Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Система захисту внутрішньої конфіденційної інформації кросплатформової системи компанії Hexide Digital
Назва теми

КРКБ. 180128.18.01.04 ПЗ
Шифр

Галузь знань 12 – Інформаційні технології
Шифр, назва

Спеціальність 125 – Кібербезпека
Шифр, назва

Освітня програма Кібербезпека
Шифр, назва

Виконав студент 4 курсу, група КБ-18-01 6.06.2022 В. О. Качинський
Підпис, дата Ініціали, прізвище

Керівник 6.06.22 В. М. Чешун
Підпис, дата Ініціали, прізвище

Нормоконтролер 06.06.22 С. В. Мостовий
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри кібербезпеки Ю. П. Кльоц
Підпис, дата Ініціали, прізвище

6 06 2022 р.

Хмельницький, 2022

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
1			Завдання на кваліфікаційну	1		
			роботу			
2			Анотація	2		
3		КРКБ.180128.18.01.04 ПЗ	Система захисту	57		
			внутрішньої			
			конфіденційної інформації			
			кросплатформової системи			
			компанії Hexide Digital			
			Пояснювальна записка			
4	A2	КРКБ.180128.18.01.04 Е8	Процес передачі даних	1		
			допомогою бібліотеки			
			Pusher			
			Схема структурна			
5	A2	КРКБ.180128.18.01.04 Е8	Контекстна діаграма	1		
			процесу розробки			
			додатку			
			Схема структурна			
6	A2	КРКБ.180128.18.01.04 Е8	Аналіз внутрішніх загроз	1		
			Схема структурна			
7	A2	КРКБ.180128.18.01.04 Е8	Принцип авторизації	1		
			за допомогою JWT токена			
			Схема структурна			

Зм.	Аркуш	№ докум.	Підпис	Дата	КРКБ.180128.18.01.04 ПЗ		
Розробив		Качинський В.О.		03.0	Система захисту внутрішньої конфіденційної інформації кросплатформової системи компанії Hexide Digital Відомість проекту	Літ	Аркуш 1 / Аркушів 1
Перевірив		Чешун В. М.		6.06		У	
Н.контр.		Мостовий С. В.		06.06		ХНУ КБ-18-1	
Затвер.		Кльоц Ю. П.		6.06			

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КІБЕРБЕЗПЕКИ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 125 КІБЕРБЕЗПЕКА

Освітня програма ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА ПІДГОТОВКИ БАКАЛАВРІВ

ЗАТВЕРДЖУЮ

Зав. кафедри Ю.П. Кльоц

1 02 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Качинському Владиславу Олеговичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Система захисту внутрішньої конфіденційної інформації кросплатформової системи компанії Hexide Digital

Керівник роботи к.т.н., доц. Чешун Віктор Миколайович

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджено наказом ректора університету від 1 03 2022 р. № 18

2. Строк подання студентом проекту (роботи) на кафедру: 6.06.2022

3. Вихідні дані до проекту (роботи) сучасні способи обміну та зберігання інформації, види програмної реалізації сервісів обміну повідомленнями, способи проектування кросплатформенних додатків.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Аналіз об'єкту захисту, обгрунтування вибору засобів для побудови системи, проектування системи безпеки, реалізація роботи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) «Процес передачі даних за допомогою бібліотеки Pusher», «Аналіз внутрішніх загроз», «Контекстна діаграма процесу розробки веб-сервісу», «Принцип авторизації за допомогою JWT токена».

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Мостовий С. В., ст. викладач		06.06.22
Антиплагіат	Мостовий С. В., ст. викладач		06.06.22

7. Дата видачі завдання «30» січня 2022р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) Кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Вибір та затвердження теми кваліфікаційної роботи	Січень	–
2	Отримання завдання на кваліфікаційну роботу	Січень	–
3	Аналіз об'єкта захисту	Січень-лютий	–
4	Проектування та розробка загальної структури захищеного веб-сервісу, розгляд можливих варіантів	Лютий-березень	–
5	Програмна реалізація запропонованого рішення та тестування сервісу; аналіз результатів і оцінювання прийнятих рішень	Березень-квітень	–
6	Написання тексту пояснювальної записки та розробка графічних матеріалів	Травень	–
7	Остаточне коригування кваліфікаційної роботи з урахуванням зауважень керівника		–
8	Оформлення кваліфікаційної роботи як документа відповідно до вимог		–
9	Отримання супровідних документів. Нормконтроль	Червень	–
10	Підготовка до захисту та захист кваліфікаційної роботи		–

Студент _____ Качинський В.О.
Підпис Ініціали, прізвище

Керівник роботи _____ Чешун В. М.
Підпис Ініціали, прізвище

АННОТАЦІЯ

Тема кваліфікаційної роботи: Система захисту внутрішньої конфіденційної інформації кросплатформової системи компанії Hexide Digital

Автор роботи: Качинський Владислав Олегович

Керівник роботи: Чешун Віктор Миколайович

Пояснювальна записка: 57 с., 16 рис., 2 дод., 17 джерел.

Графічна частина: 4 плакати.

СИСТЕМА ЗАХИСТУ ІНФОРМАЦІЇ, КРОСПЛАТФОРМЕННИЙ СЕРВІС, ОБМІН ІНФОРМАЦІЄЮ, КІБЕРБЕЗПЕКА, КОНТЕКСТНА ДІАГРАМА ПРОЦЕСУ РОЗРОБКИ.

Метою кваліфікаційної роботи є проектування та програмна реалізація кросплатформенного додатку для обміну та зберігання внутрішньої конфіденційної інформації, визначення критеріїв оцінки безпеки додатку, визначення способів проектування кросплатформенних додатків, визначення методів захисту сучасних кросплатформенних додатків та визначення способів проектування стійких до загроз несанкціонованого доступу до інформації кросплатформенних додатків.

У цій роботі проаналізовані можливі сучасні способи проектування кросплатформенних додатків для обміну та зберігання інформації в режимі реального часу. Також визначено методи захисту таких додатків.

В результаті виконання кваліфікаційної роботи реалізовано захищений кросплатформенний додаток для обміну та зберігання внутрішньої конфіденційної інформації.

підпис студента

6.06.2022

Дата

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

JS – JavaScript

HTTPS – Hypertext Transfer Protocol Secure

SSL – Secure Sockets Layer

JWT – JSON Web Token

URL – Uniform Resource Locator

CI – Continuous Integration

CD – Continuous Delivery

Frontend – Client Part

Backend – Server Part

ВСТУП

Протягом усього часу існування цивілізації йде процес – розвитку. З кожним роком люди прагнуть дізнатись щось нове та використати це для покращення умов власного життя. В певний момент часу виникло таке поняття як обмін інформацією. Це дозволило людям існувати разом та досягати спільних результатів для підвищення рівня життя.

Найперший обмін інформацією здійснювався в усній формі у вигляді спілкування між людьми. Але що робити у ситуації, коли неможливо встановити прямий контакт? Внаслідок цього з'явились перші спроби обміну інформацією у писемному вигляді. Суспільство почало використовувати написи на стінах, а згодом вже цілком знайомі нам на теперішній час – паперові листи. Проте такий спосіб обміну інформацією повністю виключав можливість залишати її конфіденційною.

Для того щоб вирішити це питання люди почали використовувати різні варіації шифрів у листах. Найвідомішим є шифр Цезаря. Його суть полягає в тому, що кожна літера у слові зміщується на певну кількість позицій відносно нумерації алфавіту. Цей спосіб був актуальний у стародавній час, але люди знайшли розв'язок цієї задачі так як він був занадто легкий. Це потребувало створення нових рішень для захисту інформації.

На сьогоднішній день людство перейшло на найвищу сходинку розвитку під назвою – діджиталізація. Станом на сьогодні майже 70% всієї інформації зберігається та передається за допомогою електронних засобів. Весь перелік цих засобів дозволяє швидко та комфортно передавати інформацію з одного куточка світу до іншого за лічені секунди без залучення третіх лиць. Проте чи є це дійсно так?

Якість захисту інформації напряму залежить від професіоналізму та освіченості розробників програмного забезпечення, але це потребує великої кількості часу, а як ми знаємо у сучасному світі час – це гроші. Багато компаній з розробки засобів передачі інформації економлять гроші на захисті власного продукту. Наскільки тоді є ці продукти безпечними для використання у повсякденному житті, а тим більше як інструмент для ведення бізнесу? Саме це питання я хочу дослідити у своєму дипломному проекті та створити програмне забезпечення для обміну інформацією на основі захищеного каналу зв'язку.

Метою моєї роботи є створення системи захисту внутрішньої конфіденційної інформації кросплатформової системи. Цю мету потрібно досягти виконавши всі завдання кваліфікаційної роботи.

Такими завданнями, які повинні бути виконані у дипломній роботі є:

- об'єднати та закріпити свої теоретичні та практичні навички, які були отримані під час навчального процесу;
- проаналізувати існуючі сучасні сервіси обміну інформацією, виявити вразливості таких сервісів;
- визначити особливості предметної області, виявити проблемні місця;
- проєктування стійкого до несанкціонованого доступу до інформації кросплатформенного додатку.

1 АНАЛІЗ ОБ’ЄКТА ЗАХИСТУ

1.1 Характеристика предметної області

Для того щоб створити систему захисту внутрішньої конфіденційної інформації кросплатформової системи потрібно провести аналіз предметної області. Предметною областю у моїй роботі є підприємство, що надає послуги у сфері створення програмного забезпечення. Підприємство - це самостійний об’єкт господарювання, що був створений для виробництва товарів, виконання робіт і надання послуг з метою задоволення людських потреб та отримання прибутку. Підприємство надає людям робочі місця, сплачує заробітну плату. Шляхом оплати податків він бере участь у економічному розвитку нашої країни. Це означає, що в умовах ринкових відносин підприємством є соціально-виробничий організмом, що самоорганізується, та являє собою автономний центр виробничих, господарських і соціальних рішень. Підприємство має власну назву, фірмовий знак (марку), рахунок у банку. Воно також має майнову відповідальність за своїми зобов'язаннями, тобто, є юридичною особою. Часто у господарській практиці люди використовують поняття "фірма". Як правило, фірма є об'єднання однорідних або змішаних підприємств.

Кожне підприємство в умовах ринку прагне виробляти товари та послуги, що дають найбільший прибуток. Водночас, в умовах ринку немає гарантії, що підприємство її отримає. Все це залежить від багатьох обставин: правильного визначення незадоволених бажань покупців та орієнтації підприємства на їх виробництво, рівня витрат виробництва, які мають бути меншими, ніж доходи, отримані від продажу своєї продукції.

Остання залежить від продуктивності системи, НТП, рівня організації виробництва та праці, ступеня конкуренції тощо. Це вимагає від підприємства пошуку свого власного шляху розвитку, системи організації, маркетингу, своїх форм господарювання.

Внутрішньою конфіденційною інформацією підприємства виступають – дані про клієнтів, співробітників, перелік проектів, що знаходяться у розробці. Важливість систематизації цієї інформації відіграє важливу роль у функціонувані фірми. Для цього підприємство використовує стороні CRM-системи та чати.

CRM розшифровується як Customer Relationship Management, що буквально перекладається як «управління взаєминами з клієнтами». Це певна стратегія розвитку бізнесу, який практикує клієнтоорієнтований підхід. Очевидно, що перед її втіленням потрібно точно знати, хто саме є клієнтом компанії, що його цікавить, на що він розраховує та звідки приходить. Для цього і існують CRM-системи – комплекс технологічних та організаційних рішень, за допомогою яких компанія збирає, обробляє та зберігає всі дані про клієнтів. Ці системи дають можливість простежити історію взаємовідносин клієнта та компанії, проаналізувати побажання клієнтів, створити узагальнений портрет споживача. І все це заради того, щоб не лише залучити клієнта, запропонувавши саме те, що потрібно, але й утримати його.

Ринок CRM-послуг в Україні не такий великий, як за кордоном – за даними на початок 2022 року, в нашій країні працює близько десяти компаній, які надають CRM-системи для бізнесу. До цього числа входять як розробники ПЗ, а й ті, хто спеціалізується на його імплементуванні[1].

Є багато сучасних методів для того, щоб розробити інформаційну систему, але основний з них побудований на виділених документарних елементах у бізнес-процесах. Саме там атрибутами є об’єкти предметної області, а потім сутності системи.

Такий варіант підходу використовується на рівні розгляду бізнес-процесів та проведення операцій з спрощення бізнес-процесів. Але, варто зазначити, що під час переходу на рівень моделювання, є необхідність використовувати інший підхід. Будь-який з підходів спочатку орієнтується

на аналіз предметної області. Він використовується в інтереса, що до виявлення різних ключових документарних об'єктів в предметній області.

Чат – це певний вид програмного забезпечення, що дозволяє користувачам обмінюватись повідомленням використовуючи мережу інтернет. Здебільшого їх використовують для особистого спілкування, але з плином часом та розвитком технологій, компанії почали також їх активно використовувати для ведення бізнесу. Тому поєднання CRM-систему з чатом може дійсно підвищити рівень комфортної та безпечної комунікації всередині підприємства, що позитивно вплине на час реалізації проектів та їх якості.

Провівши аналіз, я виявив певні недоліки цих методів обміну інформацією. Це програмне забезпечення не є зовсім безпечним та еталоним для обміну конфіденційною інформацією підприємства тому, що:

- невідома кінцева точка зберігання даних;
- види та захист методів зберігання інформації;
- можливість доступу до інформації сторонніх лиць.

1.2 Визначення методів передачі даних сучасних кросплатформенних сервісів зберігання інформації

На даний час існує багато варіантів, що дозволяють вести бізнес використовуючи безліч готових рішень, але чи є вони надійними в ролі системи захисту внутрішньої конфіденційної інформації?

Одним з небагатьох готових рішень що включають в себе одночасно функціонал CRM-системи та чату, є система – Бітрікс24.

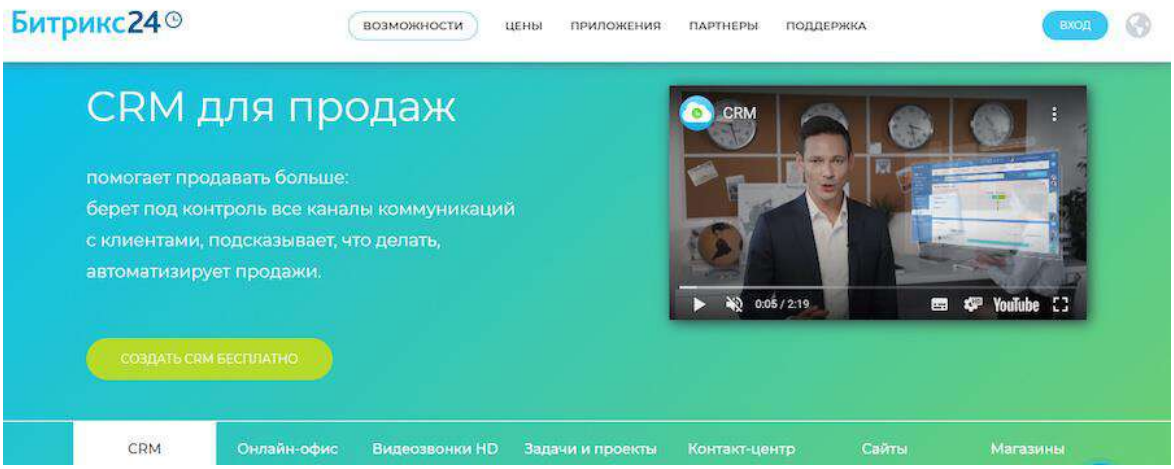


Рисунок 1.1. Прев'ю системи Бітрікс24

Бітрікс24 – це надзвичайно велика система інструментів, що користується великим попитом у багатьох підприємств. Вона дозволяє ефективно вести бізнес створюючи групи людей та призначати задачі для виконавців[1].

