Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Галузь знань 12 - Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

на тему «Метод та технологія автоматизації роботи з рецептами»

КвРКІ. 160122.20.01.26 ПЗ

Виконав студент 2 курсу група КІ2м-20-1 _____ Рей К. С.
Підпис Ініціали, прізвище

Керівник доктор техн. наук, професор _____ Говорущенко Т. О.
Науковий ступінь, звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н, проф.

_____ Говорущенко Т. О.
Підпис Ініціали, прізвище

18 05 2022 р.

Хмельницький 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень МАГІСТР

Галузь знань 12 Інформаційні технології

Спеціальність 121 Комп'ютерна інженерія

Освітня програма Освітньо-наукова програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Т.О.Говорущенко
01 09 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ)

Рею Костянтину Сергійовичу

Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Метод та технологія автоматизації роботи з рецептами

Керівник проєкту (роботи) Говорущенко Т.О., д.т.н, професор

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

Затверджена наказом ректора університету від 06.01.2022 р. № 1

2. Строк подання студентом проєкту (роботи) на кафедру 03.05.2022 р.

3. Вихідні дані до проєкту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, Проектування технології автоматизації роботи з рецептами, Реалізація технології автоматизації роботи з рецептами, Тестування технології автоматизації роботи з рецептами

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди)

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «06» 09 2021 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних там ДП	05.09.2021	
2 Дослідження предметної області, визначення задач та вимог, розробка	05.10.2021	
3 Проектування програмного забезпечення	05.01.2022	
4 Програмна реалізація	15.02.2022	
5 Тестування програмного забезпечення	05.04.2022	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	15.04.2022	
7 Попередній захист ДРМ	18.04.2022	
9 Захист ДРМ на засіданні ЕК	До 10.05.2022	

Студент _____ К.С. Рей
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: «Метод та технологія автоматизації роботи з рецептами».

Автор роботи: Рей Костянтин Сергійович.

Керівник роботи: Говорущенко Тетяна Олександрівна.

Пояснювальна записка: 120 с., 15 рис., 6 табл., 10 дод., 32 джерела.

Графічна частина: 15 презентаційних слайдів.

СОЦІАЛЬНА МЕРЕЖА, REST, WEB API, ASP.NET CORE, REDIS, TYPESCRIPT, ANGULAR, КОМП'ЮТЕРНА СИСТЕМА АВТОМАТИЗАЦІЇ ПУБЛІКАЦІЇ ТА ПОШИРЕННЯ КУЛІНАРНИХ РЕЦЕПТІВ, «РОЗУМНИЙ БУДИНОК».

Об'єктом дослідження є процес автоматизації роботи з рецептами.

Предметом дослідження є метод та технологія автоматизації роботи з рецептами.

Метою роботи є підвищення ефективності та рівня автоматизації роботи з рецептами за рахунок інтеграції в реальний світ за допомогою фізичних отримувачів інформації з оточуючого світу.

Наукова новизна отриманих результатів:

1) набули подальшого розвитку метод та технологія автоматизації публікації, поширення та пошуку рецептів за рахунок використання фізичних давачів, що допомагають зменшити витрати часу на базові людські потреби, підвищити ефективність та скоротити час приготування їжі.

Практична цінність отриманих результатів. В результаті виконання кваліфікаційної роботи були розроблені метод та технологія автоматизації роботи з рецептами, а також методи налаштування системи «Розумний будинок» для синхронізації її із соціальною мережею та наташтування веб-серверу для аналізу даних, отриманих із реального світу.

За темою кваліфікаційної роботи опубліковано тези доповіді та взято участь у Всеукраїнській науково-практичній конференції «Актуальні проблеми комп'ютерних наук», що проходила 15-16 жовтня 2021 р. в Хмельницькому національному університеті:

1) Рей К., Ковтонюк І., Грищук І. Дослідження методів керування ресурсами кіберфізичної системи «Розумний будинок». Збірник наукових праць за матеріалами Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук» АПКН–2021 (Хмельницький, 15-16 жовтня 2021). С. 191-193.

ЗМІСТ

ВСТУП

Сьогодні Інтернет є потужним інструментом для пошуку та надання інформації. Джерелами інформації можуть бути звичайні веб-сайти, блоги, цільові сторінки тощо. Але найпопулярнішими сьогодні сервісами є соціальні мережі, де люди витрачають найбільше часу в інтернеті. Виникнення та розвиток соціальних мереж сприяє розвитку нової культури та суспільства в цілому. Цей спосіб спілкування виконує велику кількість функцій, дозволяє людині самореалізуватися, отримати нові знання та навички.

Соціальні мережі – це системи, які поєднують в собі інформаційно-розважальний контент та елементи реклами. Зараз користувач переважно не створює контент, а споживає його. З роками інформація все більше групується в категорії. Користувачам стає все важче вибрати з пропонованого контенту щось корисне і потрібне. Саме тому соціальні мережі, діяльність яких обмежена певною тематикою, починають набирати популярність. Користувачам пропонується лише той контент, який вони хочуть споживати. Хоча такі послуги стають дедалі популярнішими, вони не охоплюють деякі важливі та актуальні теми, як-от їжа та приготування їжі. Такого контенту в Інтернеті багато, але він зазвичай не структурований, що робить його не дуже зручним для споживання.

Актуальність теми полягає в тому, що сьогодні в Інтернеті існує велика кількість кулінарних сайтів, але повноцінних соціальних мереж з такою тематикою практично немає. Соціальні мережі є не тільки інструментом для обміну інформацією між користувачами, а й готовими платформами для ведення власного бізнесу. Крім того, приготування їжі є одним із пріоритетів більшості людей. І якщо ми об’єднаємо це, то отримаємо потужну систему, призначену для пошуку та обміну рецептами. Така система в першу чергу повинна бути швидкою та оптимізованою, а оскільки вона розрахована на велику кількість користувачів, то й високо завантажена. Існує багато технологій і практик для створення та обслуговування таких систем, але вибрати найкращу не завжди легко. Тому ми пропонуємо реалізувати таку програмну систему та на її основі проаналізувати та дослідити основні принципи оптимізації високонавантажених

систем.

Об’єктом дослідження є процес автоматизації роботи з рецептами.

Предметом дослідження є метод та технологія автоматизації роботи з рецептами.

Метою кваліфікаційної роботи є підвищення ефективності та рівня автоматизації роботи з рецептами за рахунок інтеграції в реальний світ за допомогою фізичних отримувачів інформації з оточуючого світу.

У кваліфікаційній роботі проведено аналіз соціальних мереж, було порівняно монолітний і мікросервісний стиль розробки, розглянуто методології та інструменти для високонавантаженого системного адміністрування та рекомендовані шляхи їх покращення, було розглянуто використання реляційних баз даних і баз даних NoSQL, а також їх переваги та недоліки. Характеристики front-end фреймворків були визначені під час побудови клієнтського компонента системи. Система напряму взаємодіє з «Розумним будинком», що дає можливість використовувати соціальну мережу у помешканні як єдиний механізм, який автоматизує пошук страв та буде синхронізований з процесами, які кожна людина виконує щодня.

Наукова новизна отриманих результатів:

1) набули подальшого розвитку метод та технологія автоматизації публікації, поширення та пошуку рецептів за рахунок використання фізичних давачів, що допомагають зменшити витрати часу на базові людські потреби, підвищити ефективність та скоротити час приготування їжі.

Практична цінність отриманих результатів. В результаті виконання кваліфікаційної роботи були розроблені метод та технологія автоматизації роботи з рецептами, а також методи налаштування системи «Розумний будинок» для синхронізації її із соціальною мережею та наташтування веб-серверу для аналізу даних, отриманих із реального світу.

Поставлена мета досягається розв’язанням таких основних задач:

1. аналіз особливості та специфіки предметної області соціальної мережі;

2. аналіз монолітної та мікросервісної архітектури;

3. аналіз методів та інструментів по управлінню високонавантаженими

системами та шляхи їх оптимізації;

4. аналіз використання реляційних, документо-орієнтованих та графових баз даних при реалізації програмної системи та реалізувати серверну частину програмної системи;

5. максимізувати зручність використання системи та налаштувати її для взаємодії із системою «Розумний будинок».

Результатом вирішення завдань кваліфікаційної роботи є повноцінна комп'ютерна система для автоматизації, публікації та поширення кулінарних рецептів, яка має інтеграцію з системою «Розумний будинок».

За темою кваліфікаційної роботи опубліковано тези доповіді та взято участь у Всеукраїнській науково-практичній конференції «Актуальні проблеми комп'ютерних наук», що проходила 15-16 жовтня 2021 р. в Хмельницькому національному університеті:

1) Рей К., Ковтонюк І., Грищук І. Дослідження методів керування ресурсами кіберфізичної системи «Розумний будинок». Збірник наукових праць за матеріалами Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук» АПКН–2021 (Хмельницький, 15-16 жовтня 2021). С. 191-193.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Соціальна мережа – це сервіс, який об’єднує людей, які мають спільні інтереси або хобі, і дозволяє їм обмінюватися інформацією один з одним. Сотні мільйонів людей зараз користуються сайтами соціальних мереж у всьому світі. Люди проводять з ними все більше часу: спілкуються з ними, шукають інформацію, консультуються, знайомляться з ними, будують з ними комерційні відносини. Крім того, ви можете створювати рекламу за допомогою сучасних соціальних мереж. Крім того, оскільки шанси на інтеграцію рекламних матеріалів у соціальні мережі зросли, новий тренд, відомий як SMM (маркетинг у соціальних мережах), розвивався як інструмент для ведення бізнесу.

Розрізняють такі соціальні мережі:

1) для спілкування, такі, як Facebook;

2) для поширення інформації медійного вмісту. Дані системи реалізовують широкий спектр можливостей для обміну відеоконтенту та фотографій. Прикладом таких мереж є Instagram та YouTube;

3) для поширення індивідуальних висловлювань. До даного виду відносяться сервіси для публікації текстово-медійного контенту. Наприклад, Twitter;

4) для колективних переговорів. В основі цього виду соціальних мереж лежить потреба в обміні знань. Приклади: Reddit, Stack Exchange.

