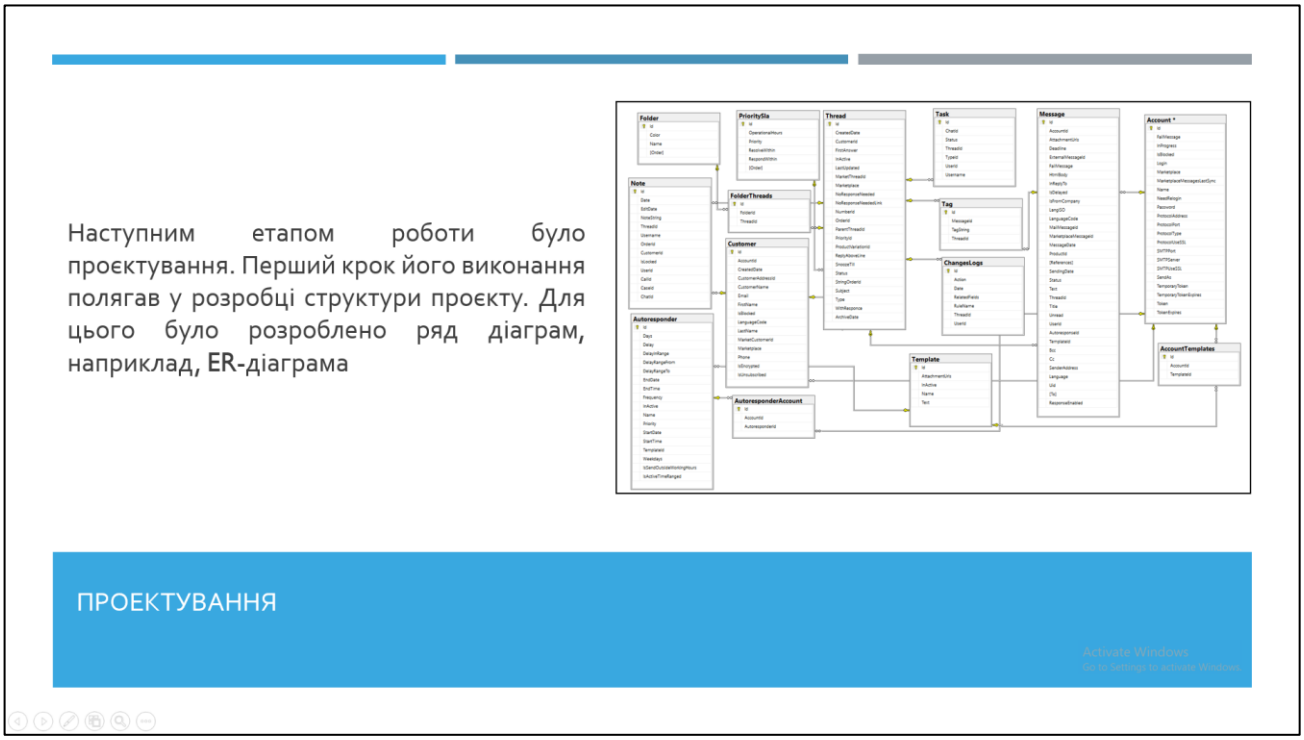


Рисунок Б.8 – Слайд №8

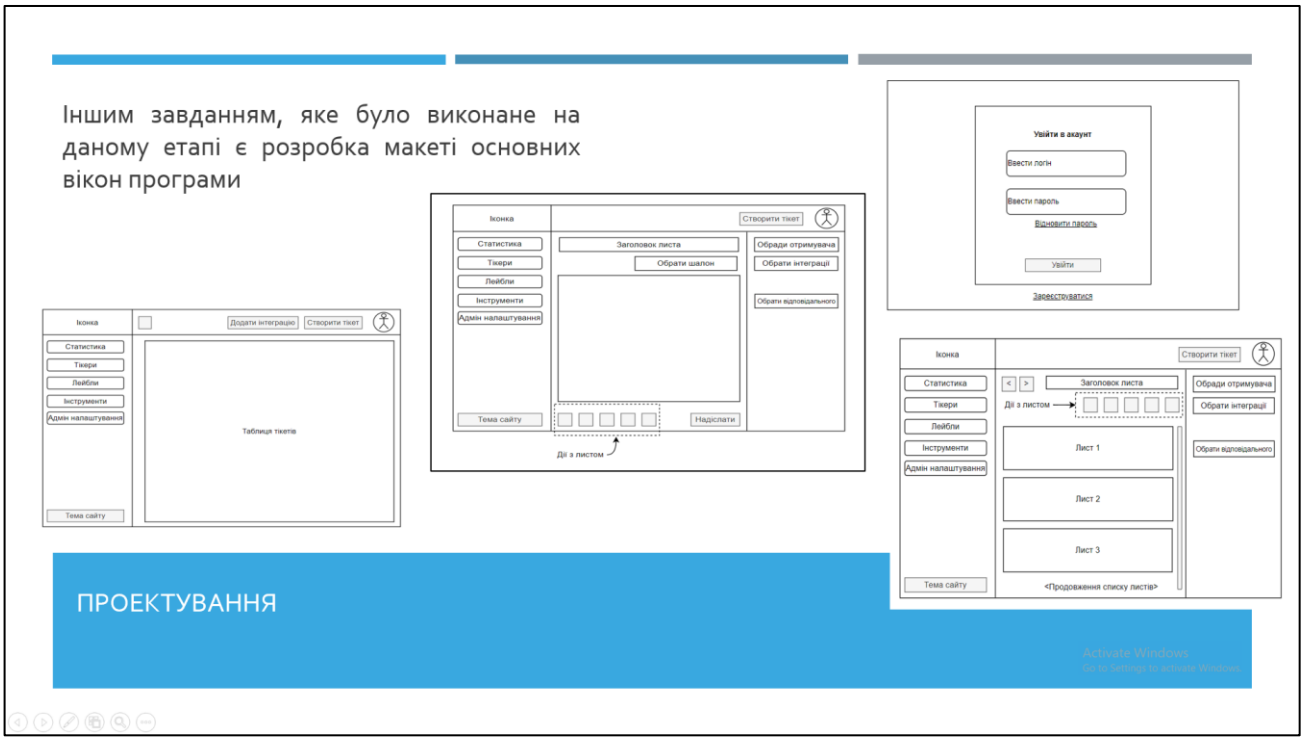