Велику популярність цьому сервісу принесло якраз те, що система включає в себе можливість не тільки зберігання внутрішньої інформації підприємства, а й можливість внутрішнього спілкування між співробітниками використовуючи чати. Це економить великий відсоток часу при веденні задач, адже вам не потрібно використовувати інше програмне забезпечення для спілкування з колегами чи вашим керівництвом. Ви можете зробити це одразу всередині сервісу. А для більшого комфорту розробники також додали можливість інтегрування вашої поштової скриньки до сервісу. Всі листи з вашого емейла автоматично будуть перенаправлені в віртуальний кошик повідомлень всередині платформи.

На перший погляд здається, що це чудове рішення використовувати цей сервіс для власних потреб – але це насправді виявилось не зовсім так. Питання якості захисту вашої внутрішньої інформації напряму залежить від вашої платоспроможності.

Безкоштовна версія сайту як і більшість інтернет ресурсів обмінюється даними з сервером використовуючи принцип – REST API.

REST API (також відомий як RESTful API) – це інтерфейс програмування прикладних програм (API або веб-API), що відповідає обмеженням архітектурного стилю REST і дозволяє взаємодіяти з веб-сервісами RESTful. REST розшифровується як репрезентаційна передача стану і був створений комп'ютерним науковцем Роєм Філдінгом.

REST – це набір архітектурних обмежень, а не протокол чи стандарт. Розробники API можуть реалізувати REST різними способами.

Коли запит клієнта здійснюється через API RESTful, він передає уявлення про стан ресурсу запитувачу або кінцевій точці. Ця інформація або подання надається в одному з кількох форматів через HTTP: JSON (нотація об'єктів Javascript), HTML, XLT, Python, PHP або звичайний текст. JSON є найбільш популярним форматом файлів, оскільки, незважаючи на свою назву, він не залежить від мови, а також його можна читати як людьми, так і машинами.

Слід мати на увазі ще дещо: заголовки та параметри також важливі в методах HTTP HTTP-запиту RESTful API, оскільки вони містять важливу інформацію про ідентифікатор щодо метаданих запиту, авторизації, єдиного ідентифікатора ресурсу (URI), кешу, файлів cookie та більше. Існують заголовки запиту та відповіді, кожен із власною інформацією з'єднання HTTP та кодами стану.

Згідно з Comnews, на травень 2022 року, переважна більшість систем мають критичні вразливості, що дозволяють зловмисникам отримати несанкціонований доступ до конфіденційної інформації компаній, користувачів та клієнтів та здійснювати різні дії під виглядом автентифікованого користувача[16]. Виходячи з вищесказаного, всім співробітникам, що беруть участь у перевірках інформаційних систем,

потрібно знати про існуючі бреші в організації доступу до API різних сервісів та систем, а також знати, як з ними можна і потрібно боротися.

За даними спільноти OWASP (відкритий проект забезпечення безпеки веб-додатків), існує кілька особливо небезпечних вразливостей, які можуть бути допущені при проектуванні та розробці REST API. Однією з таких вразливостей є порушення автентифікації користувачів (Broken User Authentication).

Broken User Authentication – вразливість, при якій зловмисник, не проходячи процедури автентифікації, або «обійшовши» її, може отримати доступ до інформації, що передається через API[4].

Отже, використання безкоштовної версії сервісу зовсім не є безпечним. Проте, якщо ваші потребності не включають в себе надійний захист інформації, яка може бути використана зловмисниками проти вас чи може завдати вам збитків - то використаня безкоштовної версії вам цілком підходить.

1.3 Аналіз відомих методів забезпечення захисту інформації

При розробці системи захисту внутрішньої конфіденційної інформації кросплатформової системи важливу роль відіграє кібербезпека та використання методів забезпечення надійного захисту інформації. Одним із таких є метод аутентифікації.

Аутентифікація – це процес розпізнавання особистості користувача. Це механізм зв'язування вхідного запиту з набором ідентифікаційних облікових даних. Надані облікові дані порівнюються з обліковими даними у файлі в базі даних інформації авторизованого користувача в локальній операційній системі або на сервері аутентифікації[2].

Процес аутентифікації завжди виконується на початку програми, до того, як відбудуться перевірки дозволів і регулювання, і до того, як буде дозволено продовжити будь-який інший код. Різні системи можуть вимагати різних типів облікових даних для встановлення особи користувача. Облікові дані часто набувають форми пароля, який є таємним і відомий лише особі та системі. Три категорії, у яких хтось може бути автентифікований: те, що користувач знає, те, що він є, і те, що є у користувача.

Процес аутентифікації можна описати у дві окремі фази – ідентифікацію та фактичну аутентифікацію. Фаза ідентифікації забезпечує ідентифікацію користувача системі безпеки. Ця особистість надається у вигляді ідентифікатора користувача. Система безпеки шукатиме всі відомі їй абстрактні об’єкти та знайде конкретний з них, який фактичний користувач зараз використовує. Після цього користувача ідентифіковано. Той факт, що користувач стверджує, не обов’язково означає, що це правда. Фактичний користувач може бути зіставлений з іншим абстрактним об’єктом користувача в системі, і, отже, отримати права та дозволи користувачеві, і користувач повинен надати докази, щоб підтвердити свою особистість системі.

Існують вже такі відомі методи аутентифікації:

- парольна аутентифікація – це найбільш відомий та популярний вид аутентифікації. Може бути організований у вигляді багаторазових та одноразових паролів. Багаторазові паролі зберігаються у базі даних та при кожному вході звіряються з вже наявними записами. Одноразові паролі генеруються при кожному вході та є валідними лише один раз;
- аутентифікація на основі особистих даних – також відомий метод. Здебільшого використовується при відновленні втрачених доступів до акаунта та генеруванні нових паролей. Наприклад, вас просять ввести останні цифри номеру телефону до якого прив’язаний акаунт, ім’я

домашньої тваринки та багато іншої інформації відомої лише власнику облікового запису;

- комбінований метод аутентифікації – метод який будується на комбінувані декількох методів. Один з найпопулярніших випадків комбінованої аутентифікації це парольний метод разом з підтвердженням за допомогою коду який надсилається на інший ресурс власника акаунту, здебільшого це SMS або Email підтвердження;
- біометрична аутентифікація – найточніший метод аутентифікації. Найпопулярнішими вже видами є скан сітківки ока, відбитку пальців та створення віртуального зліпку вашого обличчя.

В залежності від того, якою є кількість методів аутентифікації використовується, виділяють два види аутентифікації - це однофакторна та багатофакторна.

Терміни «методи» та «протоколи» мають два різні значення для аутентифікації. Протокол аутентифікації — це базова структура, яка викладає правила для перевірки. Наприклад, протокол аутентифікації пароля (PAP) вимагає від когось ввести ім’я користувача та пароль, щоб перевірити, хто вони.

У той же час метод аутентифікації знаходиться поверх протоколу. Підприємство може мати кілька методів аутентифікації, які допомагають перевірити інформацію користувача на відповідність протоколу. Наприклад, людина може ввести своє ім’я користувача та пароль, але її також можуть попросити підтвердити свою особу за допомогою коду, надісланого на її телефон, що є методом, відомим як двофакторна аутентифікація (2FA).

З одним або кількома методами аутентифікації протоколи аутентифікації можуть з надзвичайною впевненістю підтвердити, що користувач є тим, за кого вони є. Нижче наведено більш детальний розгляд різних типів методів і протоколів аутентифікації, а також порівняння найпопулярніших варіантів.

Протоколи аутентифікації викладають основні правила, які підтверджують, що користувач є тим, ким він себе є. Найменш безпечний з усіх протоколів відомий як Протокол аутентифікації пароля (PAP) і просто просить користувача ввести пароль, який збігається з паролем, збереженим у базі даних. PAP не використовує жодного шифрування, тому він вважається небезпечним і застарілим.

Протягом багатьох років було введено ряд інших протоколів аутентифікації, усі з метою покращення рівня кібербезпеки. Ось розглянемо найпоширеніші.

SAML – SAML розшифровується як Security Assertion Markup Language (SAML). Він розроблений для підтримки системи єдиного входу, дозволяючи користувачеві ввійти до постачальника ідентифікаційних даних, який перевіряє його особу щоразу, коли вони запитують доступ до програми або сайту через постачальника послуг, що бере участь. SAML був розроблений для спрощення доступу користувачів до кількох програм без необхідності входу в систему.

OAuth – OAuth означає «Відкрита автентифікація». Це дозволяє програмам надавати «захищений призначений доступ». Це один з найпопулярніших протоколів аутентифікації в Інтернеті. Наприклад, Facebook використовує його, щоб дозволити користувачам надавати веб-сайтам дозвіл переглядати та публікувати на своїй часовій шкалі, не повідомляючи пароль користувача Facebook.

OIDC – Побудований на основі OAuth 2.0, який є службою авторизації, цей протокол додає простий рівень ідентифікації поверх служби авторизації, що дозволяє клієнтам/постачальникам послуг перевіряти ідентичність кінцевого користувача.

LDAP – полегшений протокол доступу до каталогів (LDAP) був розроблений для швидкої аутентифікації. Інформація про користувача

зберігається в Active Directory (AD) і може бути витягнута лише у зручному для використання форматі за допомогою LDAP.

З усіма цими протоколами аутентифікації на ринку підприємствам може бути важко визначити, який із них найкраще підходить для їхнього використання. Однак, що стосується системи захисту внутрішньої конфіденційної інформації кросплатформової системи, SAML вважається галузевим стандартом. Далі порівняємо SAML з іншими протоколами аутентифікації.

SSO не єдиним методом аутентифікації, який можна використовувати з протоколом SAML, але вони стали синонімами один одному. Але з такою кількістю варіантів, чому SAML вважається найкращим?

Однією з найважливіших переваг використання SAML для цілей SSO є відкритий стандарт. Це означає, що постачальники та постачальники можуть легко взаємодіяти один з одним, якщо вони дотримуються стандарту SAML. Крім того, оскільки SAML використовує XML, він надзвичайно гнучкий. Ви можете передавати всі види даних, якщо XML може їх відтворити.

Зважаючи на ці деталі, деякі компанії все ще дискутують між SAML та OAuth. OAuth дещо новіший за SAML, але його розробили Google і Twitter, частково щоб компенсувати деякі недоліки SAML. З цієї причини OAuth використовує JSON замість XML.

Інша основна відмінність між SAML та OAuth полягає в тому, що OAuth створено як мережу авторизації, а не для аутентифікації. Рівень OpenID Connect розташований поверх OAuth для обробки аутентифікації, і він був випущений набагато пізніше. Іншою відмінністю є варіант використання: Google і Twitter розробили OAuth для використання в Інтернеті. SAML було розроблено з урахуванням відкритого Інтернету, але він ідеально підходить для закритих корпоративних мереж[3].

Якщо ви проглянете список і порівняєте SAML окремо з кожним іншим протоколом, згаданим тут, швидко стане зрозуміло, що SAML може

бути не найкращим для кожної програми, але він є безсумнівним переможцем, коли справа доходить до компаній, які прагнуть встановити кращу аутентифікацію для їх користувачів. Він також широко визнаний як найшвидше рішення для використання в бізнесі.

Пошук правильного протоколу та методів аутентифікації є основоположним для формування системи захисту внутрішньої конфіденційної інформації кросплатформової системи. Однак це далеко не єдина частина пазлу. Оскільки кібератаки з кожним днем стають все більш частими та складнішими, дуже важливо, щоб підприємства знайшли час відступити та подивитися на свої процедури кібербезпеки та керування даними в цілому.

Управління даними швидко зайняло позицію в розмові про кібербезпеку. Це стосується не тільки відповідності та конфіденційності даних, але й означає знати, якими даними ви володієте, як їх потрібно захищати, хто повинен мати до них доступ і що станеться, якщо вони отримають доступ. Тому будь-яка організація, яка думає про аутентифікацію, повинна також думати про управління даними.

Отже, вибір правильних методів аутентифікації відіграє одну з найважливіших ролей у системі захисту внутрішньої конфіденційної інформації кросплатформової системи. Це вирішить проблему загрози проникнення третіх лиць до системи зберігання даних.

2 ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБУ ДЛЯ ПОБУДОВИ СИСТЕМИ

2.1 Способи проектування кросплатформенних додатків

Для розробки системи захисту внутрішньої конфіденційної інформації кросплатформової системи я використовую гібридний підхід – це підхід, який дозволяє розробникам створити мобільне рішення, сумісне одночасно з кількома операційними системами та платформами (Android, iOS, Windows).

Гібридні програми мають оригінальний вигляд і відчуття завдяки поєднанню рідного коду з незалежним кодом, який підходить для кількох платформ. Розробники пишуть код один раз, а потім використовують його повторно, що дозволяє швидко випускати продукт[5].

Для кодування кросплатформного програмного забезпечення розробники використовують проміжні мови програмування – HTML, JavaScript і CSS – не рідні для пристроїв і ОС. Потім програми упаковуються в нативні контейнери та інтегруються в платформи.

Існує два підходи до створення додатків для створення мобільних рішень: нативний та гібридний/кросплатформний. Вони обидва мають переваги та недоліки і можуть підійти для вашого проекту залежно від ваших потреб і сценарію використання.

Нативна розробка покладається на інструменти та мови програмування, розроблені спеціально для однієї платформи. Вони оригінальні для пристрою та операційної системи. Наприклад, Objective-C і Swift використовуються для додатків iOS; Java , C/C++ і Kotlin допомагають створювати рішення для Android; C# і Visual Basic оптимальні для Windows Phone.

При створенні нативних додатків розробники керуються конкретними вимогами ОС. Встановлена на цільовому комп’ютерному пристрої, рідна програма може повністю використовувати доступні функції та можливості.

Недоліком є те, що рідна програма несумісна з іншими платформами. Щоб охопити ширшу аудиторію, вам потрібно буде розробити кілька рішень (один додаток для кожної цільової платформи), що забирає багато часу та витрат.

Міжплатформна розробка спрямована на створення єдиної програми, яка однаково працює на кількох платформах. Він використовує технології, що не залежать від платформи, такі як HTML і CSS, і допомагає підприємствам охоплювати багато кінцевих пристроїв за менших витрат.

Розробка багатоплатформних мобільних додатків постійно розвивається завдяки новітнім технологіям, стаючи все більш динамічними та привабливими для розробників.

Переваги:

- створення окремих нативних додатків для кожної платформи коштує дорого, тоді як гібридна програма використовує єдиний загальний код, що допомагає вам утримуватися в межах бюджету;

- витрати зменшуються, оскільки для розробки та підтримки програми потрібна лише одна команда програмістів. Більше того, достатньо базових знань стандартних мов – інструменти розробки зроблять решту роботи;

- міжплатформні програми мають оригінальний зовнішній вигляд та відчуття, що чудово підходить для користувача;

- гібридна розробка – це, безумовно, шлях для компаній, які хочуть залучити користувачів різних мобільних пристроїв і швидше випустити продукт на ринок за меншою ціною;

Однак є деякі проблеми, з якими ви можете зіткнутися.