Особливу увагу слід звернути на соціальні мережі які поєднують у собі інформацію вузького спрямування. До прикладу, соціальна мережа Goodreads дозволяє користувачам шукати та публікувати книги, анотації, цитати та огляди. Незважаючи на те, що вони популярні лише в певній галузі, вони повинні реалізовувати базову функціональність соціальних мереж, бути зручними для користувачів, а система має бути достатньо безпечними та продуктивними.

Розумний дім – це все, що ваш будинок може зробити для вас, щоб зробити

ваше життя простішим і продуктивнішим. Здається, що розумний дім використовує інтелект для досягнення цілей. Мікроконтролери проклали шлях для популярних крихітних платформ, які дозволяють великій кількості людей виконувати завдання, які в іншому випадку потребували б спеціалізованого та дорогого обладнання. Те, що колись вважалося неспроможною мрією, тепер стало реальністю завдяки розвитку технологій. Принцип однаковий в автоматизованому будинку або в розумному будинку. Система «Розумний дім» передусім дозволить користувачам контролювати та керувати побутовою технікою, поки вони відсутні. Система також може самостійно контролювати та видавати команди. Користувач має повний контроль над Інтернетом і голосовими командами за допомогою системи Smart Home, яка використовує Raspberry Pi як контролер. Щоб отримати доступ до підсистем по всьому будинку, Raspberry Pi підключається до плат Arduino. Технології бездротового зв'язку усувають проблеми з дротовими з'єднаннями та обмеженнями діапазону. Raspberry Pi та Arduino Smart Home – це проста у використанні та недорога система, якою може володіти кожен.

Під «Розумним будинком» слід розуміти систему, яка повинна вміти розпізнавати конкретні ситуації, що відбуваються в будівлі, і певним чином на них відповідати: системи мають змогу взаємодіяти між собою та змінювати свій стан за раніше прописаними сценаріями. Головною особливістю інтелектуальної будівлі є поєднання окремих підсистем в єдиний керований комплекс. Важливою особливістю та властивістю «Розумного дому», що відрізняє його від інших способів організації житлового простору, є те, що це найпрогресивніша концепція взаємодії людини з житловим простором, коли людина задає потрібне середовище однією командою, і автоматизація. встановлює і відповідно до зовнішніх і внутрішніх умов і контролює режими роботи всіх інженерних систем і електроприладів.

Кулінарні сайти за своєю реалізацією схожі на тематичні соціальні мережі. Ця тема популярна, оскільки їжа є однією з основних потреб людини, а її приготування може зайняти багато часу. Однак зазвичай вони реалізуються у вигляді авторських блогів і звичайних сайтів, що відрізняються один від одного

за своїм функціоналом і дизайном. Пошук певного розділу сайту, конкретного рецепту може зайняти значно більше часу більше часу, що є не зручним для звичайних користувачів.

Тому пропоную реалізувати компютерну систему для автоматизації публікації, розповсюдження та пошуку рецептів, інтерговану з системою «Розумний будинок», що буде проводити аналіз давачів у квартиві, статуси приладів, та на основі зібраної інформації та часових проміжків буде підбирати рекомендації по приготуванню страв, відсилати сповіщення про книги рецептів, нагадування про час приготування тощо. Варто зазаначити, що система сповіщень соціальної мережі буде максимально адаптована під користувача, оскільки він буде отримувати їх тільки тоді коли це буде справді потрібно. Тобто буде реалізовано можливість відслідковуання перебування у кімнаті, і при наявності на кухні особи, система буде надсилати сповіщення про рекомендації по приготування страв. Ці рекомендації будуть залежати також від часу доби та від вподобань користувача. Тобто, такий метод взаємодії між двома системами позбавить користувачів надокучливих сповіщень.

Комп’ютерна система реалізовує всім звичні функції типової соціальної мережі як оцінка публікацій, можливість ділитись ними та писати коментарі, можливість стежити за іншими користувачами. Оскільки ця соціальна мережа тематична, було вирішено впровадити систему модерації контенту, щоб уникнути шкідливих, непристойних чи некулінарних публікацій.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Сьогодні існує багато таких програмних систем, які дозволяють користувачам переглядати та публікувати рецепти, а отже соціальна мережа повинна бути продуктивною, функціональною, мати зрозумілий та сучасний дизайн, високу швидкодію.

Давайте подивимося на кілька популярних веб-сайтів рецептів і зробимо порівняльний опис. Основними елементами для оцінки є: функціональність,

продуктивність, зручність використання, привабливість користувальницького інтерфейсу.

На рисунку 1.1 зображено головну сторінку сайту [4]. На цьому сайті ви можете поділитися своїми рецептами з іншими користувачами. Деякі з них мають фотографії. Існує також примітивна функціональність для оцінки рецептів. Є зручне навігаційне меню, а категорії, які найчастіше обирають користувачі для перегляду, відображаються на головній сторінці сайту. Є можливість пошуку рецептів за назвою та описом.

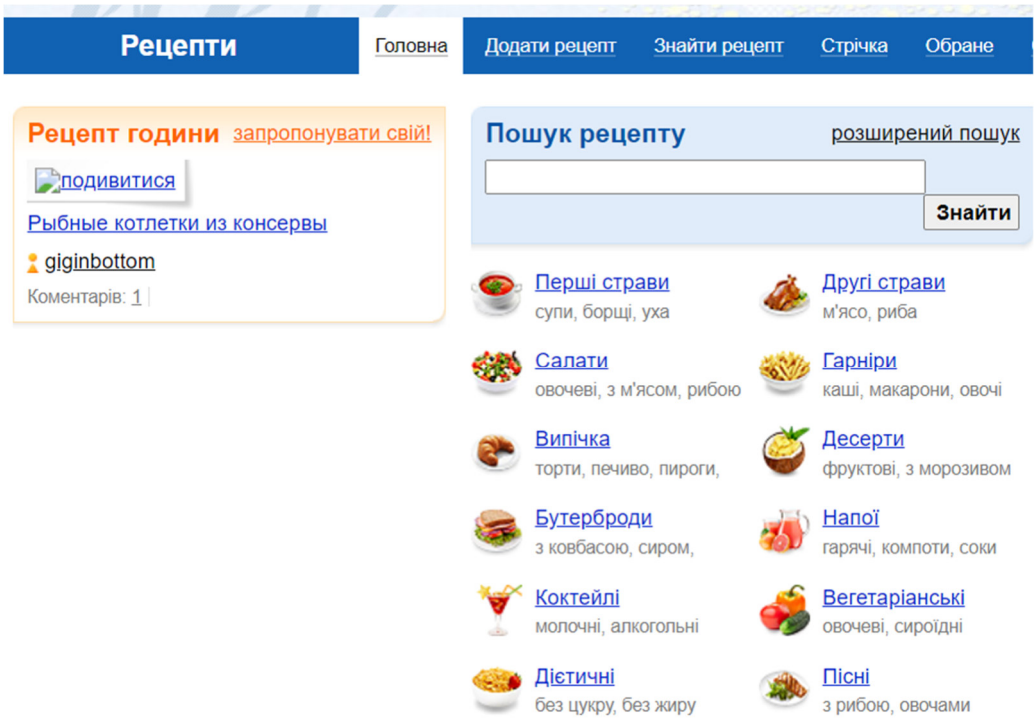


Рисунок 1.1 – Головна сторінка сайту [4]

Перш за все можна виокремити що дизайн системи є застарілим та не дружнім до користувача. Реалізовано пошук за назвою рецепту проте відсутній пошук за інгредієнтами. Сайт має достатню швидкодію, базові функції соціальної мережі також присутні.

На рисунку 1.2 зображено сторіноку сайту «Simply Recipes» [5]. Інтерфейс блогу виглядає мінімалістично та сучасно. Наявна можливість зберігати рецепти, шукати та фільтрувати. Крім того, переглядати необхідні рецепти страв можна за допомогою пошуку по назві інгредієнтів. Крім того, ви можете переглянути

необхідні рецепти, здійснивши пошук за назвою інгредієнтів. Доступні відео-рецепти та докладні інструкції з приготування страв. Вхід на сайт реалізовано через соціальні мережі, відповідно немає налаштувань особистого облікового запису. Звертаю увагу що це все ще не соціальна мережа, а блог, на якому можливість публікувати рецепти має тільки його автор.