Рисунок Б.9 – Слайд №9

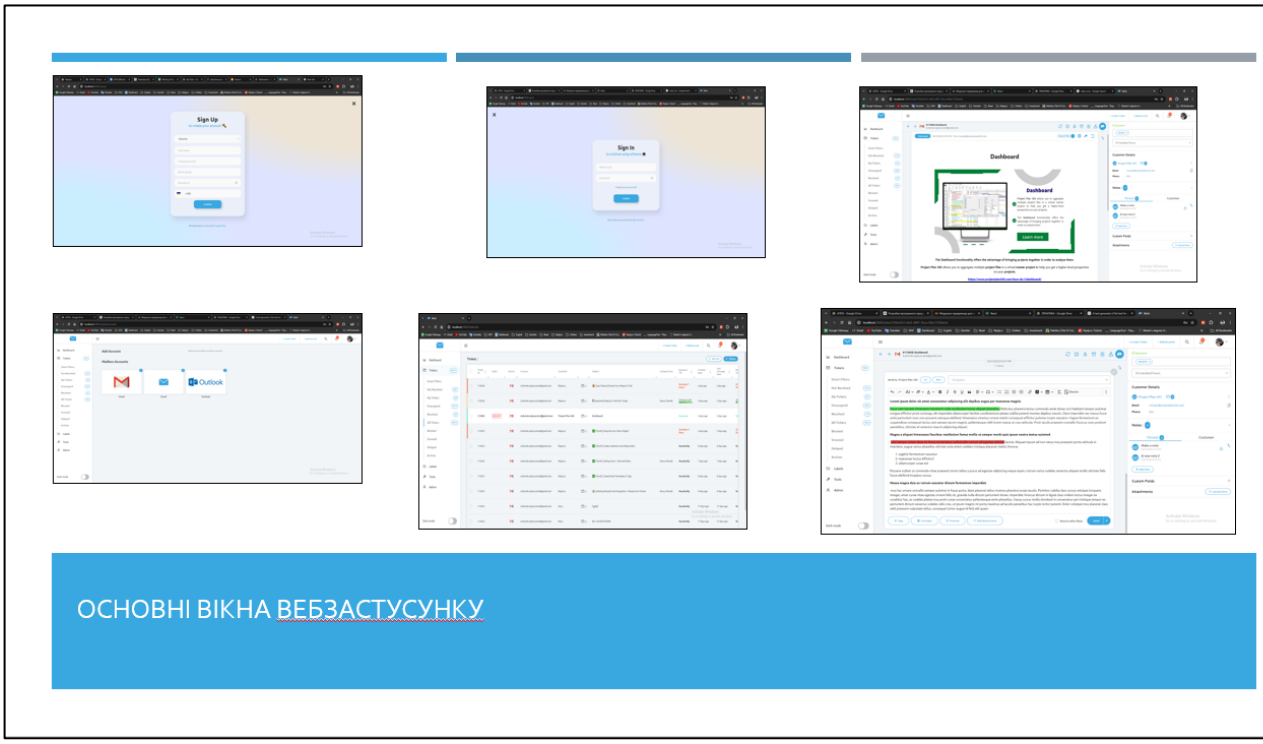
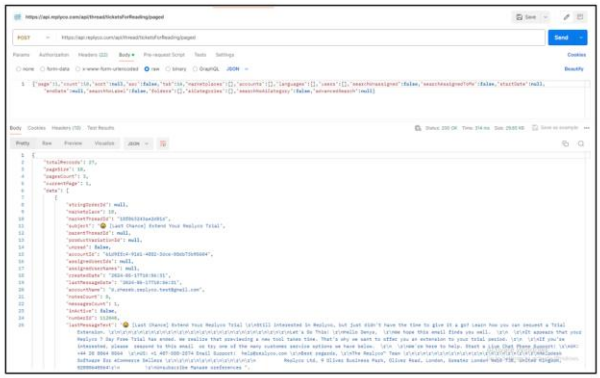