Проблеми:

- складніший код гібридних рішень поєднує рідні та нерідні компоненти, що може вплинути на продуктивність;

- міжплатформні програми не можуть підтримувати всі функції та функції мобільних пристроїв, які є лише нативними, як-от розширена графіка

та анімація або 3D-ефекти. Це призводить до обмеженої функціональності та погіршення дизайну програми;

- коли Google і Apple додадуть нові функції до платформ Android і iOS, нативні рішення можуть негайно почати їх використовувати. Але гібридним додаткам доведеться почекати, поки ці оновлення не будуть адаптовані до вибраної міжплатформної платформи. Таким чином, завжди буде затримка оновлення;

Ці проблеми можуть бути критичними чи ні для вашого проекту, це вам вирішувати.

Кросплатформна розробка підходить для рішень, які:

- не вимагають розширеного дизайну;
- не потрібно обробляти вхідні дані онлайн;
- не потрібен доступ до всіх функцій пристрою.

2.2 Використання технологій під час розробки

Створення системи захисту внутрішньої конфіденційної інформації кросплатформової системи потребує тонкого розуміння всіх процесів розробки. Розробка такого типу проектів здебільшого поділяється на два головних етапи:

- створення Backend частини застосунку;
- створення Frontend частини застосунку.

Для чіткої та злагодженої роботи потрібно обрати найкращі рішення, що не будуть викликати різного типу помилок під час поєднання двох частин застосунку.

Проаналізувавши ринок технологій, що є у тренді в наші дні для створення Backend частини додатку я обрав мову PHP, а саме фреймворк під назвою – Laravel.

Laravel – є найпопулярнішим PHP фреймворком. Як і будь-який інший фреймворк PHP, Laravel безкоштовний у використанні та ліцензований під відкритим кодом. Його створив Тейлор Отвелл. Мета створення фреймворку полягає в тому, щоб полегшити розробку веб-фреймворків, які використовують архітектуру Model-View-Controller (MVC). Laravel PHP розглядається як одна з найпростіших для роботи фреймворків, і вона має всі сучасні функції, які ви повинні очікувати від сучасного фреймворку. Кілька ключових особливостей Laravel – це модульна система пакування, кілька способів доступу до реляційних баз даних, різні утиліти та багато іншого[6].

Приблизно 4 роки тому спільнота PHP була пусткою конкуруючих фреймворків. Конференції представляли собою масу творців фреймворків, які говорили про те, над чим вони працювали і що їх фреймворк найкраще вміє, вихваляючись, чия структура була найкращим рішенням проблеми.

Тейлор Отвелл, розробник .net з Арканзасу, використовував CodeIgniter, коли були висаджені перші зародки Laravel. «Я не міг додати всі функції, які я хотів, – каже він, – без зміни внутрішнього коду фреймворку». Він хотів чогось більш компактного, простішого та гнучкого.

Ці бажання в поєднанні з .net-минулим Тейлора породили структуру, яка стане Laravel.

Традиційно PHP був схожим на дикий захід мов програмування. Оскільки він динамічно вводиться, існує багато речей, які потенційно можуть піти не так, якщо не буде попередження. Наприклад, на інших мовах код записується таким чином, що якщо буде зроблена помилка, її можна перехопити перед виконанням коду. Або намагаючись використати код, або зібравши його, компілятор фактично зупинить вас, визначить проблему у коді, і вимагають виправити це, перш ніж рухатися далі.

З коробки PHP нічого цього не має. Таким чином, теоретично ви можете продовжувати писати поганий код, не знаючи, чи працює він, чи ні,

вічно. Коли щось йде не так, вам доведеться витратити багато часу на пошук проблеми.

Тейлор прагнув вирішити всі ці проблеми з Laravel. Він використав ідеї інфраструктури .net, яку побудувала Microsoft, і витратила сотні мільйонів доларів на дослідження.

За допомогою Laravel Тейлор намагався створити фреймворк, який був би відомий своєю простотою. Він додав до цієї простоти виразний синтаксис, чітку структуру та неймовірно ґрунтовну документацію. З цим народився Ларавел.

Але звісно Laravel це лише інструмент для роботи з даними, сама по собі інформація буде зберігатись у базі даних, для цього я обрав – MySQL.

MySQL – одна з систем управління базами даних, що найбільш використовуються. MySQL управляє реляційними базами даних, тобто такими, у яких таблиці пов'язані між собою.

MySQL працює за принципом клієнт-сервер . Комп'ютер користувача (клієнт) надсилає запит. Сервер баз даних його обробляє та надає відповідь. Ось тому часто можна почути поняття MySQL-сервер. Це сервер, де зберігається база даних.

Система MySQL написана мовами програмування C та C++. Для роботи MySQL використовується мова структурованих запитів SQL.

SQL (Structured Query Language) – це мова програмування, за допомогою якої можна керувати інформацією: додавати, модифікувати, видаляти та отримувати дані. Запити до бази даних формуються мовою SQL.

SQL використовується у MySQL. Багато РСУБД (реляційні системи управління базами даних) використовують цю мову для роботи з даними.

Наприклад:

- Microsoft SQL Server;
- PostgreSQL;
- Oracle Database;

- MariaDB;

- SQLite.

SQL використовується у запитах при зверненні до бази даних. Знання SQL дозволить вам працювати з будь-якою реляційною базою даних, яка використовує цю мову. В нашому випадку я буду використовувати MariaDB[14].

На цьому етапі опис технологій для створення Backend частини додатку – завершено.

Наступним пунктом є вибір програмного забезпечення для створення головної частини додатку – Frontend середовища для роботи користувачів з інформаційними даними через графічний інтерфейс.

Існують такі етапи створення графічного інтерфейсу кросплатформенного додатку:

- створення Web-додатку (зазвичай має вигляд сайту, що відкривається у браузері);

- створення мобільного застосунку.

Почнемо з першого етапу. Створення Web-додатку напевне є простішим етапом розробки з двох наявних, тому що всі нині відомі браузери працюють за одними і тими ж принципами на всіх типах операційних систем.

У світі веб-розробки існує 3 «кита» на яких будується будь-який веб-сайт – це HTML, CSS, JS. Зараз я розповім про кожного з них детальніше.

Мова розмітки гіпертексту (HTML) – це набір текстових символів або кодів розмітки, що вставлені у файл, який призначений для відображення в Інтернеті. Розмітка повідомляє браузерам, як відображати слова та зображення веб-сторінки.

Кожен окремий текст розмітки (що розташовується між символами "<" і ">") називається елементом, хоча більшість людей називають його тегом.

Деякі елементи представлені парно, що вказують, коли якийсь ефект відображення має розпочатися або закінчитися.

За своєю суттю HTML – це набір коротких кодів, введених у текстовий файл. Це теги, що забезпечують можливості HTML. Текст зберігається у вигляді файлу HTML і відображається через веб-браузер. Браузер читає файл і трнаслює текст у форму відповідно до кодів, які автор використовував для написання того, що стає видимим зображенням. Написання HTML вимагає коректного використання тегів для створення бачення автора[7].

Теги відрізняють звичайний текст від HTML-коду. Теги – це набір символів між кутовими дужками, що дозволяють відображати графіку, зображення та таблиці на веб-сайті. Теги можуть виконувати різні функції. Найпростіші теги використовують для форматування тексту. Оскільки веб-інтерфейси повинні стати динамічнішими, можна використовувати каскадні таблиці стилів (CSS).

Cascading Style Sheets , яку люблять називати CSS – це проста мова дизайну, призначена для суттєвого спрощення процесу створення презентаційних веб-сторінок.

CSS обробляє зовнішній вигляд веб-сторінки. Використовуючи CSS, ви можете налаштовувати колір тексту, стиль шрифтів, інтервал між абзацами, розмір і розташування стовпців, фонові зображення або кольори що використовуються, дизайн макету, варіанти відображення для різних пристроїв і масштабів екрана. а також ряд інших ефектів[8].

Головними перевагам CSS є такі:

CSS економить час – ви можете написати CSS один раз, а потім знову використовувати той самий код на кількох сторінках HTML. Ви можете визначити стиль для будь-якого елемента HTML і застосувати його для будь-якої кількості веб-сторінок.

Сторінки завантажуються швидше – якщо ви використовуєте інструмент CSS, вам не потрібно кожного разу писати атрибути тегу HTML.

Просто напишіть один код CSS для тегу та застосуйте його до всіх входів цього тегу. Таким чином, менша кількість коду означає швидше завантаження.

Просте обслуговування – щоб внести глобальні зміни, просто змініть стиль, і всі елементи на всіх веб-сторінках оновляться автоматично.

Кращі стилі перед HTML – CSS має набагато ширший набір атрибутів, ніж HTML, тому ви можете надати набагато кращий вигляд своїй HTML-сторінці порівняно з атрибутами HTML.

Сумісність кількох пристроїв – таблиці стилів дозволяють оптимізувати вміст для більш ніж одного типу пристроїв. Використовуючи той самий документ HTML, можна представити різні версії веб-сайту для портативних пристроїв, таких як КПК і мобільні телефони, або для друку.

Глобальні веб-стандарти – атрибути HTML є застарілими, і рекомендується використовувати CSS. Вварто почати використовувати CSS на всіх сторінках HTML, щоб зробити їх сумісними з майбутніми веб-браузерами.

Поєднання технологій HTML та CSS вже на цьому етапі дає нам можливість отримати веб-сайт з чудовим зовнішнім виглядом, проте сайт на даний момент не буде мати ніякого функціоналу та можливості з ним взаємодіяти.

Для цього ми будемо використовувати раніше сказаний інструмент JS. JS – це скорочена назва мови програмування JavaScript.

Javascript (JS) – це мова сценаріїв, що здебільшого використовується в Інтернеті. Його використовують для покращення сторінок HTML та зазвичай він є вмонтований у HTML-код. JavaScript є інтерпретованою мовою. З цього випливає, що його не потрібно компілювати. JavaScript відтворює веб-сторінки в динамічній та інтерактивній формі. Це дозволяє веб-сторінкам реагувати на події, показувати спеціальні ефекти, приймати не статичний

текст, досліджувати дані, створювати файли cookie, виявляти веб-браузер користувача тощо[9].

Сторінки HTML дуже добре підходять для показу статичного вмісту, наприклад, простого тексту або зображення. Проте сьогодні абсолютна більшість сторінок не є статичними. На багатьох сучасних веб-сторінках є меню, форми, анімації і навіть зображення, що забезпечують взаємодію з користувачем. Javascript – це мова сценаріїв, яку використовують веб-розробники для створення можливості такої взаємодії. Так як JavaScript працює з веб-сторінками HTML, розробнику потрібно знати HTML, щоб використати весь ресурс цієї мови сценаріїв. Проте існують інші мови, які можна використовувати для різних типів сценаріїв в Інтернеті, на практиці це, по суті, весь Javascript.

Існує два способи підключення JavaScript у файлі HTML. Перший передбачає вмонтування коду JavaScript в код HTML, в той момент другий метод використовує окремий файл JavaScript, що викликається з елемента Script, тобто укладений тегами Script. Файли мови JavaScript мають розширення .js. Хоча JavaScript здебільшого використовується для взаємодії з об’єктами HTML, його також можна використовувати для взаємодії з іншими об’єктами, які не є HTML-сторінками, такими як плагіни для браузера, властивості CSS, поточний час та дата або інформацію про браузер. Щоб написати код JavaScript, вам потрібен лише звичаний текстовий редактор, наприклад блокнот у ОС Windows, Gimp в ОС Linux або BBEdit. Деякі текстові редактори, як-от GIMP, мають підсвічування синтаксису для JavaScript. Це дозволить вам одразу ідентифікувати елементи структури JavaScript.

Зараз ми вже можемо створити повноцінний веб-сайт, який може взаємодіяти з користувачем та розуміти які дії він виконує. Але цей підхід потребує окремого використання всіх вище описаних технологій, всі файли потрібно буде зберігати у різних файлах, і відслідковувати їх цілісність

щоразу при запуску проекту, що зовсім недопустиме при створенні великих та потужних проектів. З цього виникає логічне питання - чи існує рішення, що дозволяє виконувати всі ці речі в одному місці? Так, для цього ми будемо використовувати середовище Node.js.

Node.js - це програмна платформа, що робить JavaScript мовою загального призначення. Node.js також називають середовищем виконання сценаріїв JS. Вона вміє працювати з зовнішніми бібліотеками, запускати команди з коду та виконувати роль веб-сервера. Щоб пояснити простіше, цей інструмент дає можливість додати до повністю клієнтської мови програмування, серверну частину, дозволяючи робити з його допомогою не тільки веб-сайти, а також і повноцінні програми, без залучення браузера.

Як і JavaScript, Node запускається у вигляді V8. Це середовище виконання, що конвертує написане веб-розробником в машинний код – набір прямих сценаріїв для комп'ютера, які не потребують інтерпретації. Таким чином, низькорівневий код стає швидким і комфортним для розпізнавання машиною.

Завдяки цьому програмному забезпеченю можна створити як програму для Інтернету, також і для ОС Linux, OS X і ОС Windows. Використовуючи додаткові модулі, також можна сформувати API. Додаються шляхи синхронізації мобільного девайсу з комп'ютером – при написанні тексту на телефоні, його буде видно в Інтернеті та на ПК/ноутбуці. На відміну від звичайної мови сценаріїв JavaScript, Node.js вміє взаємодіяти з глобальними об'єктами, включаючи window і document, при створенні програм для Windows. Це дає можливість відкрити шлях до вінчестера та файлової системи девайсу. Крім цього, у полі доступу також є бібліотеки та програмні рішення, що вже є на комп'ютері.

Але Node.js це лише набір технологій, що дозволяють створювати проекти на їх основі. На їх основі створюються фреймворки, які вже в собі містять набір готових функцій для окремих задач. Це суттєво зменшує час

розробки та підвищує комфортність при розробці програмного забезпечення. На сьогоднішній день існують такі найбільш популярні фреймворки для створення Front-end інтерфейсів:

- Angular.js;
- Vue.js;
- React.js.

В своєму проекті я буду використовувати React.js.

React – це бібліотека на основі JavaScript, розроблена компанією Meta, яка, серед іншого, була основною для створення Instagram.com. Його ціль – дозволити розробникам легко та стандартизовано створювати швидкі інтерфейси користувача для веб-сайтів і програм. Головною концепцією React.js є віртуальний DOM. Це дерево, що базується на основі компонентів JavaScript, створених за допомогою React,js, що імітує дерево DOM. Він робить найменшу кількість маніпуляцій з DOM, щоб підтримувати ваші компоненти веб-сайту React.js в актуальному стані[10].

Будучи частиною мови сценаріїв JavaScript, використання React.js має багато переваг. Продукти, створені на основі React, прості в масштабуванні, єдина мова, що використовується на стороні сервера, клієнта, мобільного пристрою, показує чудову продуктивність, також існують шаблони робочих сценаріїв для зручної роботи в команді, код інтерфейсу читається та підтримується. Відомі світові компанії використовують React.js та інші технології JavaScript у деяких із найкращих продуктів, які визначають ринок (Instagram, Reddit та Facebook – найкращі приклади).

Однією з головних причин використання React.js для розробки є в кінцевому рахунку оптимізований та зручний інтерфейс розробки бібліотеки та мова кодування. З цього ми отримуємо легкий API React з посиленими швидкісними можливостями для забезпечення якісного та швидкого робочого процесу розробки. Компоненти та концепції React є простими для пізнання, тому тут не так багато потрібно вчитися.