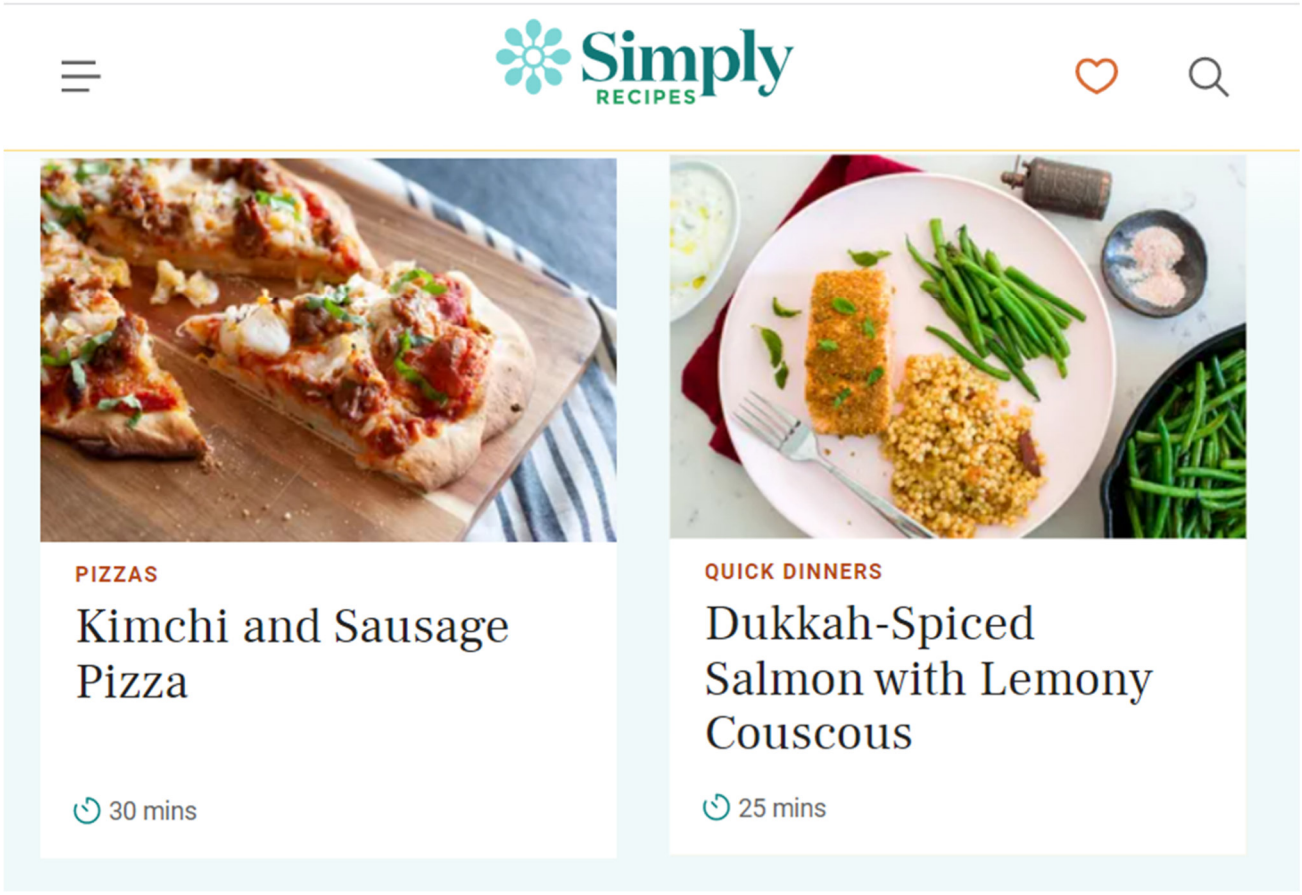


Рисунок 1.2 – Сторінка сайту «Simply Recipes»

Вхід та реєстрація у систему відбувається за рахунок електронної пошти або через інші сервіси. Є можливість налаштування особистого кабінету користувача. Суттєвим недоліком є те, що вказана система націлена на мобільні пристрої, а отже веб версія має обмежені фунції.

Отже, на основі проведеного аналізу вищевказаних ресурсів та аналізу функційних та нефункційних вимог до сучасної тематичної соціальної мережі, було скласти порівняльну таблицю 1.1 працюючих ресурсів, які зодовільняють деяким критеріям розроблюваної системи.

Отже, комп’ютерна система повинна мати:

1) можливість публікації та редагування рецептів;
2) зручний пошук рецептів, їх фільтрацію по категоріями;
3) сучасний, інтуїтивно зрозумілий дизайн інтерфейсу;
4) можливості та функції сучасних соцмереж;
5) зручну форму авторизації та реєстрації;
6) налаштування особистого кабінету.

Таблиця 1.1 – Порівняльна характеристика наявного ПЗ

Назва або посилання	Можливість публікації рецептів	Зручний пошук рецептів	Дизайн користувацького інтерфейсу	Реєстрація та функціонал особистого кабінету
http://cook.i.ua/	Так	Ні	Застарілий, мінімалістичний, інтуїтивно зрозумілий	Через логін та пароль, є редагування особистого кабінету
«Simply Recipes»	Ні	В цілому зручний	Сучасний, інтуїтивно зрозумілий	Тільки через соцмережі, немає можливості редагування особистих даних

Також слід зазначити що існуючі рішення є веб-додатками які не мають змоги бути інтегровані із іншими системами чи різного роду давачами.

1.3 Визначення вимог до програмного забезпечення та розробка технічного завдання

Головною вимогою до компютерної системи є непомітність в роботі двох головних компонентів системи, а саме клієнтського додатку та системи «Розумний будинок». Тобто це означає, що користувач не повинен звертати увагу на ті чи інші події, тобто всі команди та події мають виникати саме в той час коли це дійсно необхідно.

Загальне представлення функцій веб-додатку можна зобразити на діаграмі варіантів використання. Вона описує конкретні функції системи. У ній

преставлені дійові особи, варіанти використання системи та чіткі звязки між ними.

У таблиці 1.2 наведено характеристику акторів системи.

Таблиця 1.2 – Опис акторів системи

Актор	Короткий опис
Незареєстрований користувач (гість)	Може дивитись та має можливість пошуку страв.
Зареєстрований користувач	Має можливість писати та змінювати повідомлення, оцінювати, зберігати, коментувати дописи інших людей, підписуватися на інших користувачів та керувати своїм обліковим записом, на додаток до повноважень незареєстрованого користувача. Налаштування взаємодії системи «Розумний будинок» та соціальної мережі, додавання подій, на які буде реагувати соціальна мережа а також подій у соціальній мережі, на які буде реагувати «Розумний будинок».
Адміністратор та модератор	Може додавати нові частини до системи програмного забезпечення, контролювати дописи користувача (тобто перевіряти наявність цензури та інших обмежень відомих соціальних мереж), а також блокувати та видаляти облікові записи користувачів.

Отже, у системі є три головних актори: гість, зареєстрований користувач та модератор. Кожен з них має свої права, які обмежені відповідними ролями, та має можливість взаємодіяти із системою як на програмному рівні, тобто дії всередині системи, які також поділяються на рівні доступу, так і на фізичному рівні, тобто налаштування взаємодії двох різних систем.

У таблиці 1.3 описано головні варіанти використання системи.

Таблиця 1.3 – Опис варіантів використання системи

Актор	Найменування ВВ	Опис ВВ
1	2	3
Незареєстрований користувач	Реєстрація в системі	Користувач може зареєструватися в системі за допомогою своєї електронної пошти та паролю
	Пошук та перегляд інформації	Користувач може шукати та переглядати публікації
Зареєстрований користувач	Керування обліковим записом	Користувач може редагувати облікові дані особистого кабінету
	Авторизація в системі	Користувач може авторизувати в системі, використовуючи свою електронну пошту
	Публікація рецептів та керування ними	Користувач може публікувати свої рецепти, редагувати та видаляти опубліковані рецепти.
Зареєстрований користувач	Оцінювання, коментування, додавання в збережені	Користувач може оцінювати рецепти інших користувачів, коментувати їх та зберігати у свою книгу рецептів
Адміністратор та модератор	Модерація користувацького контенту	Адміністратор може не допустити пост до публікації у випадку нецензурного контенту, контенту, що належить іншим користувачам або контенту, що не відповідає кулінарній тематиці
	Блокування та видалення користувацького облікового запису	У випадку підозрілих дій з боку користувацького облікового запису адміністратор може заблокувати акаунт на певний термін або видалити його

1.4 Висновки

Таким чином, було проаналізовано предметну область та з’ясовано, що соціальні мережі є популярними веб-сервісами, які займають значне місце у житті сучасної людини, забезпечують спілкування між людьми, виступають майданчиком для ведення бізнесу та об’єднують людей зі спільними інтересами. Головними вимогами для них є висока швидкодія, безпека та простота у використанні.

Аналізуючи існуюче програмне забезпеченя було визначено, що є схожі ресурси які мають відмінності у зовнішньому вигляді, функціях, зручності та зрозумілоті користувацького інтерфейсу, функційні та нефункційні вимоги та проведено детальний опис варіантів використання системи.

2 ПРОЕКТУВАННЯ ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ РОБОТИ З РЕЦЕПТАМИ

2.1 Архітектурне проектування

2.1.1 Аналіз клієнт-серверної архітектури сучасних веб-систем

Всі веб-додатки та будь-які Інтернет-сервіси побудовані за принципом архітектури «Клієнт-сервер» [6]. Ця архітектура заснована на двох основних складових: клієнті і сервері. Клієнт – це зазвичай програмне забезпечення, яке має можливість надсилати запити на віддалений комп'ютер-сервер. Ним також можна вважати користувача, якому необхідно використовувати відповідне програмне та апаратне забезпечення для доступу до певної інформації. Сервер - більш потужне обладнання, призначене для вирішення різноманітних завдань, таких як доступ до даних, запитуваних клієнтом, проведення обчислень ріхзного рівня складності тощо. Однак завдання сервера одне – правильно обробити дані користувача і надіслати коректно сформовану відповідь [7].

Таким чином, архітектура «Клієнт-сервер» полягає в тому, що клієнт надсилає запит на віддалений сервер, де він певним чином виконується, а готовий результат виконання надсилається клієнту у відповідь. Сервер має змогу обробляти паралельно велику кількість клієнтських запитів. Для виконання запитів, які були надіслано одночасно, присутня черга, завдяки якій вони виконуються один за одним.

Для того, щоб сервер і клієнт могли правильно обробляти вхідні та формувати вихідні дані, зв’язок між ними здійснюється за допомогою протоколу передачі даних – HTTP (Hyper Text Transfer Protocol). Це протокол передачі гіпертексту, тобто HTML (Hyper Text Markup Language). Однак сьогодні через швидкий розвиток серверних API – наборів чітко визначених методів взаємодії компонентів сервера – все частіше використовують формат даних, відмінний від гіпертексту [8].

У зв’язку з популярністю API серверів виникли й розвиваються різноманітні архітектурні підходи до створення інтерфейсів прикладного програмного забезпечення та управління ними. Найпопулярнішим з них є REST

(Representational State Transfer). Значущою особливістю служб REST є можливість найкраще використовувати HTTP. Обробка запиту в такій архітектурі повинна оброблятись відповідно до методу запиту, яким були відправлені дані. Він описує конкретну дію, яка має бути виконана над даними. Тобто, метод, який відповідає за отримання даних не повинен видаляти чи оновлювати інформацію. Однією з вимог RESTful архітектури є повернення клієнту статус-кодів, які дають розуміння того, як виконався запит сервером.

Беручи до уваги той факт, що на даний момент системи розумних будинків мають широкий спектр можливостей, є бездротовими та мають власні вмотновані механізми взаємодії зі сторонніми системами то веб-базований метод взаємодії між такою системою та веб-сервером додатку підходить найкраще.

На рисунку 2.1 зображено архітектуру компютерної системи автоматизації публікації.