Рисунок Б.10 – Слайд №10

Завершальним етапом розробки будь-якого ПЗ є проведення тестування.

Для тестування даного ПЗ було вирішено провести тестування за допомогою **Postman** та декілька видів ручного тестування ПЗ



ТЕСТУВАННЯ

Рисунок Б.11 – Слайд №11

Для виконання даної роботи було встановлено основні завдання, потенційних користувачів системи, проаналізовано вже існуючі рішення на основі яких було здійснено проектування програмного забезпечення та у подальшому його реалізацію.

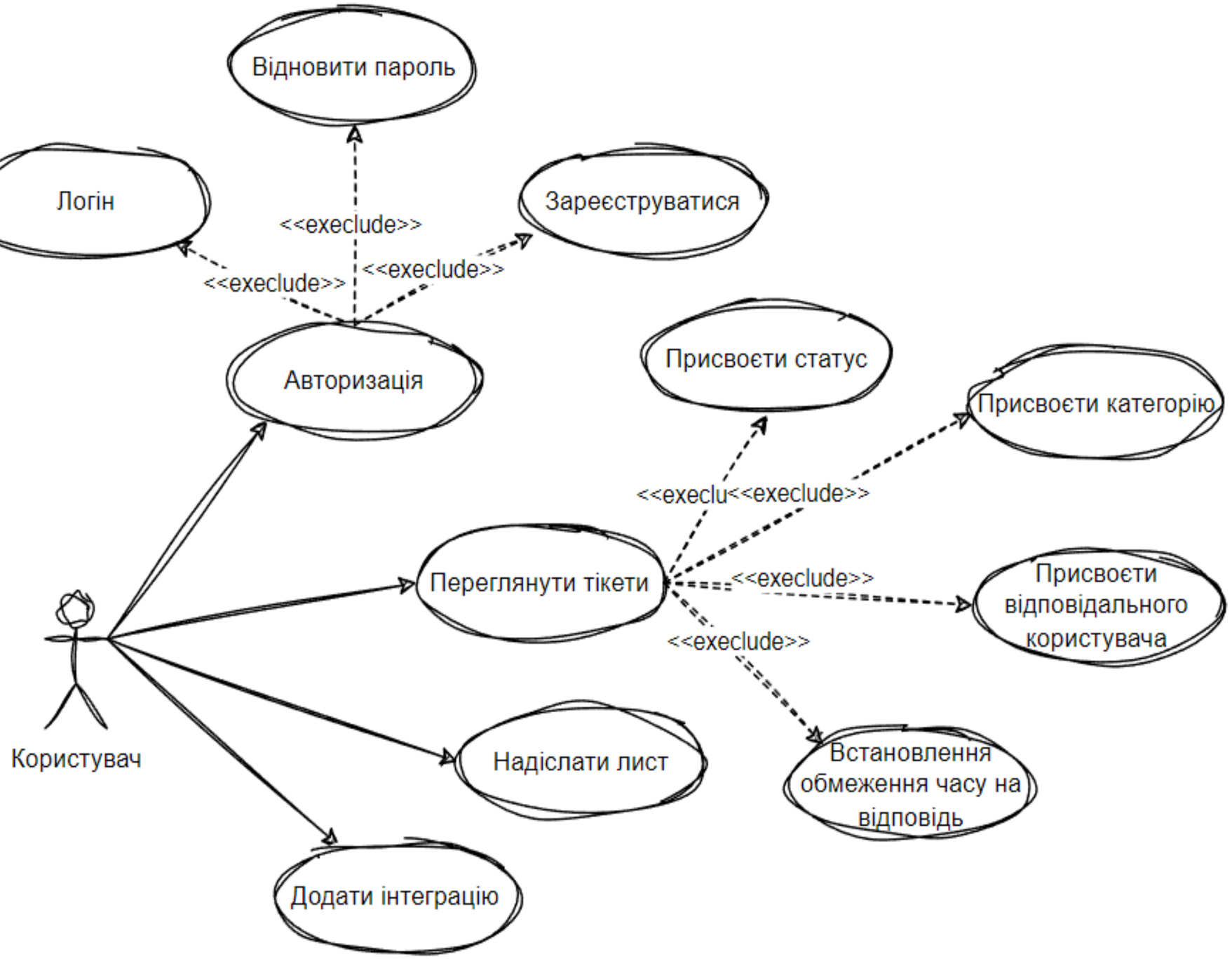
Проект відповідає поставленим цілям. Під час тестування критичних помилок у роботі програми не виявлено. У майбутньому програму можна модифікувати.