На відміну від інших відомих та популярних фреймворків, таких як Vue та Angular, тут немає зайвих тегів HTML (створюваних, коли JavaScript «вшитий» у HTML – стандартна практика для звичайних фреймворків і рішень бібліотек JavaScript). У перспективі, вбудовуючи JSX в JavaScript (буквально навпаки), React віддає набагато чистіший, краще читабельний, більш потужний код.

Використання React.js для веб-розробки може виявитися надзвичайно зручним, оскільки React – це один із тих випадків, коли ви освоюєте одну технологію, щоб легко повторно використовувати її на різних платформах. І все завдяки тому, що вона є бібліотекою за своєю природою, основною метою якої є створення окремих елементів і компонентів веб-дизайну (що завгодно, від кнопок і міток до сіток та інтерактивних функцій).

Крім того, є великий внесок давно існуючої спільноти. Поточна екосистема React.js настільки широка, що дозволяє розробникам створювати десктопні рішення та мобільні додатки, статичні сайти, обробляти серверний рендеринг та використовувати передові технологічні концепції із веб-рішеннями – і все це лише за допомогою подібних простих технологій веб-розробки на React,js.

Як вже було вище сказано, я вирішив використовувати бібліотеку React.js саме тому, що вона дає можливість створювати додаток одночасно як веб-додаток та мобільний застосунок. Можливість використовувати один і той же код сильно прискорить процес розробки проекту та дозволить вести стилістично правильний процес розробки без примінення інших рішень.

На цьому етапі опис технологій для створення Frontend частини додатку – завершено.

Наступним пунктом є опис технологій передачі даних між серверною та клієнтською частиною.

На початку роботи я розповідав про технологію RESTful API та чому вона не є безпечною у більшості випадках її використання. Про те, у моїй

роботі є інформація, яка не потребує сильного захисту, тому я вирішив використовувати – REST API.

RESTful API – це вид зв'язку (API), що використовють веб-сайти для зв'язку між клієнтською та серверною частиною. Він використовує HTTP-запити для доступу та отримання даних. Ці дані можна використовувати для типів запитів GET, PUT, POST і DELETE, що стосується зчитування, оновлення, створення та видалення даних, що стосуються ресурсів[11].

Код, що дозволяє двом програмним забезпечення взаємодіяти один з одним має назву – API для веб-сайту. API визначає коректний спосіб для розробників, щоб написати програму, котра надає можливості обміну інформацією.

RESTful API – також назввний веб-сервісом RESTful та REST API – заснований на принципі передачі стану репрезентації (REST), який є архітектурним стилем і підходом до комунікацій, що часто використовуються при розробці веб-сайтів.

Технологія REST має багато переваг над іншими подібними технологіями. Оскільки REST має меншу пропускну здатність, яка робить його більш кращим у ефективному використанні Інтернету.

REST, що використовують браузери, можна розглядати як мову спілкування в Інтернеті. Із зростанням використання хмарних середовищ, API-інтерфейси використовують споживачі інтернет ресурсів для надання та організації доступу до веб-сервісів. REST – це коректний вибір для створення API, що дозволяють користувачам підключатися до віддалених сервісів та керувати ними. RESTful API використовують такі компанії як Amazon, Google, LinkedIn і Twitter.

RESTful API розбиває запит на створення певної кількості невеликих модулів. Кожен модуль звертається до конкретної частини транзакції. Використання модульності надає розробнику велику гнучкість при розробці, але здебільшого розробити свій REST API з нуля може бути складно.

RESTful API використовує команди для отримання даних. Стан запиту в будь-який момент часу називається представленням ресурсу. RESTful API базується вже на існуючих методологіях HTTP, які визначені протоколом RFC 2616, наприклад:

- GET – щоб отримати дані;
- PUT – щоб змінити стан або оновити дані, що є у вигляді об'єкту, файлу або блоку;
- POST – щоб створити об’єкт;
- DELETE – щоб видалити об’єкт.

Це метод передачі даних, який є зручним для використання на невеликих платформах, що дозволяє швидко та комфортно отримати інформацію від бази даних без налаштувань серверу та використання спеціальних технічних засобів для його коректної роботи. Схематичний прицип його роботи ви можете побачити на рисунку 2.1.

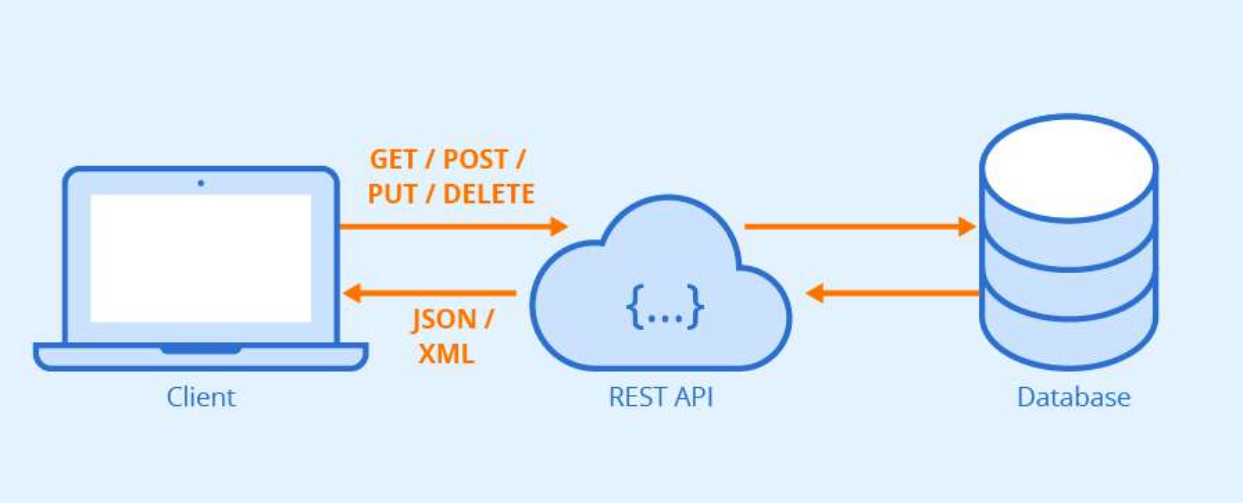


Рисунок 2.1 - Ілюстрація роботи REST API

Проте, головним мінусом є те, що REST API вміє лише отримувати запити від клієнта, але не вміє надсилати їх в зворотньому напрямку. Так як мій проект має закладений в плані розробки модуль чату для обміну інформацією – потрібно обрати інший метод з’єднання з сервером. Для цього я буду використовувати технологію WebSockets.

WebSocket – це постійний двонаправлений канал зв'язку між клієнтом (наприклад, браузером) і серверною службою. На відміну від HTTP-з'єднань запитів/відповідей, веб-сокети можуть передавати будь-яку кількість протоколів і забезпечувати доставку повідомлень від сервера до клієнта без опитування[12].

WebSockets встановлюють з'єднання у стилі TCP сумісним із браузером способом за допомогою HTTP під час початкового налаштування. Повідомлення через веб-сокети можна надавати в будь-якому протоколі, звільняючи програму від іноді непотрібних накладних витрат на HTTP-запити та відповіді (включаючи заголовки, файли cookie та інші артефакти). Але найбільш важливою є можливість доставки повідомлень від сервера до клієнта підключеним клієнтам. Наприклад, у браузері те ж саме було колись можливим лише за допомогою опитування ресурсу сервера, що є порівняно гострим, високою затримкою та інтенсивною пропускною здатністю.

WebSockets доступні на багатьох платформах, включаючи найпоширеніші браузери та мобільні пристрої. Вони часто застосовуються для вирішення проблем синхронізації стану з точністю до мілісекунди та обміну повідомленнями публікування-підписки , обидва з яких використовують забезпечення Websockets для низхідних натискань. Проблемою експлуатації системи на основі WebSocket є підтримка шлюзу з визначенням стану на серверній частині. WebSocket створюється шляхом виконання загального HTTP-запиту до цього сервера із Upgrade заголовком, який сервер (після аутентифікації та авторизації клієнта) повинен підтвердити у своїй відповіді. Після цього з'єднання залишається встановленим між цією фізичною парою клієнт-сервер; якщо в якийсь момент послугу потрібно повторно розгорнути або перерозподілити навантаження, її з'єднання WebSocket потрібно повторно встановити. Принцип роботи WebSocket зображено на рисунку 2.2.

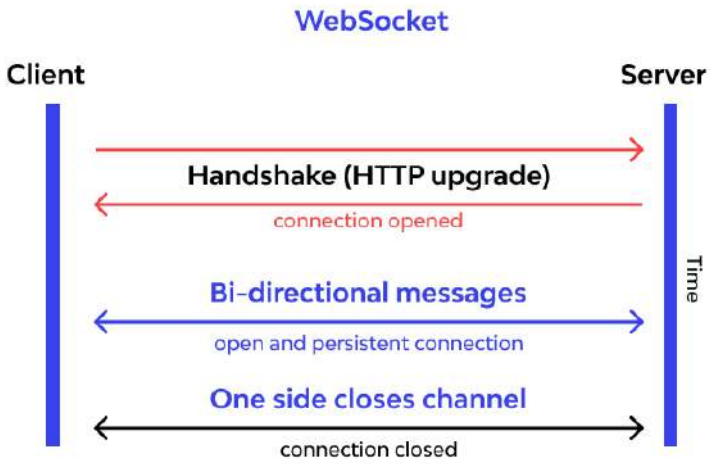


Рисунок 2.2 - Ілюстрація роботи WebSockets

Ця технологія повністю задовольняє потреби для створення модулю чату системи захисту внутрішньої конфіденційної інформації кросплатформової системи, що дозволить отримувати нові повідомлення в той же момент, коли вони були відправленні без потреби перезавантажувати сторінку і виконувати інші дії.

3 ПРОЕКТУВАННЯ СИСТЕМИ БЕЗПЕКИ

3.1 Аналіз джерел загроз

Головною метою технічного завдання є створення системи захисту внутрішньої конфіденційної інформації кросплатформового додатку компанії на прикладі програмного забезпечення, яке буде зберігати список поточних задач на конкретному проекті. Звісно, система повинна бути створена з урахуванням останніх стандартів безпеки в інформаційному просторі, але не потрібно також забувати про економічну сторону цього питання. Всі наявні готові рішення здебільшого використовуються компаніями, що мають велику кількість співробітників, досить обширну клієнтну базу, але що робити, якщо компанія не готова переплачувати за функціонал, яким вона не буде користуватись? Більшість компаній малого рівня використовують для цього - месенджери.

Технологія, що лежить в основі додатків для обміну повідомленнями, широко використовується з початку 90-х, але вона дійсно зросла популярністю на початку 2000-х з появою додатків для обміну повідомленнями, таких як AOL Messenger. До 2022 року кількість повідомлень, що надсилаються через ці програми, перевищила кількість SMS-повідомлень.

Тепер програми для обміну повідомленнями приваблюють мільярди користувачів, і вони використовуються у всьому світі. Зараз у WhatsApp 1,6 мільярда користувачів, у Facebook месенджера – 1,3 мільярда, а у популярної платформи обміну повідомленнями WeChat – 1,1 мільярда користувачів. Додатки Messenger все частіше використовуються в обслуговуванні клієнтів, щоб клієнти могли легко спілкуватися з брендами щодо проблем із обслуговуванням та питань щодо продуктів. За допомогою Facebook Messenger, чату в реальному часі або обміну повідомленнями в додатку клієнти можуть звернутися до брендів для негайної реакції.

Бренди також можуть використовувати автоматизацію та штучний інтелект, щоб боти швидкого пошуку забезпечували швидкий середній час до першої відповіді та ідеальний час вирішення . У той час, коли клієнти вимагають швидких відповідей та персоналізованого досвіду, додатки для обміну повідомленнями дозволяють і те, і інше, заощаджуючи гроші організації.

Месенджери дійсно є простим та популярним рішенням для ведення бізнесу, але в нього наявні такі явні мінуси:

- робота лише з підключення до мережі Інтернет;
- можливість втратити доступу до акаунтів керівних ланок;
- хакерські атаки, що загрожують інформаційній безпеці компанії;
- відсутність можливості експорту даних;
- неявна аутентифікація користувача.

Багато людей замінили записні книжки з важливою інформацією на повідомлення в особистих чатах. Це є додатковою загрозою при використанні сторонніх ресурсів, де при хакерській атаці зловмисник отримує не тільки доступ до вашого аккаунту, а одразу до всіх акаунті навіть на сторонніх сервісах, тому, що ви фактично самі показали йому шлях та ключ від дверей до вашого особистого облікового запису.

Отже було виявлено такі аспекти в інформаційній безпеці, на які може бути спрямована загроза на підприємстві:

- конфіденційність – тобто неправомірний доступ до інформації. Його суть полягає в тому, що передаючи інформацію, на прикладу по телефону вона може стати доступниою для того, хто не повинен володіти нею. Це проблемне місце для тих випадків, коли дані передаються від однієї системи до іншої. Це може статись, коли було отримано доступ до конфіденційної інформації, яка вже знаходиться в системі. Не можна виключати того, що такі загрози можуть утворюватись внаслідок людського фактору. Тобто можливий випадок не вірного надання прав іншому користувачеві. Також

такий вид загрози може утворитися через збій у програмних або апаратних засобах;

- цілісність – ця загроза може виникнути внаслідок не правильних навмисних дій персоналу. Також можливий варіант, коли ця загроза була створенна через збій в обладнанні чи системі;

- доступність – не менш небезпечна загроза, яка являє собою те, що можливе створення умов, під час яких буде заборонений або ж обмежений доступ до ресурсів або інформації.

Цілісність – відноситься до антропогенних джерел загроз. Антропогенними джерелами загроз можуть бути суб'єкти, чиї дії класифікуються як навмисні, так і ненавмисні злочини. Це джерело є найпоширенішим і являється найцікавішим моментом організації захисту, через те що дії суб'єкта можна прогнозувати та прийняти відповідно адекватні дії.

До антропогенних джерел відносяться суб'єкти, що мають несанкціонований/санкціонований доступ до роботи зі штатними засобами підприємства, а їхні дії можуть бути зовнішніми і внутрішніми. Джерела загроз, що є результатом діяльності людини та розвитку цивілізації, складніше прогнозуються і є залежними від властивостей технічних засобів, і тому потребують особливої уваги.

Ця класифікація джерела загроз надзвичайна актуальна в сучасному світі, через те, що спеціалісти в сфері кібербезпеки очікують різкого зростання кількості техногенних катастроф, що викликані матеріальним та моральним старінням технічної бази обладнання, а також відсутністю матеріального забезпечення для її покращення[16].

Зовнішні загрози з'являються внаслідок дій шахраїв, хакерів та недобросовісних партнерів, представників контролюючих органів, що зловживають службовим становищем. Внутрішні випадкові загрози пов'язані з відмовавою в обслуговуванні обчислювальної помилками програмного

забезпечення, та іншими діями, що порушують нормальну роботу компанії.

На рисунку 3.1 зображено класифікацію антропогенних загроз.