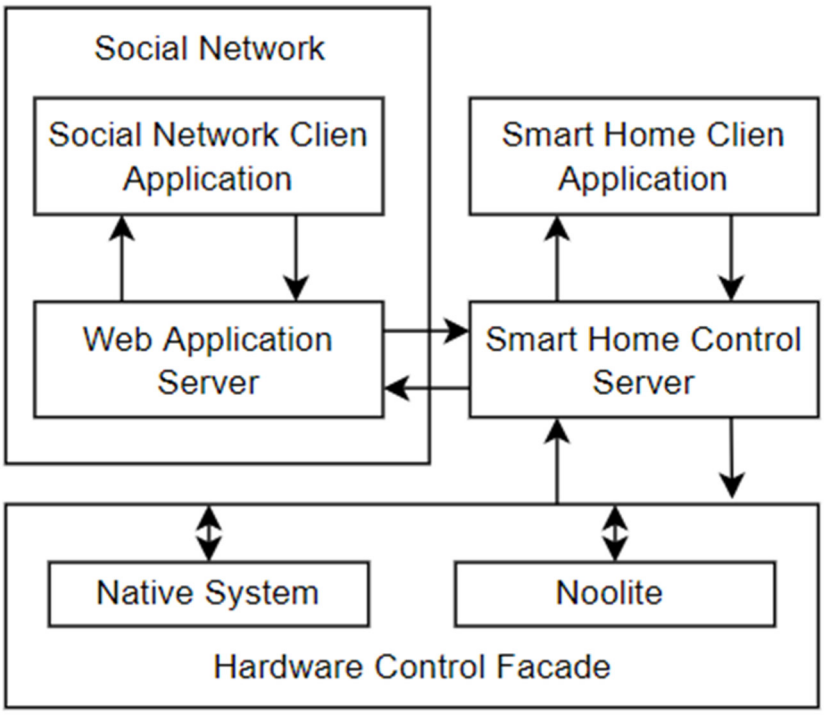


Рисунок 2.1 – Архітектура компютерної системи автоматизації публікації.

Модуль Smart Home Clien Application взаємодіє з Smart Home Control Server за допомогою REST API по протоколу HTTP. Smart Home Control Server і Hardware Control Modules також взаємодіють між собою за допомогою REST API

по протоколу HTTP. Вони можуть знаходиться як на одному фізичному сервері так і на різних.

Як видно з рисунку, модуль Web Application Server соціальної мережі також взаємоємодіє з модулем Smart Home Control Server системи «Розумний будинок» за допомогою REST API по протоколу HTTP, тобто це означає, що системи можуть існувати окремо одна від ондої, або бути зкомпоновані в одним механізм через серверну взаємодію.

Smart Home Control Server містить у собі бізнес логіку системи «Розумний дім».

Дана підсистема є сполучною ланкою між фізичними пристроями «розумного дому» та користувацьким додатком. Вона відповідає за моніторинг стану систем життєзабезпечення будинку, відмовостійкість системи, та безпеку системи.

Smart Home Control Server складається з наступних компонентів. User Request Engine – модуль, який відповідає за обмін даних з користувацьким додатком (обробка запитів користувацьких додатків та надсилання відповіді користувацькому додатку).

Security service – модуль, який забезпечує авторизацію користувача у системі «розумний дім».

Planning service – модуль, який відповідає за планування подій у системі «розумний дім» (ввімкнення/вимкнення опалення, ввімкнення/вимкнення електроприладів у запланований час). Operation service – модуль, що містить у собі бізнес логіку для роботи з API фізичних пристроїв системи «Розумний дім».

Monitoring service – модуль, який веде моніторинг роботи системи і виконує логування станів системи. Також даний модуль відповідає за відмовостійкість системи.

На рисунку 2.2 зображено структуру компонентів Smart Home Control Server.

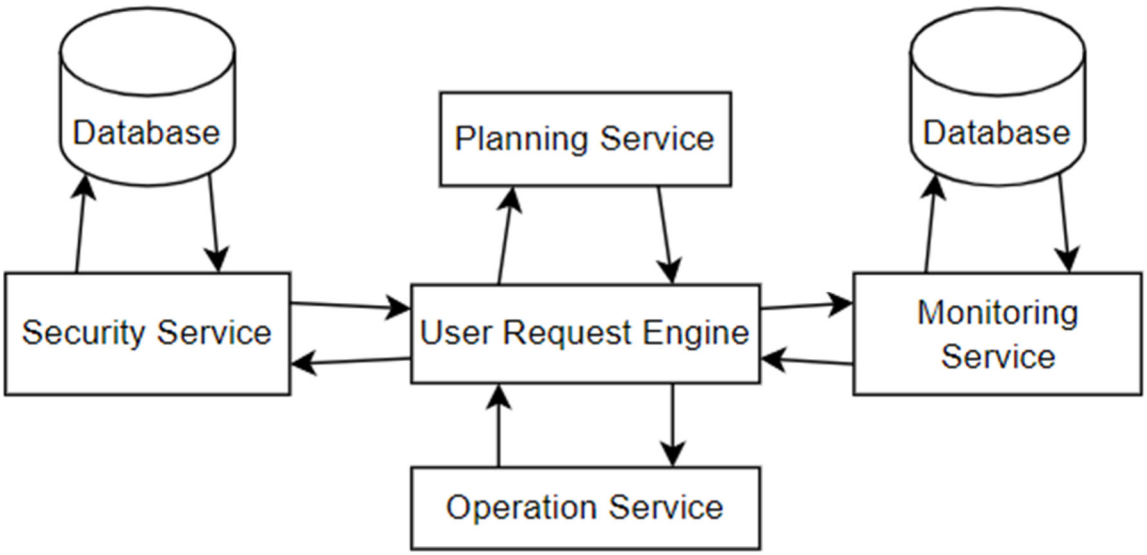


Рисунок 2.2 – Структурна компонентів Smart Home Control Server.

Система має двосторонню взаємодію між сервером соціальної мережі та між сервером розумного будинку.

Тобто, кінцевий користувач може змінювати стан приладів через клієнтський додаток з рецептами і навпаки, система «Розумний будинок» може взаємодіяти із соціальною мережею.

Алгоритм виконання дії для системи «Розумний дім» полягає в наступному: відбувається перевірка доступності вибраного фізичного пристрою, відбувається надсилання запиту з параметрами, що змінюють стан вибраного пристрою відповідно до вибору користувача.

Алгоритм зображено на рисунку 2.3.

Повідомлення у випадку компютерної системи являють собою статус-коди про успішність виконання операції.

Соціальна мережа включає рецепти із часом приготування. Тобто, обравши страву, відправиться запит до системи «Розумний будинок», та автоматично встановиться таймер на приладі приготування.

За ці функції відповідає плануальний дій системи «Розумний будинок».

Схема роботи алгоритму планувальника дій зображена на рисунку 2.4.

Рисунок 2.3 – Алгоритм операції виконання дії.

Рисунок 2.4 – Схема роботи алгоритму планувальника дій.

Тому для досягнення мети даної дипломної роботи за основу побудови взаємодії між віддаленими системами було обрано архітектуру «Клієнт-сервер».

Також пропонується використовувати певні прийоми архітектурного шаблону REST, які можуть збільшити ефективність застосування протоколу HTTP.

2.1.2 Аналіз методу проектування серверної архітектури

У наш час є багато способів реалізувати архітектуру сучасного веб-серверу. З усіх них можна виокремити два найбільш вживаних – це монолітний та мікросервісний. Кожний з них застосовується для конкретних випадків, оскільки має переваги в тій чи іншій ситуації [10].

Традиційним підходом до реалізації веб-додатків являється монолітний каркас архітектури. Сервер , що реалізований за таким підходом є однією цілою системою, що складається з чотирьох основних рівнів. Усі рівні функціонують у одному сервісі.

Рівень бізнес-логіки описує моделі предметної області. Найчастіше це набір класів, які повязані між собою певним типом відношення (ієрархія, агрегація тощо). Модель повинна описувати обєкт реального світу та його призначення та є однією з найважливіших компонентів системи.

Один з найважливіших рівнів – доступ до сховища інформації. На даному рівні проходить робота із базами даних та реалізуються основні операції з ними. Їх скорочено називають CRUD. Цей термін означає призначення кожної із цих операцій: Create – додає інформацію у базу, Read – читає дані, Update – оновлює, Delete – видаляє дані.

Рівень представлення являє собою користувацький додаток, який обробляє запити протоколу HTTP. Він може бути створений зі статично або динамічно створених наборів сторінок, що є різними частинами програми. Користувацький інтерфейс скорочено прийнято називати UI (User Interface).

При проектуванні монолітної серверної системи рівень інтеграції грає незначну роль, проте він є невідємною частиною при реалізації взаємодії з системами поза монолітним блоком. Цей рівень забезпечує загальні правила для управління даними вбудованих веб-сервісів. Зазвичай це робиться за допомогою різних адаптерів, які перетворюють представлення даних інтегрованого сервісу в подання основного і навпаки.

Важливою перевагою моноліту є незначний обєм наскрізних проблем. Монолітній програмі значно простіше вирішують такі проблеми завдяки єдиній

кодовій базі. Тому такі наскрізні тести легко виконати. Важливо також відзначити, що моноліт легко розгортати і легко масштабується завдяки єдиній кодовій базі.

Однак є значні недоліки. Один з головних – з часом база коду збільшується і стає все важче знайти проблемні місця, які в свою чергу можуть істотно вплинути на тривалість реалізації. У міру зростання бази коду гнучкість поступово втрачається, тобто для кожного невеликого оновлення потрібно повторне розгортання системи. Стає все складніше додавати нові технології, так як всі модулі в такій архітектурі тісно пов’язані один з одним, і розрив таких зв’язків може призвести до нових потенційних проблем і дефектів, що, вимагає від розробників вітрачати більше часу на написання тестів, що значно збільшує витрати часу на тестування системи.

Ще один стиль проектування серверної архітектури – мікросервіси. Це варіант сервіс-орієнтованої архітектури (SOA) [9], в якому ізольовані компоненти (сервіси) слабко пов'язані та взаємодіють один з одним для спільного виконання завдань, проте кожен сервіс у такій системі повинен мати можливість незалежно надавати різні послуги та взаємодіяти з одним із цих елементів. Використовуючи методи інтерфейсу прикладного програмування (API). На відміну від монолітів мікросервіси мають кілька одиниць розгортання – кожен сервіс розгортається незалежно.