ВИСНОВКИ

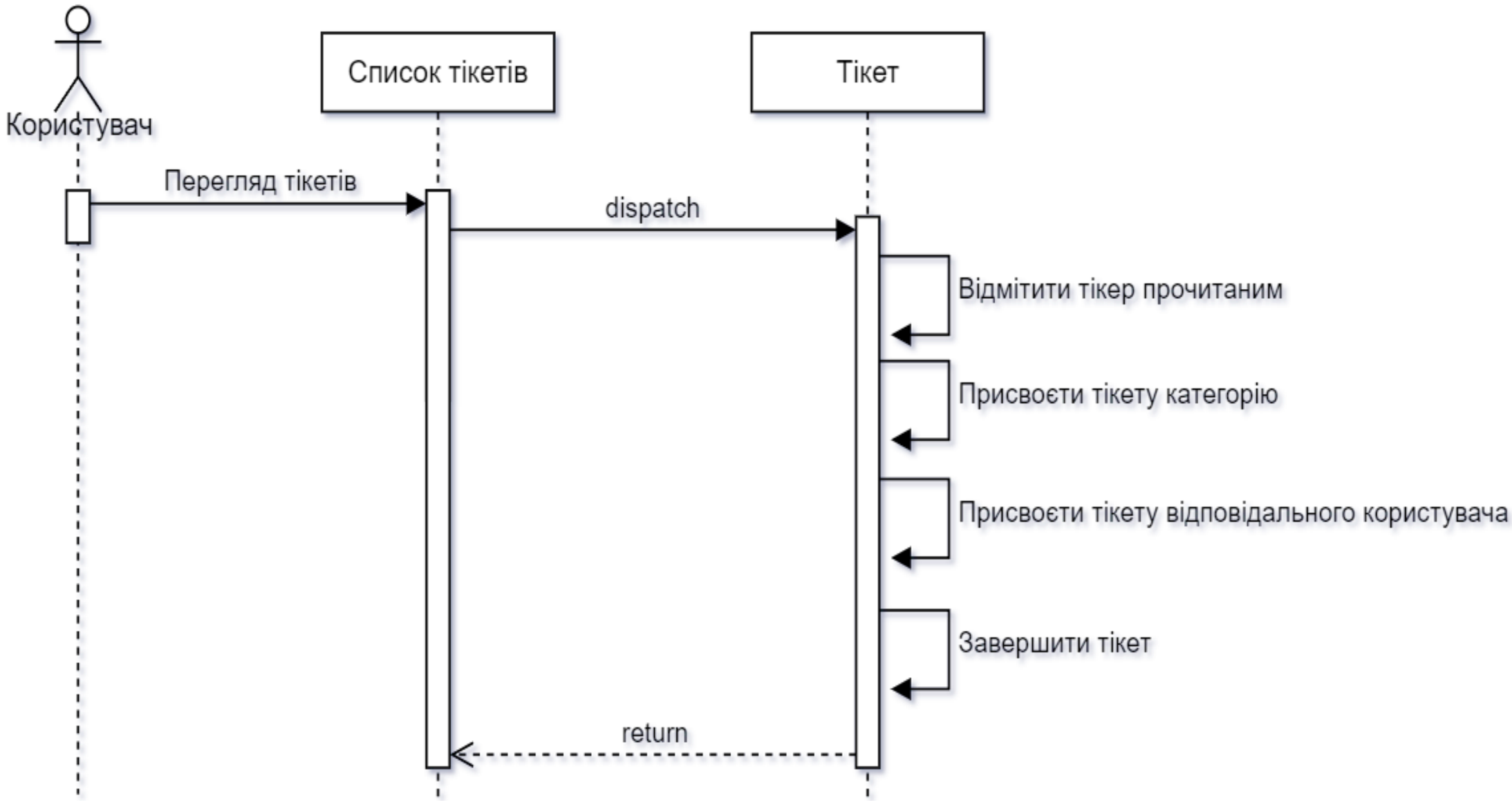
Рисунок Б.12 – Слайд №12

Рисунок Б.13 – Слайд №13

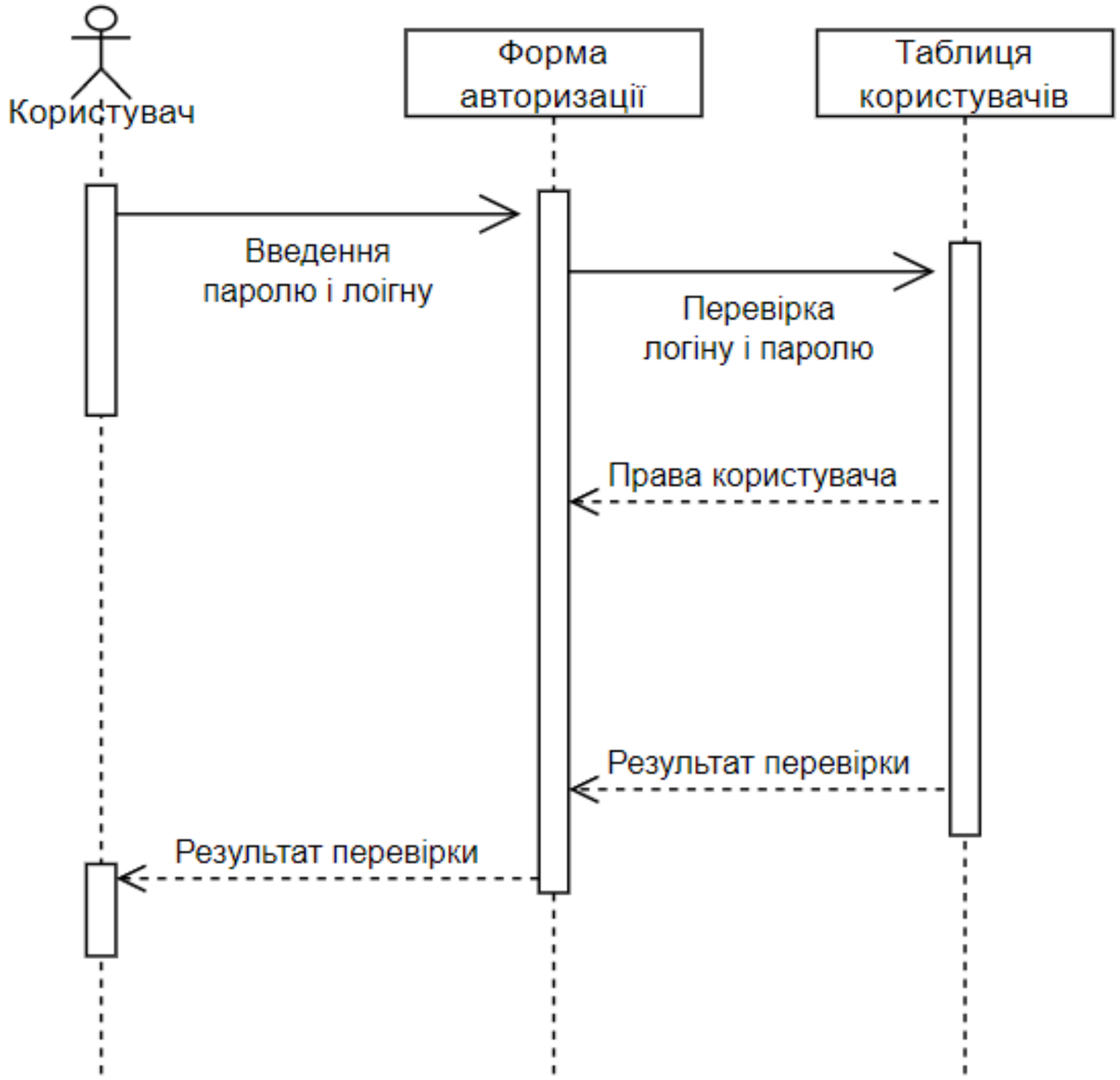