Рисунок 3.1 - Класифікація антропогенних загроз

Аналіз класифікації показаної на рисунку 3.1 свідчить про те, що випадкові загрози, що пов’язані з помилковими діями, виконуються співробітниками випадково, через неуважність, незнання чи халатність, або просто з цікавості. При цьому, ці дії не мають якогось поганого наміру. До головних ненавмисних загроз належать наступні:

- випадкові дії, які спричиняють збої системного обладнання, програмного забезпечення та інформаційних ресурсів;

- неправомірне підключення чи відключення пристроїв та зміна режиму роботи програм та обладнання;

- ненавмисне пошкодження засобів збереження інформації;

- запуск програм, що здатні при неправильному використовуванні викликати можливість коректної працездатності програмного забезпечення або виконати незворотні зміни в системі;

- незаконне встановлення та використання програм, що як наслідок, пізніше стане необґрунтованим використанням ресурсів;

- зараження операційної системи вірусами;

- неправильна поведінка, що призводить до поширення або розкриття конфіденційної інформації;

- ігнорування організаційних правил при роботі в системі;

- відсилання даних на некоректну адресу отримувача;

- введення некоректної інформації;

- ненавмисне виведення з ладу каналів зв'язку.

Водночас умисні джерела загроз пов'язані, здебільшого, зі свідомими діями працівників для виконання своїх планів[17].

До найпоширеніших навмисних загроз відносяться такі:

- несанкціонований доступ до інформації, що зберігається в пам'яті комп'ютера з метою незаконного використання;

- розробка спеціального програмного забезпечення, що використовується для здійснення несанкціонованого доступу та інших дій;

- заперечення дій, які пов'язані з маніпуляціями даними та інші незаконі дії;

- введення в програмне забезпечення та проекти «логічних бомб», що спрацьовують при певних умовах або через певний період частково чи повністю виводять з ладу комп'ютерну систему;

- розробка та розповсюдження вірусів;

- фальсифікація цифрових підписів;

- викрадення даних з подальшим маскуванням;

- перехоплення даних;

- приховування факту викрадення інформації;

Залежність внутрішніх загроз передбачає використання зловмисником ненавмисних помилок працівників, що мають доступ до захищенної інформації. А незалежність внутрішніх загроз передбачає відсутність вище

вказаних особливостей, тобто внутрішні загрози виникають в результаті самостійних дій злочинця.

3.2 Моделювання інформаційної системи

Для початку розробки системи захисту внутрішньої конфіденційної інформації кросплатформового додатку потрібно спроектувати та створити модель роботи цієї системи. Показати всі процеси, їх послідовність виконання і варіанти різних типів розвитку подій. Після цього потрібно створити контекстну діаграму IDEF0. На основі цього вже створюється декомпозиція попередньої діаграми та варіанти використання інформаційної системи.

Спочатку необхідно створити контекстну діаграму IDEF0, що відображена на рисунку 3.1. Головною метою є створення кросплатформенного додатку для зберігання інформації.

Діаграма вище демонструє загальний опис інформаційної системи. Для того, щоб побачити повний опис, потрібно створити декомпозицію цієї системи. Це детально покаже логіку та послідовність виконання роботи.

Перший етап – створення Back-end частини. Вхідні дані: технічне завдання для розробки кросплатформенної системи обміну та зберігання інформації. Механізми: технічне забезпечення, розробник, Laravel (PHP), MariaDB. Керування:, документація Laravel (PHP), документація MariaDB. Вихідні дані: API.

Другий етап – розробка клієнтської частини. Вхідні дані: технічне завдання для розробки. Механізми: технічне забезпечення, розробник, Node.js, React. Керування: документація React. Вихідні дані: user interface (UI).

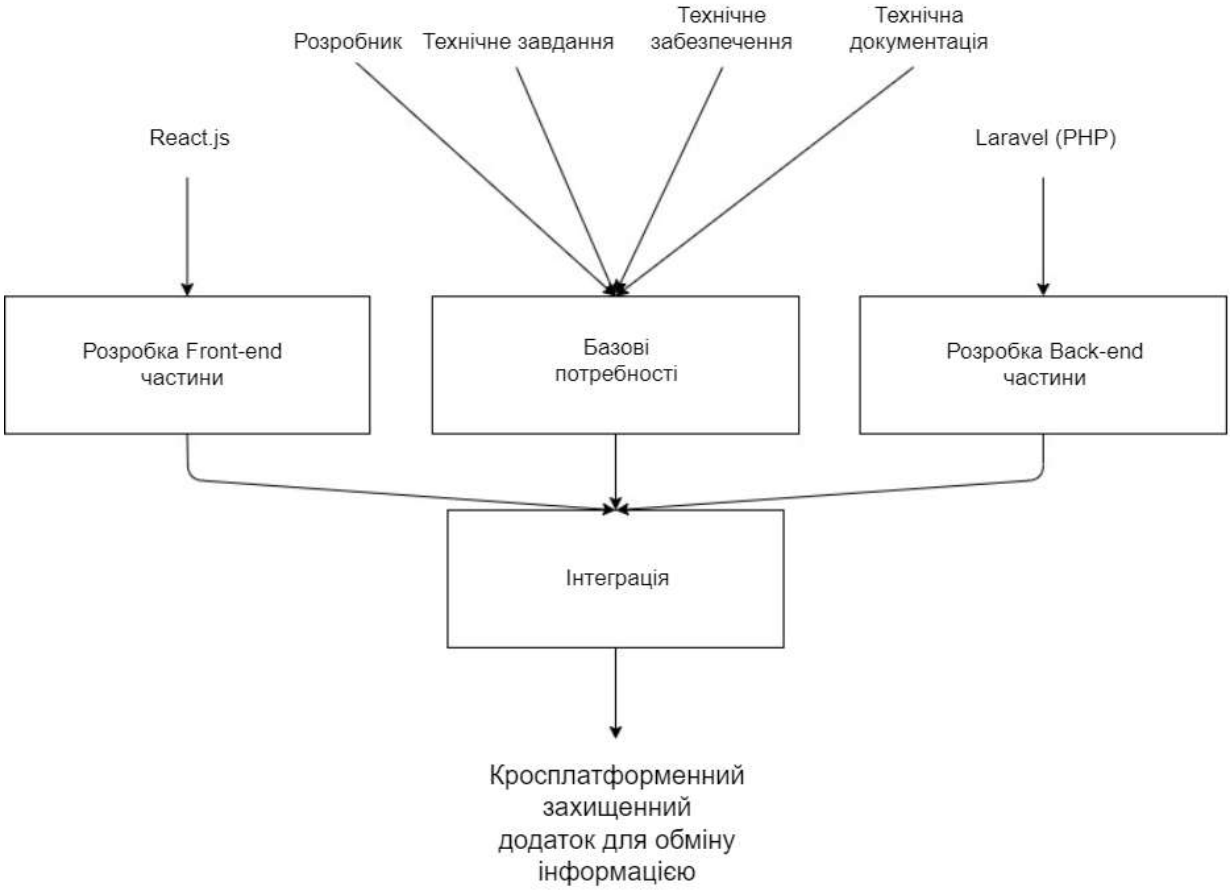


Рисунок 3.2 - Діаграма IDEF0

Третій етап – інтеграція Backend та Frontend частин. Вхідні дані: API, Pusher Private Key. Механізми: RESTful API, WebSockets. Керування: документація Pusher. Вихідні дані: обмін даними всередині додатку.

Четвертий етап – налаштування та розміщення додатку у хмарному просторі. Вхідні дані: Vercel Private Key, App Store Private Key та Google Private Key, GitLab Private Key. Механізми: Vercel Cloud Platform, App Store, Google Play. Керування: документація Vercel. Вихідні дані: web-hosting, mobile store, CI/CD process.

П’ятий етап – додаток для зберігання та обміну інформацією. Вхідні дані: API, UI, Hosting, Mobile Store. Механізми: API, WebSockets. Керування: документація WebSockets. Вихідні дані: Кросплатформенна система обміну та зберігання інформації.

Тепер розглянемо кожен етап детальніше за допомогою розгорнутої діаграми IDEF0 на рисунку 3.3.

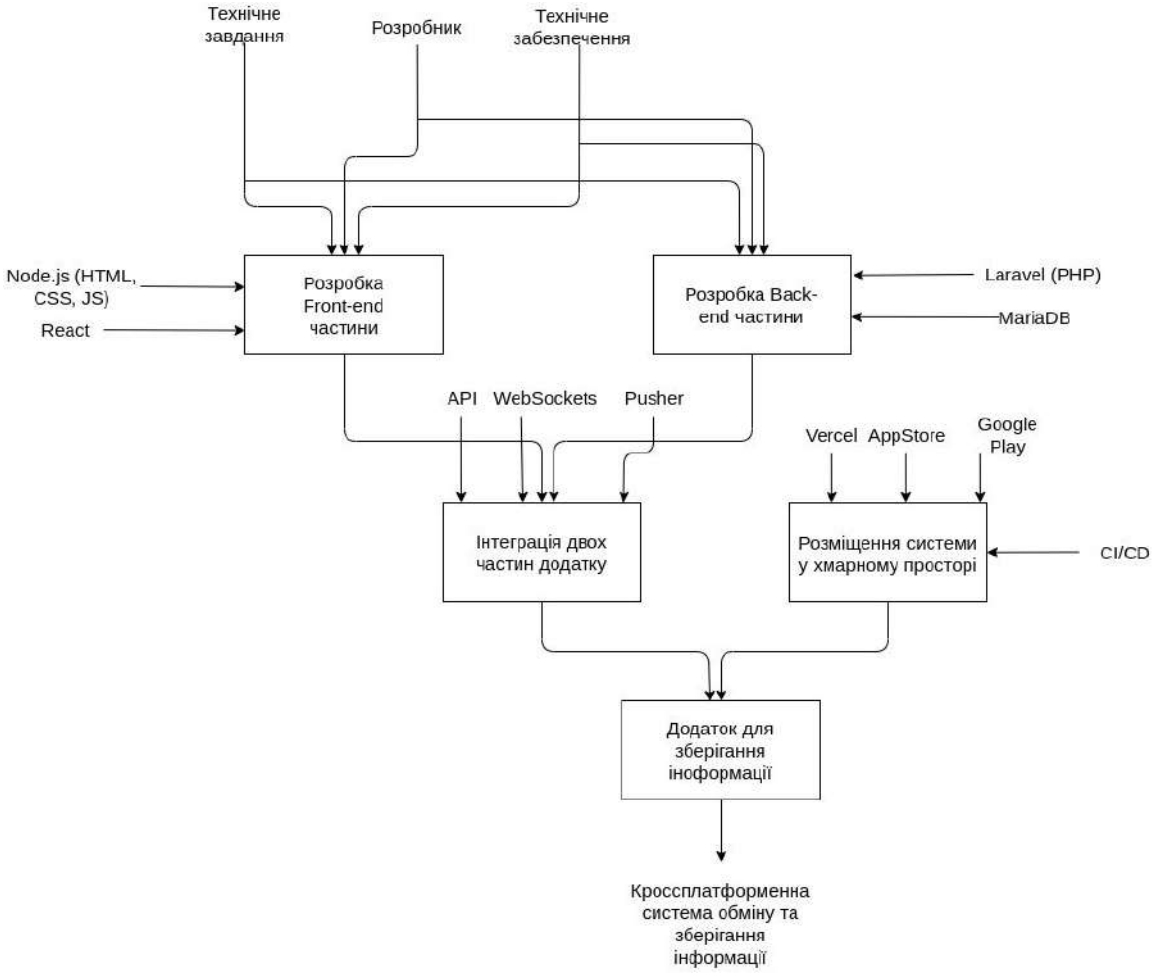


Рисунок 3.3 – Розгорнута діаграма IDEF0

Для більш зручної роботи над проектом було створено репозиторії на GitLab. Ця платформа була використана для контролю версій та CI/CD[15].

На першому етапі ми створюємо Back-end частину додатку за допомогою вивчення технічної документації Laravel та MariaDB. Тут ми створюємо моделі бази даних, систему роутингу для надсилання та отримання даних з клієнтської частини додатку. На виході ми отримуємо API, до якого потрібно звернутися, щоб отримати дані з серверу.

На другому етапі ми створюємо Front-end частину додатку за допомогою вивчення технічної документації React. Тут ми налаштовуємо ту

ж саму систему роутингу, але вона вже призначена для переходу між сторінками та екранами додатку. Також налаштовуємо стилі для всіх наявних компонент. На виході отримуємо готовий UI.

На третьому етапі ми поєднуємо попередніх дві частини в одне ціле. Налаштовуємо зв’язок між інтерфейсом та серверною частиною. Для звичайних дій на платформі використовуємо метод передачі даних за допомогою API, а для чату Web-sockets. На виході отримуємо готовий локальний додаток для обміну та зберігання інформації.

На четвертому етапі ми робимо додаток доступним для всіх користувачів за допомогою перенесення його на хостинг та завантаження його в магазини мобільних додатків. Для цього використовуємо технологію CI/CD, яка після змін у репозиторії автоматично оновлює систему на хостингу до актуальної версії. На виході отримуємо додаток, вже у хмарному доступі.

На останньому п’ятому етапі ми налаштовуємо хостинг та магазини надаючи права на завантаження додатку всім користувачам та доступ до веб-версії додатку.

4 РЕАЛІЗАЦІЯ РОБОТИ

4.1 Захист кросплатформенного додатку для безпечного обміну інформацією

Головною частиною захисту внутрішньої конфіденційної інформації кросплатформової системи платформи є метод комбінування різних типів зв’язку по типу клієнт-сервер та наявність SSL-сертифікату на сервері.

Сертифікат SSL потрібний для того, щоб зловмисники не змогли перехопити особисту інформацію, яку користувачі вводять на вашому сайті. Особисті дані – це логіни та паролі від облікових записів, номери банківських карт, адреси електронної пошти та інші. Це свідчить, що SSL-сертифікат стане в нагоді на великих корпорацій, де постійно проходять транзакції різного типу.[13].

До прикладу клієнт оформлює замовлення у вас на сайті. Щоб заплатити за товар, він вписує дані кредитної картки. Коли замовлення підтверджено, інформація надсилається на ваш веб-сервер. На цьому етапі її можуть перехопити злочинці.

Якщо на сайті є SSL-сертифікат, між браузером вашого клієнта та сайтом встановлюється захищене з'єднання. У цьому випадку браузер спочатку конвертує номер картки на випадковий набір літер та символів і лише потім відправляє його на сервер. Розшифрувати повідомлення не вийде без знання спеціального ключа, що зберігається на сервері. Якщо злочинці перехоплять інформацію, вони не зрозуміють, що вона означає.

Далі йде вже програмна частина захисту даних. Використовуючи метод передачі даних REST API захист інформації відбувається за допомогою таких етапів:

- на сервері створюється список довірених доменних імен, після чого сервер відповідає на запити лише з відомих йому доменів;

- складна структура URL шляху для коректного встановлення з’єднання з сервером;

- аутентифікація користувача за допомогою JWT методу, при кожному запиті до серверу йде набір HTTP заголовків, що містить в собі додаткову інформацію про дозволи та дані користувача. В нашому випадку це заголовок запиту авторизації HTTP містить облікові дані для автентифікації агента користувача на сервері;

- перевірка ролі користувача на можливість доступу до певних розділів збереження інформації.

Етап доступу до модуля чату потребує комбінованого методу перевірки аутентифікації:

- авторизація на платформі за допомогою JWT;

- перевірка ролі, тому що, не всі користувачі мають доступ до чату, в більшості це коли адміністратор заблокував певного користувача;

- генерування унікального ключа для підключення до каналу WebSocket, що базується на умові 1 аккаунт – 1 унікальний ключ, що не дозволяє одночасно з одного аккаунту бути присутнім у чаті з двух девайсів

- при підключенні до чату проходить авторизація користувача;

- при отриманні та під час надсилання нового повідомлення проходить повторна аутентифікація для перевірки актуальності ключа користувача та неможливості його підмінни при потенціальній хакерській атаці;

Виконання всіх вище описаних пунктів гарантує вам належну безпеку при використанні цього ресурсу та конфіденційність ваших даних при збереженні їх на сервері.