Сервісний модуль у даній системі являється невеликим монолітом, який також має багаторівневу архітектуру, представлену чотирма основними рівнями. Рівень інтеграції в таій системі є обов'язковий і грає більшу роль, ніж у монолітної архітектурі, оскільки зазвичай сервіси взаємодіють друг з одним. До того ж, вони можуть бути реалізовані з використанням різних технологій, сховищ даних, шаблонів проектування і т.д.

Однією з переваг мікросервісів є їхній незначний розмір. Це допомагає скоротити час компіляції, запуску та тестування, оскільки всі ці фактори впливають на продуктивність та час розробки. Мікросервіси не залежать від технологій, які використовуються в інших модулях. В свою чергу це призводить до того, що можна використовувати більш сучасні технології, а старі можна

швидко і легко переписати за допомогою нових та продуктивніших.

Оскільки мікросервіси зазвичай є невеликими незалежними модулями, їх легше перевірити, ніж монолітні програми. Крім того, якщо є якісь недоліки, вони просто залишаться в тому ж мікросервісі. Більшість інших служб продовжуватимуть виконувати свої функції без дефектів. У той же час мікросервіси стійкі до збоїв, що також є великою перевагою.

Однак є і недоліки. Найчастіше це відбувається через транзакції. Транзакція – це логічна одиниця роботи, яка обробляє дані. Це можна зробити досить успішно, дотримуючись цілісності даних, або не зробити зовсім. Якщо сталася помилка, транзакція недійсна. У мікросервісах підтримання цілісності даних між різними модулями не є тривіальним завданням, тому транзакції найпростіше виконувати в моноліті. Мікросервісне рішення – використання розподілених транзакцій. Це досить громіздке, але ефективне рішення. Продуктивність мікросервісів може бути знижена через їх великий розмір, на відміну від монолітних додатків.

Ще одним недоліком мікросервісів є складність їх тестування. Він виражається в тому, що спочатку потрібно протестувати кожний сервіс, а тоді перевірити правильність їх взаємодії.

Таким чином монолітна архітектура найкраще підходить для розробки невеликих і простих систем. Мікросервісна архітектура, зі свого боку, призначена для розробки складних і розподілених рішень. З огляду на те, порівнюючи ці два стилі, не можна відразу сказати, що один кращий за інший. Стиль архітектури слід вибирати, виходячи з досвіду, часу розробки та складності розроблюваної системи.

Соціальні мережі, як правило, є великими, складними системами з багатьма сервісами та компонентами. Монолітний дизайн не підходить для таких систем, оскільки додавання функціональних можливостей розширює базу коду, створює додаткові з’єднання та ускладнює керування, тестування, масштабування та розгортання програмної системи. У більшості випадків, однак, новий продукт на ринку не може бути повністю проданий відразу. MVP (мінімально життєздатний продукт) – продукт з мінімальною функціональністю,

який може задовольнити запити потенційних споживачів – це термін для таких програм. Ці системи створюються за допомогою ітераційного процесу, що означає, що функціональність постійно розширюється за допомогою оновлень. Враховуючи короткий час розробки та мінімальну архітектурну складність системи, для таких додатків була обрана монолітна серверна архітектура.

2.2 Детальне проектування

Існує два основних типи веб-сервісів для впровадження програмної системи: Process-Based – основою є процеси і Event-Based – основою є події.

У веб-сервісах процеси використовують будь-який запит для обробки одним потоком або процесом. Кожен потік або процес вимагає певної кількості серверних джерел (апаратних засобів). Ресурси сервера не працюють і не застосовуються до відкладеного запиту та надсилання результату клієнту. Це погано позначається на швидкодії при високих навантаженнях, коли вибраного потоку процесів не вистачає для опрацювання всіх запитів. До них відносяться такі веб-сервіси, як IIS і Apache, що не є вдалим рішенням для таких додатків як соціальні мережі.

Веб-служби використовують ресурси на основі подій з усього серверного обладнання. Цикл контролює ранг запиту від користувача. Потім використовуються всі потужності сервера, які дозволяють максимально швидко обробляти запити, а у випадку виникнення затримок – перемикатись на інші запити з черги (Event Queue). До таких веб-сервісів відносяться платформи Node.js і Nginx. Дані рішення є хорошим варіантом для реалізації систем із високою навантаженістю, тому для розробки наданої програмної системи було прийнято рішення використовувати веб-сервіс Nginx.

Враховуючи зручність, безперервний розвиток, постійну чудову підтримку Microsoft і широкий спектр сторонніх бібліотек, фреймворків і модулів, об'єктно-орієнтована статично типізована мова програмування C# і ASP.NET Core ідеально підходить для реалізації цільового дождатку.

Серверна частина програмної системи побудована з використанням модифікованої конструкції моделі MVC (Model-View-Controller), де замість представлення використовуються фронтенд-системи, розгорнуті як окремий блок, відповідальний лише за відображення інтерфейсу користувача. Вони можуть не бути пов’язані з системою обслуговування (також званою Web API), але можуть бути розгорнуті на основі її можливостей.

Відповідно до монолітного архітектурного жанру, система складається з 3 взаємопов’язаних шарів: рівня даних, рівня сервісу та рівня презентації.

Рівень даних – шар опису даних. Він відповідає за організацію інформації та управління базами даних на рівні запитів. Здебільшого класи, які існують на цьому рівні, називаються контекстними. Цей рівень містить основні моделі реального світу, що чітко описують предметну область.

Модулі керування даними реалізовані за допомогою базових операцій з даними – CRUD – на рівні сервісу, який є рівнем абстракції програми. Рівень сервісу підключається до наступного рівня презентації за допомогою шаблонів проектування адаптера та команд для передачі оброблених даних на контролери.

Найвищим рівнем монолітної системи є рівень презентації. Працюючи як повноцінний інтерфейс прикладного програмування, він обмінюється даними з сервісним рівнем і використовує їх для показу користувачеві шляхом створення веб-сторінок або надсилання запитів REST клієнту (Web API).

На рисунку 2.5 зображено концепцію тришарової монолітної архітектури.

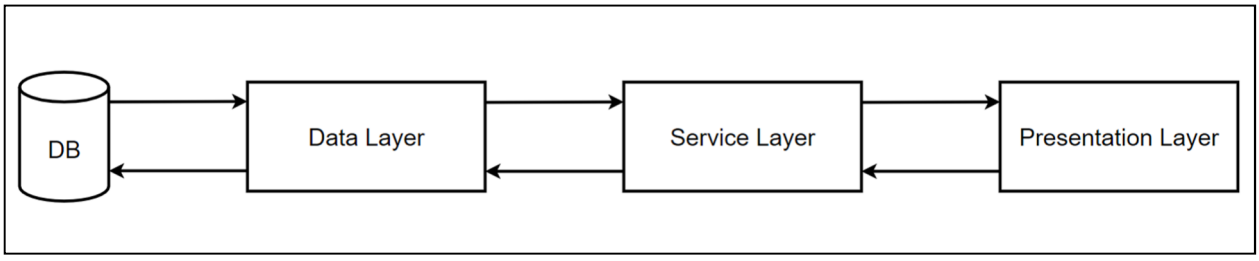


Рисунок 2.5 – Монолітна архітектура програмної системи

Кожен рівень складається з набору класів, які виконують певні функції на основі рівня розташування. Усі ці класи обмінюються даними та спілкуються один з одним.

2.2.1. Аналіз та вибір типу бази даних

Усі онлайн-додатки працюють з даними, які необхідно обробляти належним чином, а також безпечно зберігати. Для цього використовуються різні методи зберігання даних, відомі як бази даних. Точний і збалансований вибір сховища даних є одним із найважливіших питань, які необхідно вирішити при створенні програмної системи, оскільки це впливає на швидкість програми, час розробки, методи масштабування, підтримку тощо.

Сьогодні існує безліч рішень для управління даними. Реляційні бази даних, бази даних NoSQL та інтегровані програмні рішення – це три основні категорії.

Бази даних зі зв'язками [11]. Через широке використання реляційних моделей в системах управління базами даних цей тип зустрічається досить часто. Дані організовані у серію пов’язаних таблиць, кожна з яких містить інформацію про певний тип об’єкта. Рядок таблиці містить інформацію про один елемент, а стовпці таблиці визначають численні якості цих об’єктів. Структура записів однакова: вони складаються з полів, які зберігають атрибути об’єкта. Кожне поле має певний тип даних і описує лише одну властивість об’єкта. Усі записи мають ідентичні поля, але вони відображають різні інформаційні атрибути об’єкта.

Усі сутності у базах даних вимагають наявності унікального ідентифікатора, яке є полем або набором полів, які однозначно визначають кожен рядок у таблиці.

Однією з найважливіших переваг реляційних баз даних є те, що вони дозволяють користувачам швидко класифікувати та зберігати дані, які згодом можна аналізувати для отримання точної інформації. Реляційні бази даних також легко розширюються і на них не впливає фізична структура. Для доступу до реляційної бази даних зазвичай використовуються спеціальні програми, відомі як системи керування базами даних (СКБД). Це ще одна перевага цієї форми бази даних, оскільки СКБД зазвичай мають багато можливостей і надають базі даних додаткові функції, такі як індекси, очищення таблиці, підтримка JSON і власні принципи впорядкування даних, які називаються механізмами БД. Крім того,

реляційні бази даних мають чітко визначену структуру, тобто їх проектування здійснюється завчасно, що виключає невизначеність архітектурних рішень у майбутньому та дає повну картину сутності предметної області та їх взаємодій.

Крім того, забезпечення відповідності реляційних баз даних стандартам ACID (атомарність, цілісність, ізоляція та довговічність) допомагає обмежити випадки виникнення несподіваної поведінки системи та забезпечити її цілісність.

З іншого боку, у міру збільшення розмірів даних реляційні бази даних стають більш складними, що впливає на продуктивність програми.

Ідеальним рішенням є використання реляційної бази даних як основного сховища системи, оскільки програмне забезпечення для автоматизації публікації, розповсюдження та пошуку рецептів має чітко визначену суть предметної області та є продуктом MVP.