ГРАФІЧНА ЧАСТИНА



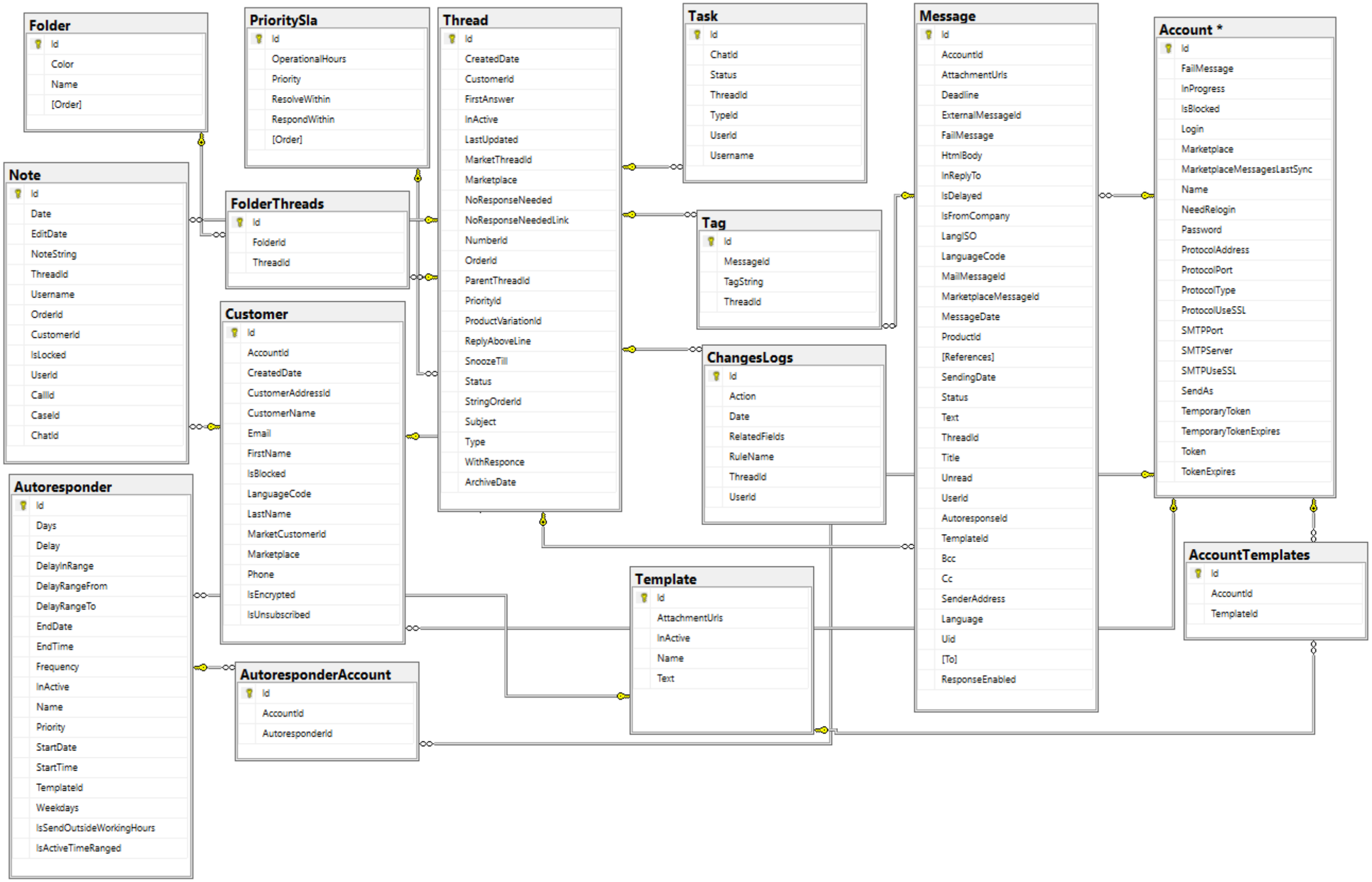
					КвРІПЗ. 2101099.01.02.Е8	
Зм	Арк.	№ докум.	Підпис	Дат		
Виконав		Жереб Д. В.			Діаграма варіантів використання	Літ. / Арк. 1 / Акрушів 4
Керівник		Бедратюк Л.П.				ХНУ, ІПЗс-21-1
.						
Н. контр.		Праворська Н. В.				
Зав. каф.		Бедратюк Л.П.				