4.2 Вимоги до веб-сервісу

Система захисту внутрішньої конфіденційної інформації кросплатформової системи повинна використовувати клієнт-серверну технологію.

Веб-сервіс повинен містити в собі функціонал для керівника та працівника.

Працівник повинен мати можливість входити у систему за допомогою власної email адреси.

Адміністратор повинен мати здатність користуватись веб-сервісом використовуючи свій email адрес та пароль.

У процесі розробки проекту потрібно проаналізувати засоби та технології, які створять безпечний веб-сервіс для обміну повідомленнями.

Адміністратор та працівник має стартову сторінку, що зображена на рисунку 4.1.

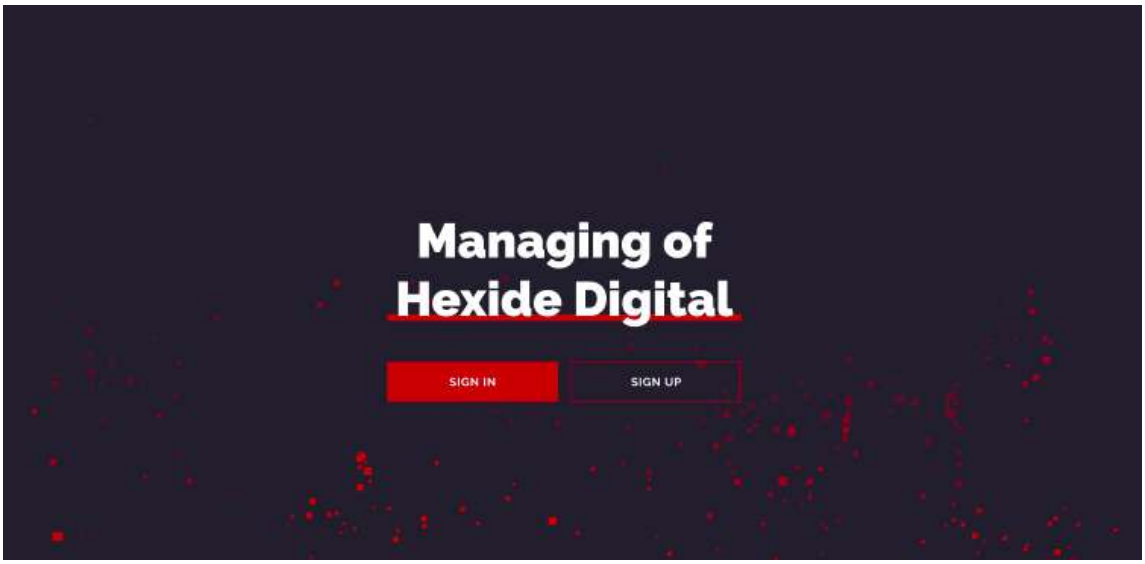


Рисунок 4.1 - Стартова сторінка веб-сервісу

Людина має змогу увійти у власний акаунт, а якщо працівник ще не має власного облікового запису - то він має змогу створити аккаунт на сторінці, що зображена на рисунку 4.2 та рисунку 4.3.

Рисунок 4.2 - Сторінка входу

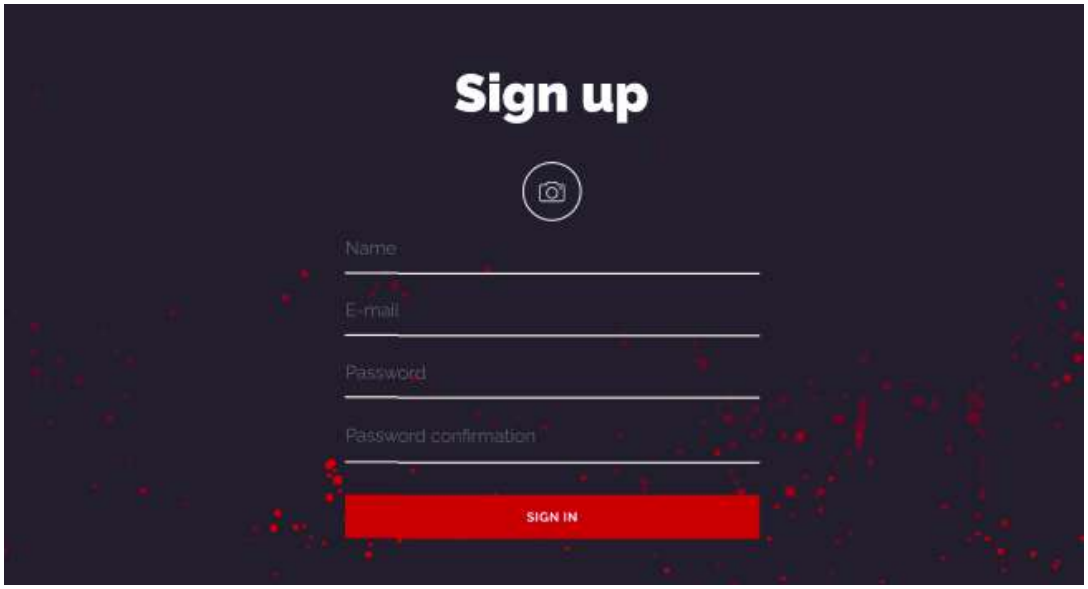


Рисунок 4.3 - Сторінка реєстрації

Але після реєстрації працівник не може увійти одразу у свій аккаунт, тому що, потрібно підтвердження статусу від керівника компанії.

Після успішної авторизації працівник отримує доступ до вже наявних записів, що зображена на рисунку 4.4.

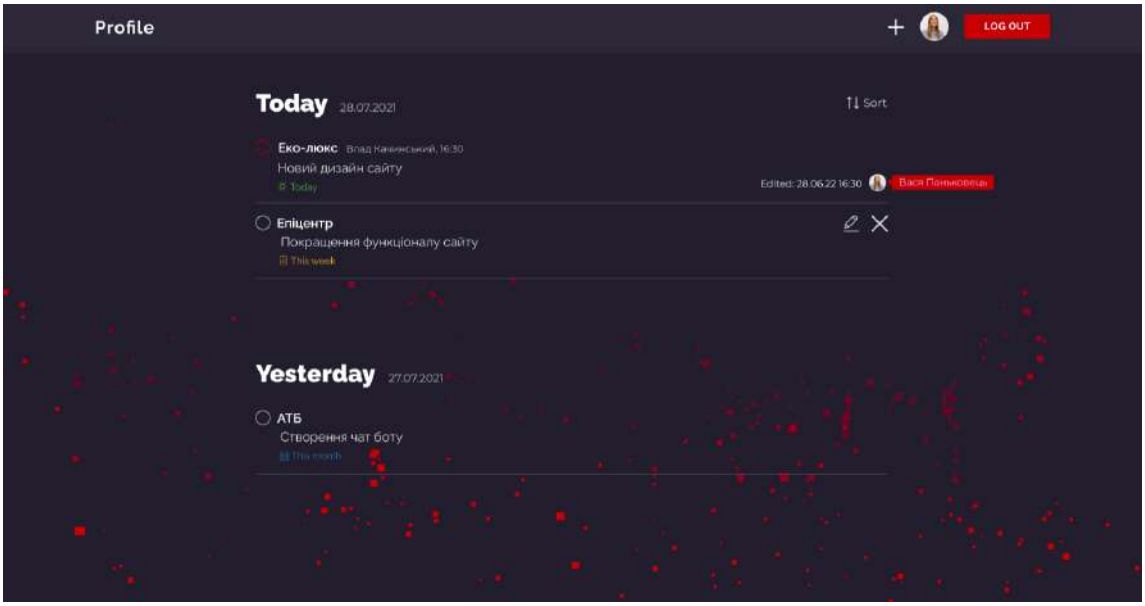


Рисунок 4.4 - Сторінка записів

Працівник на цій сторінці може створювати власні записи та переглядати записи інших користувачів. На зображені показані такі характеристики запису :

- заголовок;
- текст;
- пріоритетність;
- період, протягом якого повинна задача бути виконана;
- час створення;
- час редагування;
- автор;
- автор останніх змін запису.

Працівник може редагувати та видаляти запис лише той, який він створив, проте адміністратор може виконувати будь-які дії.

Також записи можуть бути відсортовані за датою та пріоритетністю виконання, що зображено на рисунку 4.5.

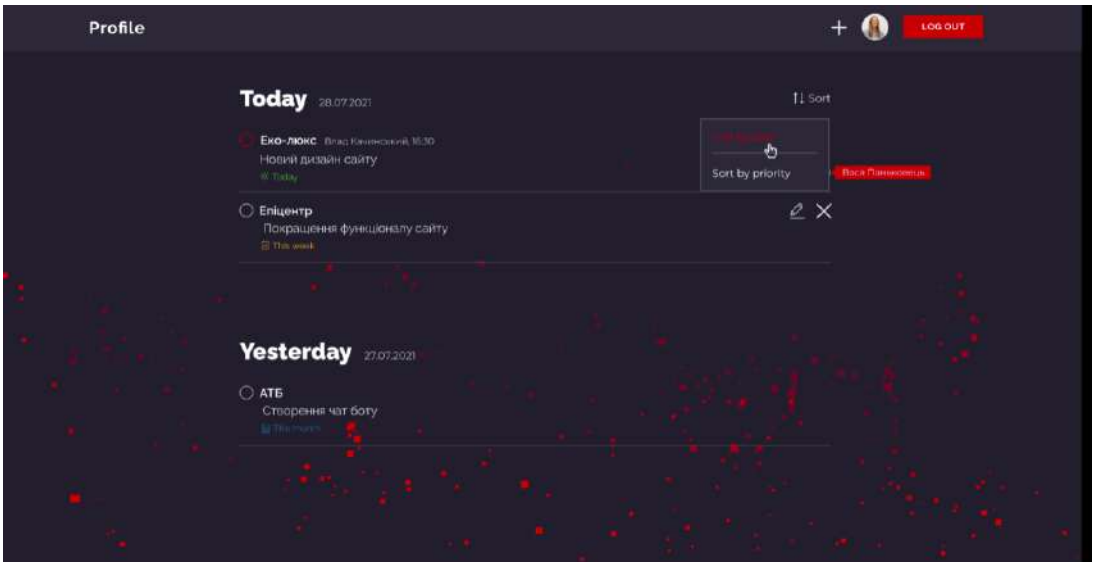


Рисунок 4.5 - Вигляд можливості сортування

Процес створення нового запису є простим та зрозумілим та зображено на рисунку 4.6.

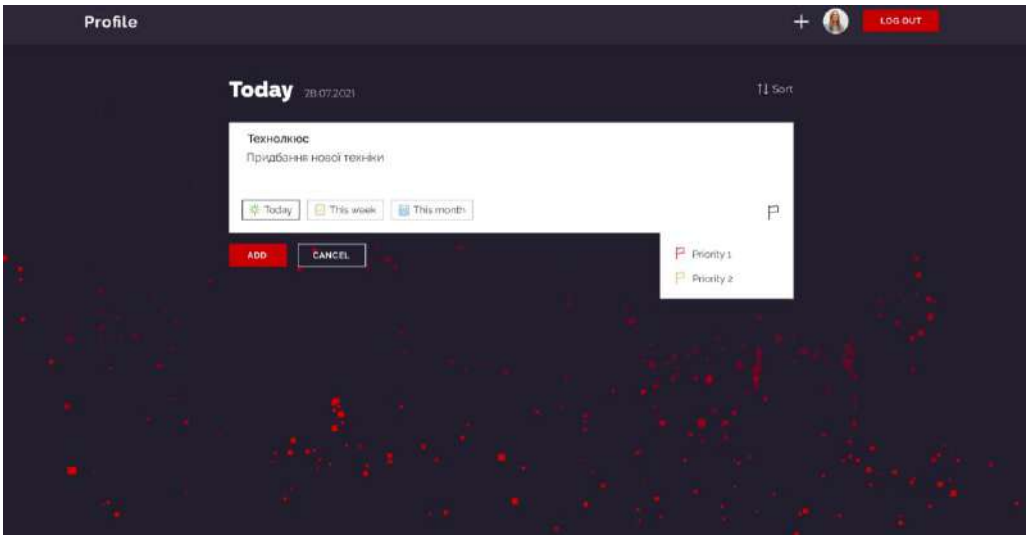


Рисунок 4.6 - Сторінки створення запису

При створені запису працівник спочатку вказує заголовок, а потім детальний опис завдання. Після цього йому потрібно обрати період, протягом якого часу ця задача повинна бути виконана, наразі на макеті ми бачимо три головних періоди, це: сьогодні, на цьому тижні та в цьому місяці. Також є можливість додатково вказати пріоритет цієї задачі, це не обов'язковий пункт, тому якщо його пропустити, то задача буде мати

пріоритет – звичайний. Проте є можливість вибрати пріоритет другої важливості, що позначається жовтим кольором та першої важливості - що позначається червоним.

Важливим аспектом у цьому моменті, є те - що якщо під час створення вашого запису у вас зник доступ до мережі інтернет, то запис буде додано у локальне сховище користувача та додане при наступному під’єднанні.

Процес редагування запису майже нічим не відрізняється від сторінки створення та виконує ті ж самі функції, що зображено на рисунку 4.7.

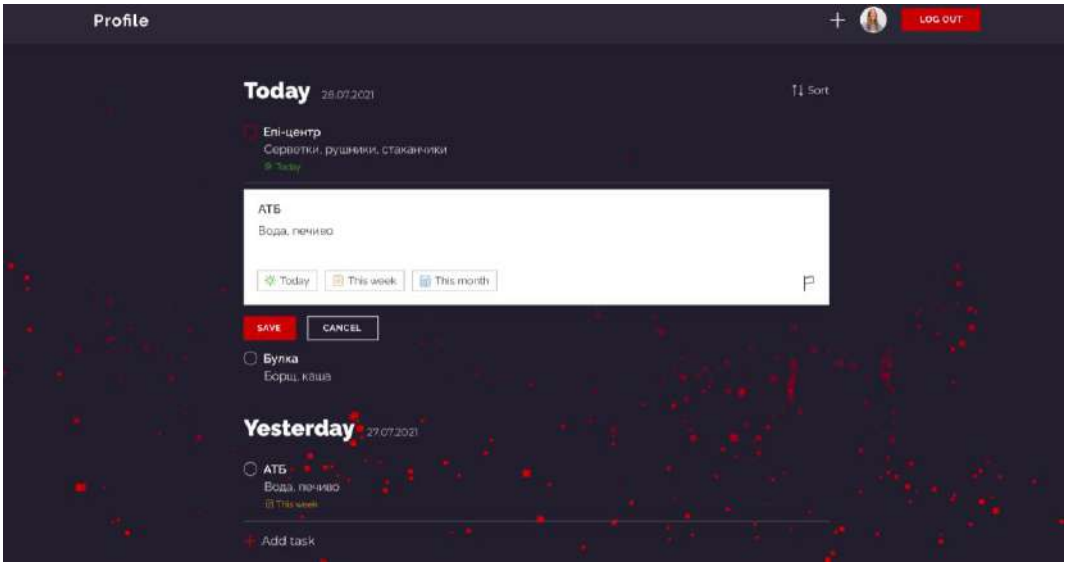


Рисунок 4.7 - Вигляд процесу редагування запису

Також у веб-сервісі є можливість перейти у розділ особистого кабінету та побачити власну інформацію про акаунт та кнопку редагування даних, що зображено на рисунку 4.8.