Бази даних NoSQL [12] – це набір типів баз даних, які виходять за рамки традиційної реляційної моделі. Ці бази даних розроблені спеціально для конкретних моделей даних і мають гнучкі схеми, що дозволяють швидко впроваджувати нові системи. Завдяки простоті побудови, функціональності та продуктивності в будь-якому масштабі бази даних NoSQL стали популярними. З іншого боку, нереляційні бази даних втрачають сто відсотків цілісності даних через свою гнучкість і високу продуктивність, тому їх слід використовувати як доповнення до реляційних баз даних. Такий спосіб використання не тільки підтримує цілісність даних РБД, але також може покращити продуктивність системи. В якості рівня кешування програми часто використовуються бази даних NoSQL, зокрема прості за структурою репозиторії «ключ-значення».

Графіки бази даних також доступні в сховищах NoSQL. Вузли використовуються для утримання сутностей, тоді як ребра використовуються для збереження зв’язків між таблицями. Кожне ребро має початкову і кінцеву вершину, а також тип і напрямок. Кількість і тип зв’язків, які може мати вершина, не обмежені.

У базі даних графів обхід графа можна здійснити на певних видах ребер або на всій мережі [13]. Оскільки зв’язки між вершинами зберігаються в базі даних, а не обчислюються під час запиту, вони виконуються швидко. Коли вам

потрібно створити зв’язки між даними та швидко проаналізувати їх, графічні бази даних мають багато переваг у таких системах, як реферальні служби та системи виявлення шахрайства. З іншого боку, графічні бази даних важко масштабувати через їх однорівневу архітектуру. У цьому дизайні клієнт, сервер і база даних знаходяться в одній системі. Прикладами є Neo4J, OrientDb, Neptune, ArangoDb та інші графічні бази даних.

Соціальна мережа – це програмна система, яка автоматизує публікацію, розповсюдження та пошук рецептів, тому найкраще зберігати зв’язки між користувачами та підписниками та інші подібні зв’язки у зв’язаному цільовому графі. Однак більшість функцій системи не потребує використання даних, які можна швидко обробити, зберігаючи їх у графі або дереві. Крім того, використання іншого типу бази даних значно ускладнить систему, її обслуговування та час розробки, що несумісно з програмним продуктом, таким як MVP.

Основну семантику доступу та пошуку сховищ ключів і значень спільно використовують бази даних документів. У цих базах даних також використовується ключ для ідентифікації унікальних даних. Дані зберігаються в структурованих форматах, таких як JSON, BSON або XML, у документно-орієнтованих базах даних. Їм бракує послідовної та чітко визначеної структури даних. Вони є чудовим рішенням для швидкого створення невеликих додатків із змінною схемою даних.

Ще однією перевагою таких баз даних є горизонтальне масштабування або додавання фізичних серверів і розповсюдження даних між ними. Однією з проблем є те, що бази даних документів повільно оновлюються, оскільки дані можна копіювати та розподіляти на кількох серверах. Крім того, атомарні транзакції, які відповідають за цілісність даних, не підтримуються. MongoDb, CouchDb і MarkLogic є прикладами таких баз даних.

Хоча бази даних стовпчастого типу є частиною сімейства NoSQL, за зовнішнім виглядом вони нагадують реляційні бази даних. Такі сховища, як і реляційні бази даних, зберігають дані в рядках і стовпцях, але відносини між частинами різні. Усі рядки в реляційних базах даних мають відповідати набору

правил, які визначають, які стовпці будуть у таблиці, типи даних та інші фактори. Замість таблиць у стовпцях типу стовпців є структури, які називаються «сімействами стовпців».

Рядки входять до сімейств, кожна з яких має свій унікальний формат. Рядок починається з ідентифікатора для пошуку, за яким слідують набори назв і значень стовпців.

Такі бази даних особливо корисні для зберігання інформації.

У таких базах даних не всі таблиці заздалегідь визначені та задані.

Вони створені для пошуку даних, тому дублювання або денормалізація є природною частиною процесу. Цей метод використовується, коли потрібно постійно записувати свіжу інформацію, а не виконувати запити на читання, наприклад, зчитувати дисплей датчиків у режимі реального часу.

Однак використання баз даних стовпців не є життєздатним рішенням для програмної системи, створеної в дипломному проекті, оскільки система має велику кількість вимог до фільтрації та сортування, які надзвичайно важко виконати при використанні таких баз даних.

Крім того, недоліком таких баз даних є те, що видалені дані «стираються», генеруючи блоки, які можуть збиратися з часом і вимагати подальших операцій для очищення, що збільшує складність обслуговування такої програмної системи, що є нездійсненним для продукту MVP. Бази даних стовпців включають Cassandra і ScyllaDb.

Після вивчення та порівняння [14] різних типів баз даних було визначено, що реляційні бази даних слід використовувати разом із сховищами ключ-значення, які виконують операції кешування даних, щоб прискорити роботу системи та усунути один із недоліків ACID.

2.2.2. Проектування логічної моделі «Сутність-зв’язок»

Модель «Сутність-зв’язок» [16] зображує концептуальну модель світу, яка буде відображатися в базі даних. Він заснований на сприйнятті реального світу, який складається з сукупності предметів, відомих як сутності, та їх зв’язків.

Модель ER не залежить від даних у високій мірі. Для створення схеми бази даних у моделі ER можна використовувати діаграму сутність-зв’язок.

Основними будівельними блоками у ER-моделі є:

– Entity – сутність, що є проекцією об’єкту із предметної області;

– Attribute – атрибут, це властивості сутностей, тобто те, що їх описує;

– Relationship – відношення між сутностями.

У моделі ER відносини є одним з найважливіших компонентів для опису зв'язків об'єктів. Існують три типи зв’язків між суб’єктами.

З'єднання "один до багатьох" – це зв'язки, в яких одна сутність має кілька залежних. Користувачі, які зробили пости, є ілюстрацією цього типу. Це приклад зв’язку, оскільки одна особа може подати кілька дописів.

Відношення «один до одного» – це тип зв’язку «один до багатьох». Наприклад, користувач із лише одним набором налаштувань програми має лише один набір налаштувань, а кожен користувач має лише один набір налаштувань. Такі зв’язки використовуються для поділу величезного об’єкта, наприклад представлення характеристик, на окремі сутності. Це призводить до отримання двох одиничних елементів.

З’єднання «багато до багатьох» з’єднує кілька сутностей з багатьма залежними об’єктами. Це можуть бути, наприклад, відносини між користувачами та людими, за якими вони стежать. Один може бути підписаний на кількох користувачів, і один може бути підписаний на багатьох користувачів. Зазвичай такі з’єднання поділяють на дві категорії: «один до багатьох» і «багато до одного».

Побудова належної ER-моделі, а отже, і побудова моделі реляційної бази даних вимагає розщеплення одного зв’язку та встановлення численних функціонально залежних зв’язків. Цей процес відомий як нормалізація даних. Нормалізація мінімізує зайві дані, покращує загальну організацію бази даних, дозволяючи дизайну бути більш гнучким, і, головне, забезпечуючи узгодженість даних.

Існує три основних форми норми. Перша нормальна форма усуває зайві групи в окремих сутностях, створює нові сутності для кожного зв’язаного набору

даних і використовує первинний ключ для ідентифікації кожного зв’язаного набору даних. Первинний ключ – це одна властивість (або комбінація властивостей), яка однозначно ідентифікує кортеж (рядок) зв’язку.

Окремі сутності встановлюються у другій нормальній формі для наборів значень, які застосовуються до кількох записів. Зовнішній ключ з'єднує ці дві сутності. Зовнішній ключ – це властивість зв’язку, яка відповідає основному ключу іншого або того самого зв’язку. Записи не повинні покладатися на головний ключ таблиці, якщо це не є абсолютно необхідним.

Поля, які не залежать від ключа, опускаються з третьої нормальної форми. У більшості випадків для правильної побудови логічної моделі достатньо третьої нормальної форми.

На початку дослідження предметної області під час створення логічної моделі програмної системи було виявлено ряд сутностей.

User – зареєстрований користувач додатку.

Атрибути сутності User:

- First name – ім’я;
- Last name – прізвище;
- User name – коротке ім’я зареєстрованого користувача;
- Country – місце проживання;
- Email – адреса електронної пошти;
- Hashed password – хешований пароль;
- Picture key – посилання на зображенно користувача;
- Locale – мова інтерфейсу;
- User Type – тип користувача (Superuser, модератор, користувач).

Post – публікація із рецептом, що створюється користувачами системи.

Атрибути цієї сутності:

- Title – назва страви;
- Description – опис;
- Preview key – посилання на зображення страви;
- Youtube link – посилання на відео приготування страви;
- Cook time – час приготування страви;

– Complexity – складність приготування;

– Calorific – калорійність страви;

– Potions – кількість порцій;

– Healthy food – визначає чи є страва корисною;

– With multicooker – визначає чи можливе приготування страви за допомогою мультиварки;

– Status – визначає статус посту: 0 – активний, 1 – знаходиться у модерації, 2 – заблокований;

– Reports – кількість скарг;

– Private – визначає налаштування видимості.

Comment – коментар до посту іншого користувача, має поле Content – зміст коментарів.

Recipe Book – книга рецептів, що може містити як рецепти власника так і рецепти інших користувачів. Має такі атрибути:

– Name – назва;

– Description – опис.

Дії користувачів щодо публікації описуються сутністю Post Action. Це може бути лайком або закладкою. Вона містить два атрибути: Preffered і Bookmarked, які вказують, чи була публікація оцінена чи збережена іншими людьми.

Post Detail – детальний опис страви.

Ingredient – складові страви. Складається з наступних полів:

– Name – назва складової;

– Optional – чи є він потрібним;

– Units – одинці міри кількості.

Сутності Category (Категорія), Subcategory (Підкатегорія), Specific (Специфікація – безглютенова дієта, наприклад), Assignment (Призначення – на обід, для дітей, наприклад), Cuisine (Національна кухня) мають атрибут Name.