Зм	Арк.	№ докум.	Підпис	Дат	КвРІПЗ. 2101099.01.02.Е8		
Виконав		Жереб Д. В.			Діаграма послідовності для варіанту використання перегляд тікетів	Літ.	Арк. 2 / Акрушів 4
Керівник		Бедратюк Л.П.					
						ХНУ, ІПЗс-21-1	
Н. контр.		Праворська Н. В.					
Зав. каф.		Бедратюк Л.П.					



Зм	Арк.	№ докум.	Підпис	Дат	КвРІПЗ. 2101099.01.02.Е8		
Виконав		Жереб Д. В.			Діаграма послідовності для варіанту використання авторизація	Літ.	Арк. 3 / Акрушів 4
Керівник		Бедратюк Л.П.					
Н. контр.		Праворська Н. В.				ХНУ, ІПЗс-21-1	
Зав. каф.		Бедратюк Л.П.					



					КвРІПЗ. 2101099.01.02.Е8	
Зм	Арк.	№ докум.	Підпис	Дат		
Виконав		Жереб Д. В.			Схема даних	Літ. / Арк. 4 / Акрушів 4
Керівник		Бедратюк Л.П.				
.						ХНУ, ІПЗс-21-1
Н. контр.		Праворська Н. В.				
Зав. каф.		Бедратюк Л.П.				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Жереб Д. В.
Прізвище, ініціали

факультет ІТ, 3 курс, група ІПЗс-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

06.01.2024
дата

підпис

Mon Jun 10 07:15:00 EEST 2024, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 17%**

ID: 129206 Назва: БКР_Вебзастосунок для централізації електронних листів з різних поштових сервісів Додано в БД: 2024-06-10 Автора: Жереб Д. Керівники: Бедратюк Л.П., дкоктор фіз.мат. наук, професор Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	110855	993	3653 (3%)	43 (4%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1016334307

Дата перевірки:
08.06.2024 09:32:14 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
08.06.2024 09:42:37 EEST

ID користувача:
100012953

Назва документа: **БКР_Вебзастосунок для централізації електронних листів з різних поштових сервісів_Жере...**

Кількість сторінок: **83** Кількість слів: **18838** Кількість символів: **164998** Розмір файлу: **3.12 MB** ID файлу: **1016134767**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7.11%
Схожість

Найбільша схожість: **0.89%** з джерелом з Бібліотеки (ID файлу: **1015040217**)

6.42% Джерела з Інтернету	877	Сторінка 85
2.65% Джерела з Бібліотеки	101	Сторінка 92

0.65% Цитат

Цитати	14	Сторінка 93

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	4
Підозріле форматування	33 сторінки

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Жереб Денис Валерійович

Тема Веб додаток для централізації електронних листів з різних електронних поштових сервісів

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 4 ; кількість сторінок записки 102

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі Жереба Дениса Валерійовича розглянуто створення веб додатку для централізації електронних листів з різних поштових сервісів. Проект включає аналіз предметної області, проектування архітектури системи, програмну реалізацію та тестування застосунку. Використано мову програмування JavaScript, технології React для фронтенду, ASP.NET Core API для бекенду та SQL Server для бази даниюх.