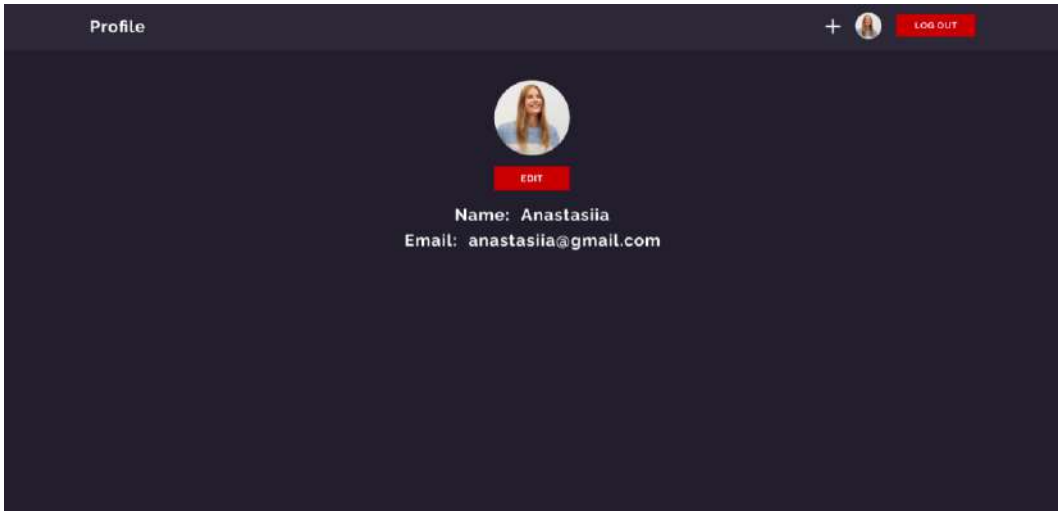


Рисунок 4.8 - Вигляд особистого кабінету

На сторінці редагування профілю ми можемо змінити світлину користувача, його ім’я та емейл, що зображено на рисунку 4.9.

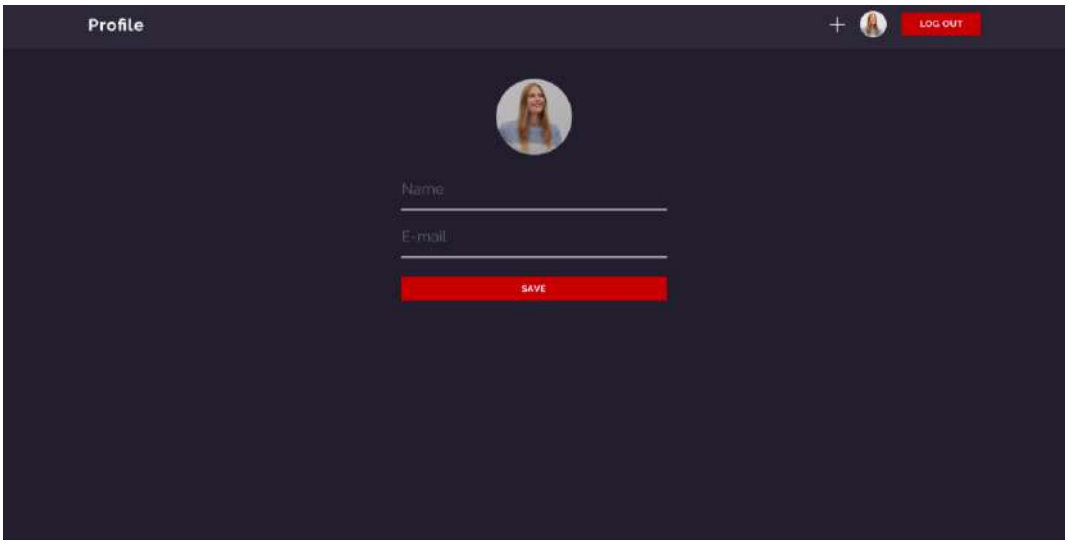


Рисунок 4.9 - Вигляд сторінки редагування особистого кабінету

Стартовою точкою для входу в додаток є файл index.tsx в якому ми підключаємо всі головні модулі для роботи:

- React.js;
- Axios;
- Styled Components;
- Pusher;

- Dotenv.

На прикладі сторінки авторизації розглянемо процес передачі інформації з клієнтської частини на серверну та отримання відповіді від нього.

Спочатку ми вводимо значення в поле емейл та пароль, після цього натискаємо на кнопку “sign in”. Після створюється запит який надсилається на сервер та йде очікування відповіді від серверу.

Для цього я використовую власну функцію, яка спрощує роботу з REST API запитами, що має назву $apiClient, яка будується на основі модулю axios.

```
const config: AxiosRequestConfig = {
  baseURL: `${process.env.REACT_APP_API_URL}/`,
  headers: {
    Accept: 'application/json',
    'Content-Type':  'application/json;multipart/form-data;application/x-
www-form-urlencoded',
  },
  withCredentials: false,
};
const $apiClient: AxiosInstance = axios.create(config);
export { $apiClient };
```

З її допомогою не потрібно щоразу вказувати посилання на серверну частину, а потрібно лише вказувати шлях до конкретного роуту. Шлях до серверної частини зберігається у файлі .env, що знаходиться в папці з проектом. Після цього він автоматично підтягується в конфіг запиту.

З використанням цього методу весь лістинг коду на запит авторизації має такий вигляд:

```
export function loginService(payload) {
  const url = 'api/v1/auth/login';
  return $apiClient.post(url, payload);
}
```

В якості змінною payload виступає інформація з форми авторизації, що є у вигляді об’єкту:

```
{
email: admin@admin.com,
password: admin.
}
```

Також ми вказуємо поруч змінну url, яка до початкової адреси серверу додає потрібну назву роуту та має кінцевий вигляд ‘api.hexide-system.com/api/v1/auth/login’, де api.hexide-system.com і є адресою серверу. Та вказуємо метод для запиту на сервер у нашому випадку це метод POST.

Після успішної авторизації сервер віддає нам JWT токен, що зображено на рисунку 4.10. За допомогою якого ми потім можемо авторизуватись при запиті на захищений роут.

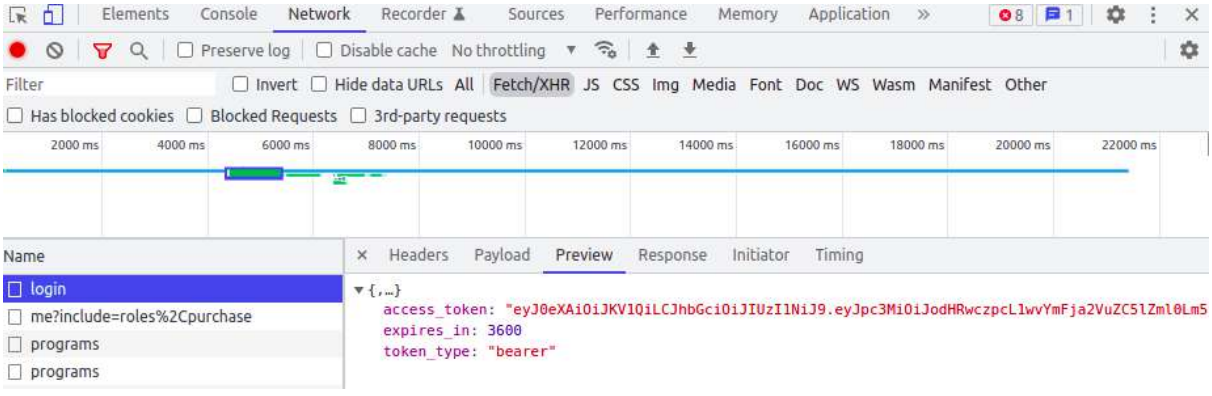


Рисунок 4.10 - Структура JWT токену

JWT токен також виступає в якості об’єкту, що має в собі такі поля

- access_token - сам токен авторизації
- expires_in - час валідності токену у секундах
- token_type - тип токену

Але вказувати щоразу цей токен при кожному запиті не є комфортним рішенням, тому я за допомогою функції $apiClient задаю його в якості значення по замовчуванню при кожному запиті в полі 'Authorization' та зберігаю його у локальному сховищі браузера, що зображено на рисунку 4.11.

```
export function setAuthAccessToken(token) {
 localStorage.setItem('access_token', token);
 $apiClient.defaults.headers['Authorization'] = `Bearer ${token}`;
}
```

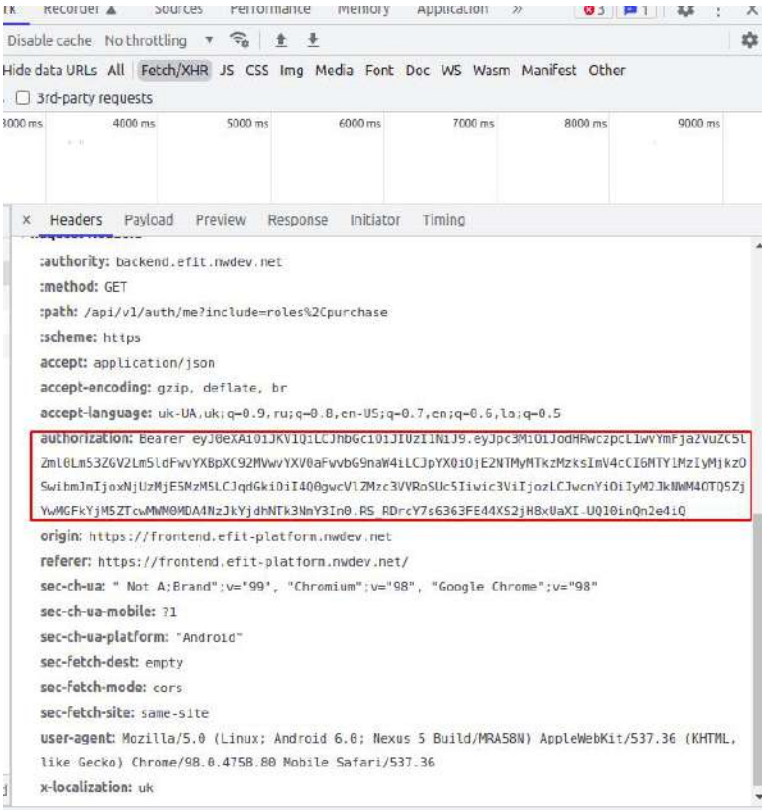


Рисунок 4.11 - Заголовок з JWT токеном

Після цього кожен запит має своїй структурі заголовків цей токен завдяки якому сервер розуміє що цей акаунт має доступ до певної інформації.

Тепер детально розглянемо процеси роботи чату.

При вході на сторінку ми ініціалізуємо клієнт Pusher та авторизуємось у ньому за допомогою вже наявного JWT-токена.

```
var pusher = new Pusher('1ca7a2e8522f70464f15', {
    authEndpoint:  'https://api.hexide-
system.com/api/broadcasting/auth',
    cluster: 'eu',
    encrypted: true,
    auth: {
      headers: {
        'Accept': 'application/json',
        'Authorization': 'Bearer ' +
window.localStorage.getItem('access_token')
      }
    },
  });
```

Після успішної авторизації, ми визначаємо канал, на який хочемо підключитись та прослуховувати на наявність нових повідомлень.

```
let channel = pusher.subscribe('presence-chat.1');

  channel.bind('App\\Events\\PrivateChat', function (data) {
    addMessages([...messages, data.message])
  });
```

Повідомлення ми зберігаємо у масиві під назвою messages. При надходженні повідомлення автоматично спрацьовує функція addMessages, що додає нове повідомлення у список наявних.

Вивід повідомлень відбувається методом перебору масиву map(), що трансформує об’єкт у HTML розмітку.

```
messages.map((el =>
<MessageOwn own={el.user_id === me.data.id} key={el.id}>
{el.content}
<br/>
<span>{el.created_at)}</span>
</MessageOwn> ))
```

В ролі елемента масиву виступає змінна el. Вона в собі містить дані про повідомлення. Поле тегу own порівнює ID користувача з ID власника повідомлення та застосовує один з стилів для відокремлення власних повідомлень від чужих.

Відправка повідомлень відбувається за допомогою функції sendMessage.

```
const sendMessage = (string) => {
    let message = {
      content: string,
      chat_id: 1,
      user_id: me.data.id
    }
    $apiClient.post('/messages', message)
  }
```

Вона формує об’єкт з даними в яких є текст повідомлення, ID чату, до якого присвоюється повідомлення та інформацію про користувача, що надіслав повідомлення.

Після успішної відправки повідомлення воно одразу з’являється у списку за допомогою методу передачі інформації WebSockets, що лежить у основі модуля Pusher.

ВИСНОВКИ

В цій кваліфікаційній роботі було створено кросплатформову систему захисту зберігання та передачі інформації. Основну роль в захисті інформації має комбінування методів передачі інформації від клієнта на сервер та надійна система управління вже наявною інформацією. На виході ми отримали веб-додаток у вигляді веб-сайту та два мобільні додатки для операційних систем IOS та Android.

Головною умовою і першочерговою запорукою успішного розвитку будь-якої сфери діяльності на сьогодні залишається інформація. Це продукт, що завжди буде користуватись попитом, та в боротьбі за нього є і буде велика конкуренція, і далеко не всі люди досягають бажаного чесними методами.

Для того, щоб покращити рівень безпеки підприємства, необхідно створювати нові та удосконалювати існуюче інформаційне забезпечення операційної діяльності підприємства. Це також відіграє важливу роль для того щоб підвищити ефективність підприємства. Важливим аспектом є те що вирішення цього завдання дозволить створити необхідні умови для того щоб покращити продуктивність ефективність та рентабельність для всього підприємства.

Використання додатку, що має хороший захист це лише половина досягнутої цілі, що дозволяє конкурувати на ринку.

Здебільшого велику роль у попиті відіграє комфорт та швидкість роботи додатку. У моєму проекті при розробці було створено інтуїтивний дизайн та добре продуманий та оптимізований зв’язок клієнтської частини з серверною. Це дозволяє користувачу використовувати додаток без різних типів проблем, головними з яких є зависання та незрозумілий для звичайного працівника інтерфейс.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. The best CRM software in 2022. Z*apier*.

URL: https://zapier.com/blog/best-crm-app/ (дата звернення: 12.04.2022).

2. What is 'Authentication'. *Economictimes*.

URL: https://economictimes.indiatimes.com/definition/authentication (дата звернення: 12.04.2022).

3. What Are The Different Types of Authentication?. *Logicmonitor*.

URL: https://www.logicmonitor.com/blog/what-are-the-different-types-of-authentication (дата звернення: 12.04.2022).

4. Broken User Authentication. *Wallarm*.

URL: https://www.wallarm.com/what/broken-user-authentication (дата звернення: 12.04.2022).

5. Cross-Platform Mobile Development. *Sam-solutions*.

URL: https://www.sam-solutions.com/blog/cross-platform-mobile-development/ (дата звернення: 12.04.2022).

6. History of Laravel. *Github*.

URL: https://manbearpig0.github.io/laravel/pages/history.html (дата звернення: 12.04.2022).

7. What is HTML – Definition and Meaning of Hypertext Markup Language. *Freecodecamp*.

URL: https://www.freecodecamp.org/news/what-is-html-definition-and-meaning/ (дата звернення: 12.04.2022).

8. What is CSS?. *W3schools*.

URL: https://www.w3schools.com/css/css_intro.asp (дата звернення: 12.04.2022).