Налаштування користувача програми описані в сутності User Settings. Має мовні налаштування інтерфейсу, ShowLastActiveFollowings – показує останніх підписників користувача. ShowDayTimeRecipes – відображення рецептів на

основі поточного часу доби. Крім того, усі об’єкти включають час створення запису, який потрібен при роботі з розбиттям на сторінки – розділення даних на маленькі шматки, щоб зменшити серйозне навантаження на систему через розмір даних.

Налаштування користувача мають атрибути Locale та Language, як показано в таблиці, що вказує на те, що програмна система забезпечує багатомовний інтерфейс. У результаті було прийнято рішення розробити сутності, які локалізують їх за допомогою атрибутів Locale на додаток до довідкових сутностей, таких як інгредієнт, категорія тощо.

Облікові дані користувача, такі як адреса електронної пошти, пароль і тип, також були переміщені до окремої таблиці UserIdentity, яка має відношення із таблицею User. RefreshToken і RefreshTokenExpired – це нові характеристики, які реалізують систему безпеки програми за допомогою пари токенів безпеки. Коли обліковий запис користувача позначено для видалення, властивість MarkedForDelete визначає, коли він отримає сповіщення. Відповідно до стандартів соціальних мереж дані користувача мають бути відновлені, тому, коли обліковий запис знищено, він позначається певним значенням, яке дозволяє системі видалити їх, якщо користувач не відновить дані через певний період часу. EndBlockDate вказує, коли система розблокує користувача.

В результаті система сутностей, показана у вигляді діаграми ER, була роздріблена після нормалізації даних відповідно до третьої нормальної форми.

Складені ключі, які є частиною первинного ключа і однозначно описують з’єднання, показані на діаграмі CK (Composite Key). У відносинах «багато-до-багатьох» є також інші суб'єкти, які діють як посередники. Об’єкт UserFollow, наприклад, описує підписників і підписки користувачів, тоді як об’єкт PostIngredient визначає набір інгредієнтів та їх кількість для певного рецепта.

Тому що кожна соціальна мережа потребує встановлення певних даних, таких як кількість підписок, вподобань, публікацій для певної категорії тощо. Ви можете досягти цього, використовуючи базові запити, включені в усі поширені реляційні бази даних. Оскільки системи баз даних перевіряють кількість даних на узгодженість відповідно до принципу ACID, ця процедура може зайняти

непомірно багато часу для систем із високим навантаженням. В результаті було вирішено денормалізувати дані, що тягне за собою додавання нових атрибутів-лічильників, що визначають кількість.

Щоб уникнути зайвих запитів під час читання даних, атрибути кількості були додані до об’єкта User, наприклад FollowersCount – кількість підписників, FollowingsCount – кількість підписок, BookmarksCount – кількість збережених рецептів та LastPublicationTime – останнє опубліковане. PreferencesCount – кількість лайків, BookmarksCount – кількість збережених разів, Comments Count – кількість коментарів до публікації, доданих до публікації.

В результаті можна створити реляційну фізичну модель, використовуючи логічну модель як основну. Однак для застосування цієї концептуальної моделі необхідно вибрати правильну СКБД.

2.2.3 Вибір реляційної системи керування базами даних

Для вибору реляційної бази даних було розглянуто три популярних варіанта: MySQL, MS SQL Server та PostgreSQL [36-39].

MySQL є найбільш широко використовуваною безкоштовною системою управління базами даних з відкритим вихідним кодом. Він підтримує різноманітні механізми баз даних з різною архітектурою та продуктивністю. Завдяки своїй великій продуктивності механізм зберігання MyISAM в основному використовується для читання. У результаті ця база даних може бути адаптована до ваших конкретних вимог. Ще однією перевагою є величезна інтернет-спільнота та велика кількість безкоштовної технічної документації. MySQL також не потребує багато комп’ютерних ресурсів. Ця система управління базою даних постачається з низкою інструментів для моделювання та виробництва даних, а також із середовищем запитів SQL та візуалізації даних. Він працює на будь-якій операційній системі і є кросплатформним.

Microsoft SQL Server – це рішення для керування базами даних для реляційних баз даних. Його основна мета – створення комерційних додатків. Він

поставляється з багатоцільовим клієнтом бази даних і інструментами розробки. Ще одна перевага полягає в тому, що ця база даних тісно пов’язана з платформою ASP.NET, що робить її швидким і простим налаштуванням. Microsoft надає просту для розуміння документацію в Інтернеті. Ця база даних має чудову продуктивність. Має безпечну систему відновлення даних. Недоліком є ціна версії Enterprise, яка надає численні можливості для обслуговування систем із високим навантаженням. Її надто дорого включати в продукт MVP, а версія Express, яка є безкоштовною, має певні недоліки, наприклад, обмежений простір для зберігання та недостатній захист даних користувача. Іншим недоліком є те, що Microsoft SQL Server сумісний лише з системами на базі Windows. У результаті ми можемо зробити висновок, що ця база даних непридатна для використання в програмній системі, яка автоматизує публікацію, розповсюдження та пошук рецептів [40-43].

PostgreSQL – це реляційна система керування базами даних для об'єктів [17]. Він відрізняється від усіх інших рішень тим, що повністю підтримує нереляційні дані, обробляє та ефективно їх зберігає. Наприклад, спеціальні оператори використовуються для пошуку даних JSON у стовпці відповідного типу. Це допомагає вам знаходити дані цього типу набагато швидше, ніж, наприклад, у MySQL. В результаті PostgreSQL є універсальною базою даних. Також доступний добре розроблений механізм індексації таблиць, який покращує швидкість читання з величезних таблиць. За допомогою конкретних індексів повнотекстовий пошук значно швидше, ніж із еквівалентами PostgreSQL, що є однією з основних потреб для швидкого пошуку даних у соціальних мережах. Вона є кросплатформенною, тобто може працювати та розгортатися в будь-якій операційній системі. Підтримка сторонніх розширень є однією з її переваг, що робить цю базу даних максимально універсальною. Це дозволяє вам застосовувати розширення, щоб ще більше підвищити швидкість роботи бази даних. PostgreSQL має кращий оптимізатор запитів, ніж MySQL, що дозволяє швидко виконувати складні запити. Складна система транзакцій. Він також має значну онлайн-спільноту, велику технічну літературу та просту у використанні онлайн-документацію. Дозволяє дублювати базу даних, що

означає, що ви можете робити її копії, щоб зняти напругу системи. Він також включає драйвери ASP.NET для різноманітних платформ і фреймворків.

Однак є і деякі недоліки. Оскільки PostgreSQL робить акцент на сумісності, збільшення швидкості обробки запитів вимагає більше зусиль, ніж у MySQL. Крім того, якщо ви не використовуєте такі функції бази даних, як індекси або автоматичне очищення зайвих кортежів (вакуумування), виконання запиту може значно сповільнитися, що негативно вплине на продуктивність. Як наслідок, налаштування, адміністрування та підтримка PostgreSQL набагато складніше, ніж інші реляційні системи.

Після огляду найпоширеніших систем керування базами даних для впровадження програмної системи було обрано базу даних PostgreSQL, оскільки вона має численні переваги порівняно з іншими базами даних, постійно розвивається та пропонує широкий спектр опцій.

2.2.4 Проектування модульної архітектури

Після вибору системи управління базою даних проектується модульна архітектура програмного забезпечення.

Сім пов'язаних модулів утворюють основу системи. Усі вони є частиною відповідних шарів багатошарової монолітної архітектури.

Application є основним модулем. Це відноситься до рівня презентації. Він містить основну логіку програми, реєстрацію функціональних служб, файли конфігурації, набір контролерів для зв’язку з клієнтською стороною програми та класи міграції для керування історією змін структури бази даних. Модуль Program відповідає за виконання програми та інтеграцію всіх інших компонентів програмної системи.

Модуль Contracts – це загальна колекція інтерфейсів, реалізованих в інших модулях, які не належать до жодного рівня архітектури [44-48].

Entities – це модуль, який надає моделі, які визначають спосіб відображення реляційних даних. Це на рівні керування даними (рівень даних). Також включено набір класів DTO (Data Transfer Object). Він складається з

об’єктів, які представляють ресурси запиту. Контекстні класи надають абстракцію над запитами до бази даних у пакеті Contexts.

Модуль Services дозволяє централізоване адміністрування даних, наприклад, виконання операцій CRUD над даними (створення, читання, оновлення та видалення). Це частина сервісного рівня.

До складу модуля Utility входить, наприклад, сервіс для відправки листів на адреси електронної пошти, що є допоміжною можливістю системи.

Об'єктні моделі, які є проекціями сутності діаграми ER, знаходяться на рівні даних. Кожен клас моделі представляє окрему сутність.

Майже всі класи повторюють інформацію, включену в сутності ER-діаграми, проте були додані нові, уніфікуючі класи, що містять фундаментальні атрибути для додаткових моделей. Оскільки всі сутності мають властивість створення записів, усі класи успадковуються від PostgresEntityBase. Цей атрибут знадобиться в майбутньому для виконання розбиття даних на сторінки, тобто поділу даних на розділи. PostgresRelatedEntity – це набір класів, пов'язаних з об'єктами Post. В основному це таблиці з’єднань, які за моделлю ER служать для розриву зв’язків «багато до багатьох». Такі зв’язки в об’єктній моделі називаються асоціаціями. Вони є результатом проектування зв’язків «один до багатьох» або «один до одного».

Реляційна модель проектується на об'єкт за допомогою об'єктно-реляційного відображення ORM (Object-Relational Mapping). Entity Framework Core (EF Core) – це технологія Microsoft, яка дозволяє взаємодіяти з базами даних на більш високому ступені абстракції, що дозволяє абстрагуватися від бази даних та її таблиць і працювати з даними незалежно від формату зберігання. Контексти являються спеціальними класами, які EF Core використовує для доступу до даних.

Модуль DTO являє собою набір класів, які використовують лише дані, необхідні для вираження моделі. Їх також можна використовувати для опису лише деяких аспектів моделі. Ідея використання такого шаблону проектування полягає в тому, що в більшості випадків користувач не подає на сервер інформацію, яка повністю відповідає певній моделі, а лише її частину, або не

вимагає повної інформації про модель. Модуль Mappings, з іншого боку, використовує відображення між DTO та об’єктною моделлю. Оскільки класи DTO не є стабільними і не можуть бути сплановані заздалегідь, вони пояснюються в розділі 3 записки.