2. Висновок про відповідність роботи поставленому завданню Робота повністю відповідає поставленому завданню. Виконані всі необхідні етапи розробки: дослідження предметної області, проектування, реалізація та тестування веб застосунку.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи

Дослідження предметної області: Виконано детальний аналіз поштових сервісів, визначено вимоги до нового застосунку, проаналізовано наявне програмне забезпечення.

Проектування програмного забезпечення: Розроблено архітектуру системи, моделі бази даних, інтерфейс користувача. Використано UML діаграми для візуалізації структури системи.

Програмна реалізація: Реалізовано функціонал веб додатку з використанням JavaScript, React, ASP.NET Core API та SQL Server. Включено можливість централізації листів, управління тікетами та інтеграціями.

Тестування системи: Проведено детальне тестування функціоналу застосунку, описано результати тестування.

Використано сучасні методи розробки, такі як об'єктно-орієнтоване програмування та модульне проектування.

Інтеграція з різними поштовими сервісами через RESTful API забезпечує високу сумісність.

Реалізація безпечного доступу та авторизації користувачів через OAuth 2.0.

Використано передові технології для тестування та оптимізації продуктивності застосунку.

4. Позитивні сторони роботи

Використання сучасних технологій і передових методів розробки.

Високий рівень деталізації проектування та реалізації.

Інтеграція з різними поштовими сервісами забезпечує централізований доступ до електронних листів.

Комплексне тестування, яке забезпечує високу якість кінцевого продукту.

5. Негативні сторони роботи

Деякі аспекти роботи з оптимізацією продуктивності могли бути розглянуті більш детально, зокрема питання оптимізації швидкості завантаження даних з серверу та зменшення часу відгуку системи.

6. Оцінка графічного оформлення та пояснювальної записки

Графічне оформлення та пояснювальна записка виконані на високому рівні, містять усі необхідні схеми, діаграми та ілюстрації, що робить їх зрозумілими і зручними для користувачів.

7. Відгук про кваліфікаційну роботу в цілому

Кваліфікаційна робота Жереба Дениса Валерійовича є завершеним проектом, який демонструє високий рівень знань та навичок у галузі інженерії програмного забезпечення. Робота відповідає поставленим завданням та вимогам, містить теоретичні та практичні результати, які можуть бути використані в подальшій розробці веб додатків.

8. Інші зауваження

Рекомендується додати більше тестових сценаріїв для різних ситуацій використання, що дозволить більш повно оцінити стабільність та функціональність застосунку.

9. Оцінка кваліфікаційної роботи

Кваліфікаційна робота заслуговує на оцінку відмінно за змістом, якістю виконання та відповідністю поставленим завданням.

РЕЦЕНЗЕНТ Говорущенко Тетяна Олександрівна, доктор технічних наук, зав. кафедри комп'ютерної інженерії та інформаційних систем (КІІС) ХНУ

"06" 06 2024 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів:

Назва кваліфікаційної роботи: «Вебзастосунок для централізації електронних листів з різних поштових сервісів»
Автор: Жереб Денис Валерійович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системою перевірки на плагіат Unicheck виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів, у рамках основних написів, у назвах публікацій переліку джерел посилання;

2) в якості запозичень системою Unicheck було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) запозичення, виявлені в тексті роботи, є фрагментарними.

Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1.0%. Обсяг запозичень, визначений системою Unicheck виявлення збігів ідентичності/схожості, складає 7.11% і адресується до 877 джерел з Інтернету і 101 джерела з бібліотеки, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 08.06.2024 р.

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Леонід БЕДРАТЮК

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Жереб Денис Валерійович,

студент ІІІ курсу спеціальності 121 – Інженерія програмного забезпечення, група ІПЗс-21-1

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився (-лась) з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

06 03 2024 р.

Підпис

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗс-21-1
Жереба Д. В.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Вебзастосунок для централізації електронних листів з різних поштових сервісів
(керівник роботи – Бедратюк Леонід Петрович)
Прізвище, ім'я, по батькові

02.01.2024
Дата


Підпис студента