9. What is JavaScript?. *Javascript.info.*

URL: https://javascript.info/intro (дата звернення: 12.04.2022).

10. What is React.js?. *C-sharpcorner*.

URL: https://www.c-sharpcorner.com/article/what-and-why-reactjs/ (дата звернення: 12.04.2022).

11. What is a REST API?. *Redhat.*

URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (дата звернення: 12.04.2022).

12. What is web socket?. *Geeksforgeeks.*

URL: https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/ (дата звернення: 12.04.2022).

13. What is SSL?. *SSL.*

URL: https://www.ssl.com/faqs/faq-what-is-ssl/ (дата звернення: 12.04.2022).

14. MariaDB Server: The open source relational database.

URL: https://mariadb.org/ (дата звернення: 12.04.2022).

15. CI/CD - continuous integration. *Wikipedia.*

URL: https://en.wikipedia.org/wiki/CI/CD (дата звернення: 12.04.2022).

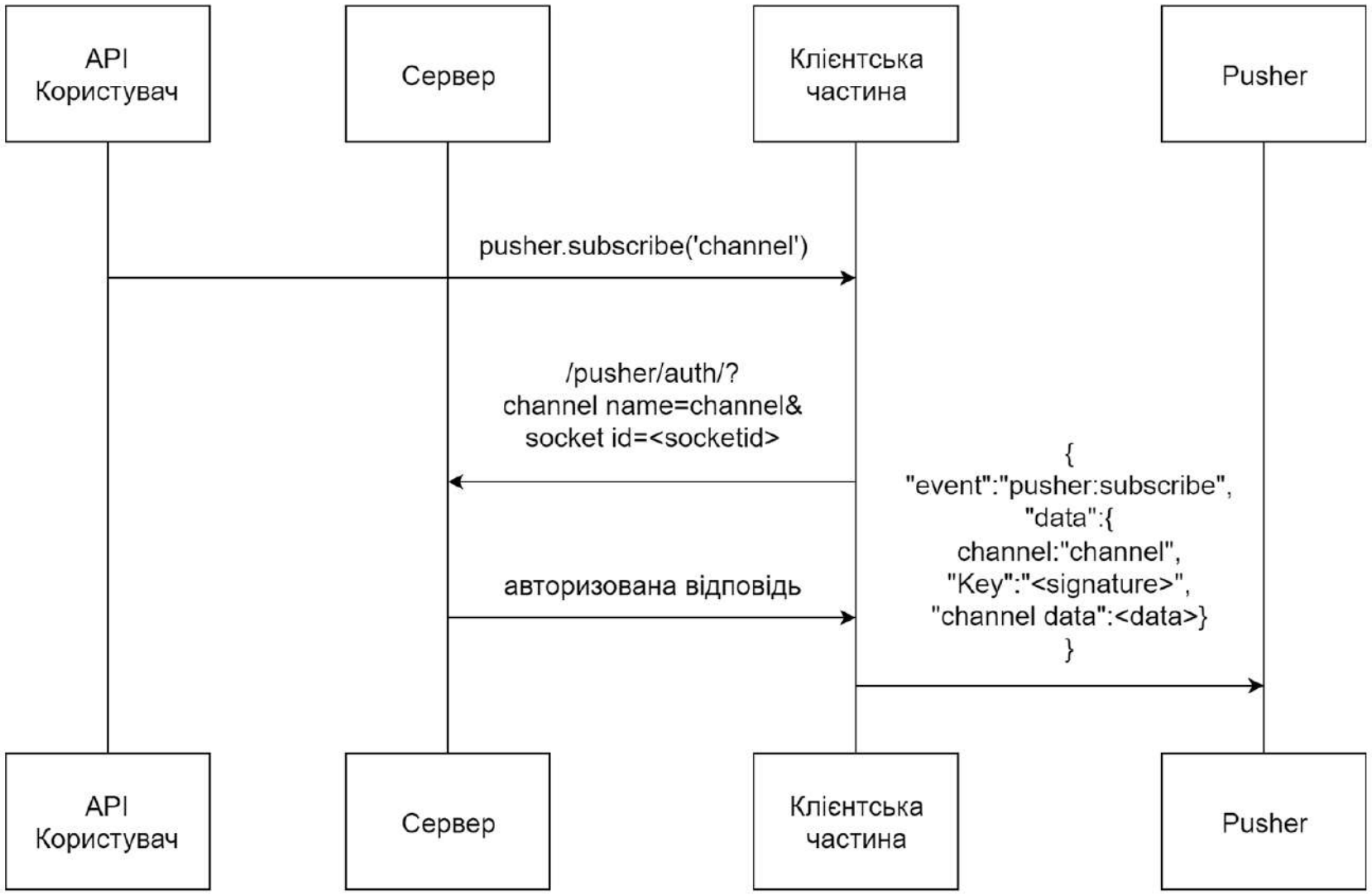
16. Поняття загроз інформаційній безпеці. *Pidru4niki.*

URL: https://pidru4niki.com/12800528/politologiya/ponyattya_zagroz_informatsiyniy_bezpetsi (дата звернення: 12.04.2022).

17. Види інформаційних загроз. *Google.*

URL: https://sites.google.com/site/vidiinformacijnihzagrozinform/ (дата звернення: 12.04.2022).

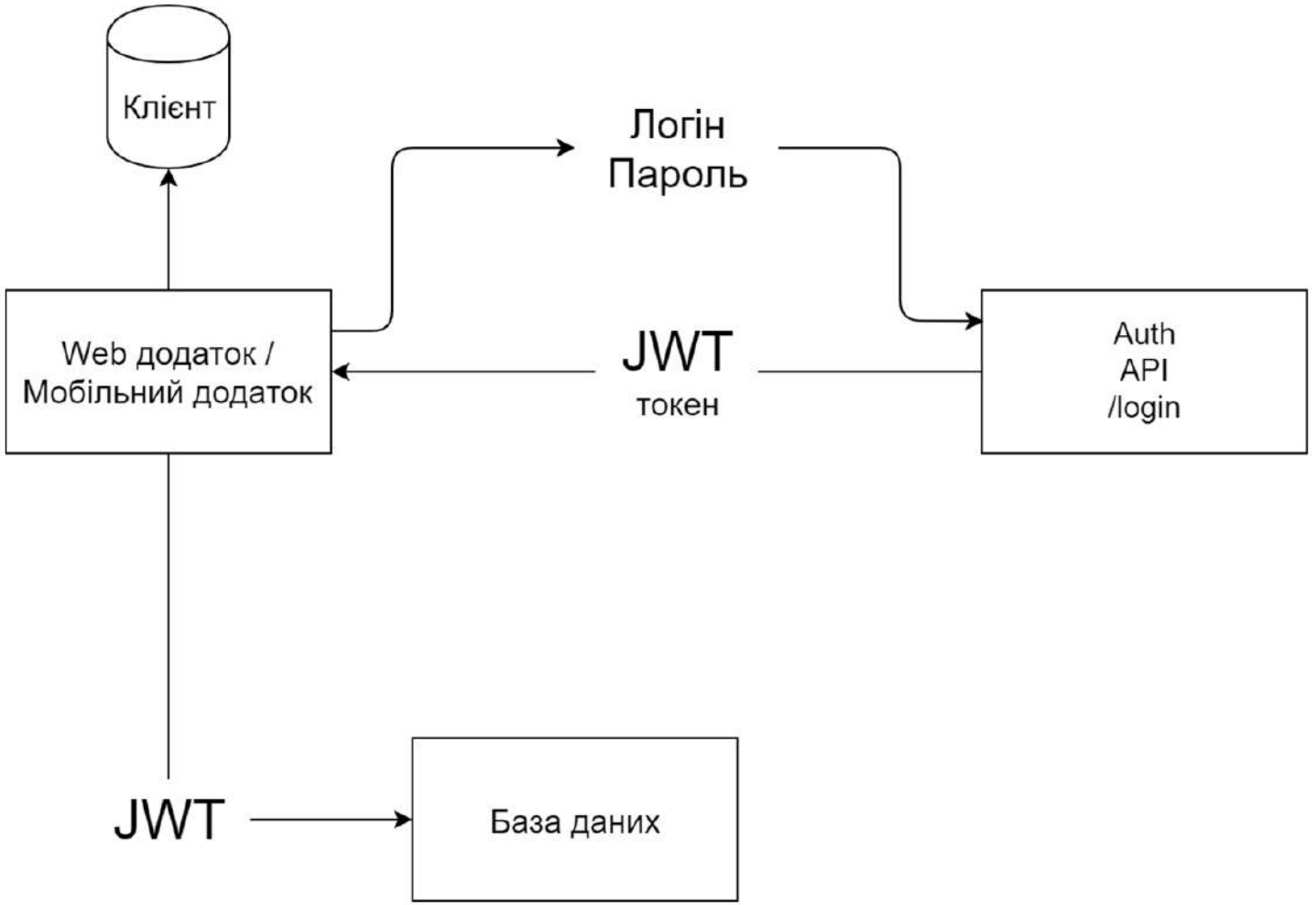
ДОДАТОК А
(обов’язковий)
Копія графічної частини



					КРКБ.180128.18.01.04 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Процес передачі даних за допомогою бібліотеки Pusher	Літ.	Маса	Масшт.
Розроб.		Київський В.О.				Н		
Перевір.		Чешун В. М.				Аркуш 1	Аркушів 4	
Т.контр.								
Н.контр.		Мостовий С.В.				КБ-18-1		
Затв.		Кльоц Ю.П.						

КРКБ.180128.18.01.04 Е8

Зм.	Арк.	№ докум.	Підпис	Дата	КРКБ.180128.18.01.04 Е8			
Розроб.		Катинський В.О.			Контекстна діаграма процесу розробки додатку	Літ.	Маса	Масшт.
Перевір.		Чешун І. М.				Н		
Т.контр.						Аркуш 2	Аркушів 4	
Н.контр.		Мостовий С.В.				КБ-18-1		
Затв.		Кльоц Ю.П.						



					КРКБ.180128.18.01.04 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Принцип авторизації за допомогою JWT токена	Літ.	Маса	Масшт.
Розроб.		Катаєвський В.О.				Н		
Перевір.		Чешун В. М.				Аркуш 3	Аркушів 4	
Т.контр.								
Н.контр.		Мостовий С.В.				КБ-18-1		
Затв.		Кльоц Ю.П.						

					КРКБ.180128.18.01.04 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Аналіз внутрішніх загроз	Літ.	Маса	Масшт.
Розроб.		Катаєвський В.О.				Н		
Перевір.		Чешун І. М.				Аркуш 4	Аркушів 4	
Т.контр.								
Н.контр.		Мостовий С.В.				КБ-18-1		
Затв.		Кльоц Ю.П.						

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Качинського Владислава Олеговича
ПІБ здобувача вищої освіти
студента ФІТ, 4 курсу, групи КБ-18-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та 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 та 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 Anti-Plagiarism) та використання роботи для виявлення 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.

6.06.2022
дата


підпис

UNICHECK

Ім'я користувача:
Кафедра кібербезпеки

ID перевірки:
1011428844

Дата перевірки:
02.06.2022 11:53:36 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2022 12:09:20 EEST

ID користувача:
100008300

Назва документа: **Диплом Качинський - без додатків**

Кількість сторінок: **62** Кількість слів: **11424** Кількість символів: **87102** Розмір файлу: **2.00 MB** ID файлу: **1011309101**

2.36%
Схожість

Найбільша схожість: **0.63%** з джерелом з Бібліотеки (ID файлу: **1011309100**)

1.49% Джерела з Інтернету	114	Сторінка 64
1.74% Джерела з Бібліотеки	93	Сторінка 65

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 1.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 12%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 104370 Название: Система захисту внутрішньої конфіденційної інформації кросплатформової системи компанії Hexide Digital Добавлено в БД: 2022-06-02 Авторы: Качинський Владислав Олегович Руководители: В. М. Чешун Консультанты: Оппоненты:	Символы	Лексемы	Символы	Лексемы
	66996	1038	1500 (2%)	25 (2%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:
Назва: Система захисту внутрішньої конфіденційної інформації кросплатформової системи Hexide Digital

Автор: Качинський Владислав Олегович
Спеціальність: 125 – Кібербезпека
Освітня програма: освітньо-професійна
Науковий керівник: Чешун Віктор Миколайович, к.т.н, доцент
Після аналізу звіту подібності 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 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 та електронної версії роботи	
3	Виявлені запозичення не є 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 досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою Unichek складає 97,64%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism v-15.257 складає 99%.

Згідно з Положенням про дотримання академічної доброчесності в Хмельницькому національному університеті (http://www.khnu.km.ua/root/files/01/10/03/0005.pdf) така авторська робота, обсяг оригінального тексту у відсотках до загального обсягу матеріалу в якій складає 90-100 %, визнається роботою з високою унікальністю тексту.

Керівник роботи Віктор Чешун

Гарант ОП Віктор Чешун

Завідувач кафедри Кб Юрій Кльоц

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Качинський Владислав Олегович

Тема Система захисту внутрішньої конфіденційної інформації кросплатформової системи компанії Hexide Digital

Спеціальність 125 – Кібербезпека

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 4; кількість сторінок записки 61.

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі розроблено систему для захисту конфіденційних даних компанії.

2. Висновок про відповідність кваліфікаційної роботи завданню Кваліфікаційна робота у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі подана загальна характеристика поставленої задачі, чітко визначено об'єкт, предмет та методи дослідження, сформульована актуальність. Визначені задачі, які необхідно вирішити для досягнення поставленої мети, практична цінність отриманих результатів, їхня новизна та наведені відомості про публікації. У першому розділі проведено огляд використовуваних в комп'ютерних системах методів захисту конфіденційної інформації, виконане обгрунтування актуальності теми дослідження і виконана постановка задачі. В другому розділі наведені засоби та технології використані для побудови системи захисту. В третьому розділі визначено основні положення системи та розроблено алгоритми її роботи. Четвертий розділ було присвячено саме розробці системи

4. Позитивні сторони роботи Кваліфікаційна робота має комплексну практичну цінність. Практична цінність полягає у розробці сервісу, який захищає конфіденційну інформації компанії та її клієнтів. На основі даного аналізу в подальшому здійснюється керуючий вплив на витік конфіденційних даних. Практична цінність результатів кваліфікаційної роботи полягає у створені системи захисту від витоків даних, що може бути підлаштована для будь якої категорії даних, і у описі оптимальних моделей функціонування систем захисту подібного характеру.

5. Негативні сторони роботи Розроблена система захисту від витоків даних досить чутлива до великих навантаження.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення виконане відповідно до теми кваліфікаційної роботи з дотриманням стандартів. В загальному графічне оформлення виконане якісно, пояснювальна записка відповідає нормам щодо її оформлення.

7. Відгук про роботу в цілому В загальному кваліфікаційна робота заслуговує позитивної оцінки. Весь матеріал кваліфікаційної роботи структурований, чіткий та послідовний. Усі розділи роботи послідовні та логічні, що дозволяє чітко розуміти викладений матеріал в рамках тематики кваліфікаційної роботи. Графічний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті за основу для досягнення поставленої мети.

8. Інші зауваження Окремі описи в пояснювальній записці подано занадто деталізовано, що ускладнює сприйняття матеріалу фахівцями в обраній предметній галузі.

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінку «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)

Лисенко Сергій Миколайович, д.т.н., професор
кафедра комп'ютерної інженерії та інформаційних систем, Хмельницький національний університет

« 6 » червня 2022. (підпис)