Модуль QueryObjects є основою сервісного рівня програмної системи. Він використовується для відстеження даних і виконання запитів, є обгорткою для стандартних методів EF Core.

Загалом є два шаблони проектування, які дозволяють отримувати дані та інкапсулювати на них загальні дії CRUD. Репозиторій і об’єкт запиту. Шаблон сховища – це шар абстракції, який містить прийоми обробки даних. Відокремлює бізнес-логіку від особливостей реалізації методів доступу до даних. Найкраще створити окремий репозиторій для кожної бізнес-одиниці, оскільки охоплення всієї функціональності системи лише простими операціями CRUD, як правило, непрактично у великих проектах. В результаті в його дизайні використовується шаблон, відомий як «Одиниця роботи». Він використовується для зберігання всіх сховищ і формує до них єдину точку доступу під час запуску програми. Одним з найпоширеніших креативних шаблонів дизайну для створення окремого екземпляра об’єкта є Singleton.

На рисунку 2.6 зображена архітектура програми з використанням шаблону «Репозиторій» на прикладі трьох моделей програмної системи: User, Post і Comment.

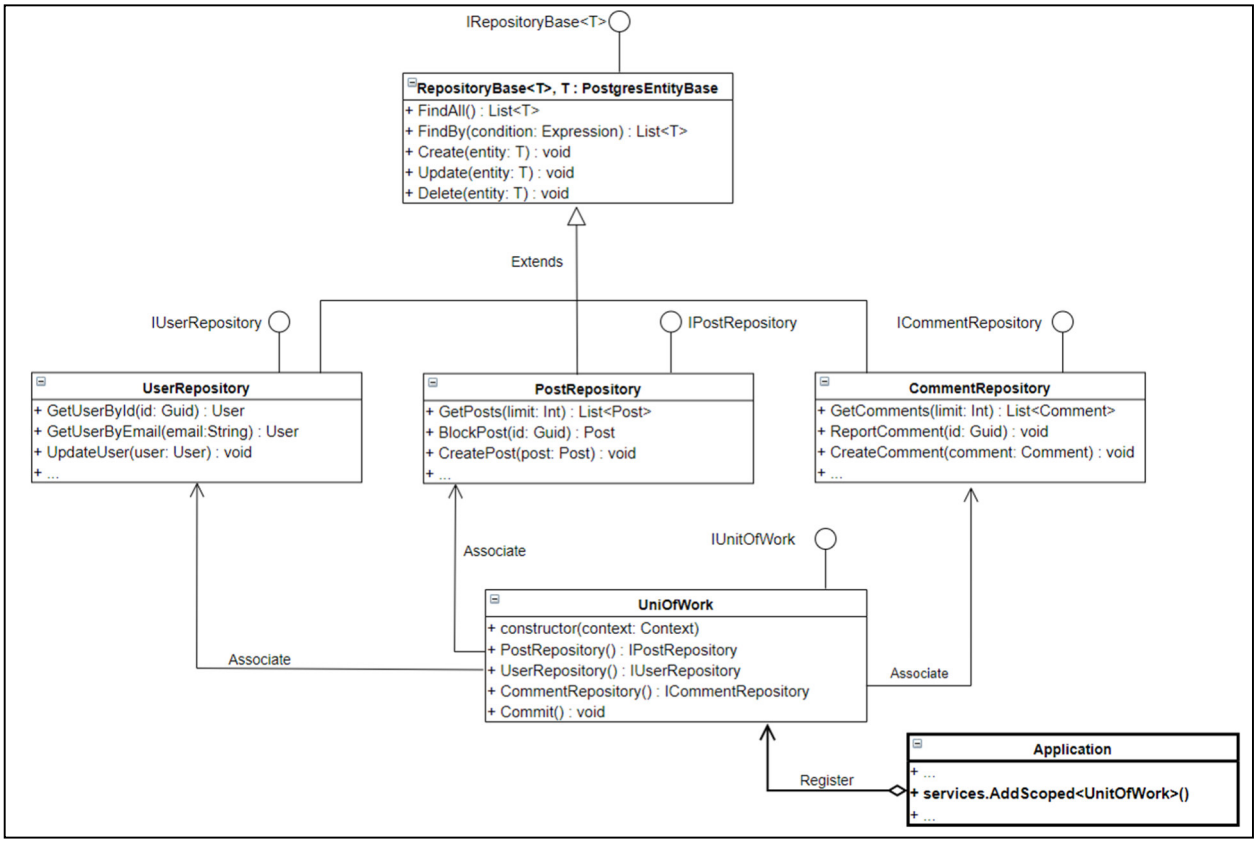


Рисунок 2.6 – Діаграма патерну проектування «Репозиторій»

Усі класи супроводжуються відповідними інтерфейсами, як показано на схемі (рис. 2.2). Класи UserRepository, PostRepository і CommentRepository надають нові методи для основного класу RepositoryBase, який включає лише основні методи CRUD для взаємодії з даними. Клас UnitOfWork є єдиним способом доступу до репозиторіїв, і це єдиний елемент у класі основної програми. Метод Commit зберігає всі зміни бази даних. Шаблон репозиторію має перевагу, що дозволяє легко знаходити дефекти в коді та проводити модульне тестування, розділяючи його на локальні репозиторії. Цей шаблон проектування також є прикладом тематичного програмування, оскільки він дотримується концепції DDD (Domain Driven Design) генерування моделей, які відповідають заданій предметній області. Наприклад, кожний зі створених репозиторіїв працює лише з даними для однієї моделі [50-53].

Однак наявність великої кількості моделей збільшує кількість репозиторіїв та інтерфейсів, які мають ці репозиторії, що погано впливає на LOC-метрики (рядки коду), що значно уповільнює розробку програмного забезпечення та ускладнює тестування. Проект також містить нетривіальні запити до бази даних,

такі як розбиття курсором на сторінки та повнотекстовий пошук за допомогою вбудованого механізму сканування слів і фраз. Оскільки такі запити не підтримуються звичайними методами EF Core, їх можна виконувати лише без використання стандартних бібліотечних інструментів, тобто безпосередньо через інтерполятор запитів SQL, який також включено в EF Core. Це означає, що на практиці методи CRUD, визначені в базовому репозиторії, не будуть використовуватися, що призведе до надмірності всіх локальних сховищ.

Об'єкт запиту є альтернативним шаблоном проектування. Він складається з набору об’єктів запиту. Кожен клас відповідає або за виконання окремого запиту (наприклад, отримання списку користувачів), або за набір запитів, які працюють разом з даними бізнес-логіки (наприклад, для створення користувача потрібні три запити одночасно, оскільки дані повинні вводити відразу). з трьома столами, з’єднаними разом). Усі об’єкти запиту передаються до певного класу під назвою «Виконавець запитів», який виконує запити та повертає результат виконання об’єкта.

В результаті перевага цього методу полягає в тому, що кожен об’єкт запиту є окремою ізольованою структурою, яка, як і репозиторій, дотримується парадигми DDD, тобто обробляє лише дані з однієї моделі. Модульне тестування та виявлення несправностей з такими об’єктами легше, ніж із репозиторіями. Крім того, об’єкт запиту містить всю логіку для складних запитів.

Недоліком є те, що ми повинні розробляти об’єкти запиту навіть для простих запитів, які можна виконувати за допомогою основних методів EF Core. В результаті було вирішено побудувати прості методи для виконання основних дій CRUD, поєднавши стиль «Об’єкт запиту» з частиною шаблону «Репозиторій». На рисунку 2.7 показано виведення трьох об’єктів запиту: «GetUsersByName», «CreatePost» і «UpdateUserCredentials», що означають «Отримати список користувачів за іменами», «Створити публікацію» та «Оновити облікові дані користувача» відповідно.

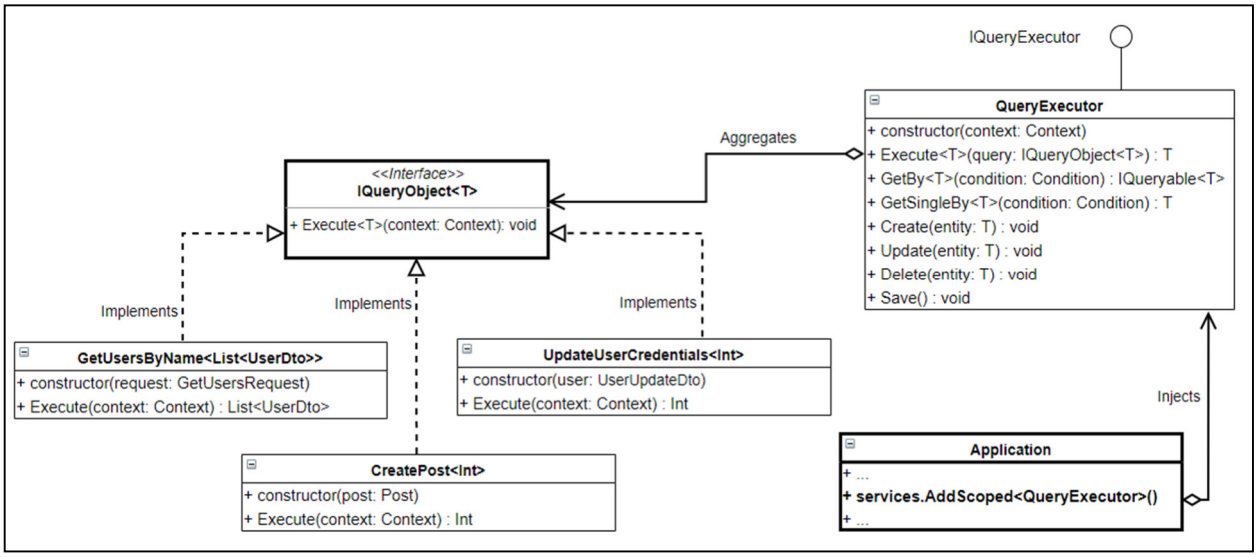


Рисунок 2.7 – Діаграма класів розширеного патерну «Об’єкт запиту»

Інтерфейс IQueryObject, який забезпечує єдиний метод Execute, який реалізує всі об’єкти запиту та містить логіку їх виконання, є основним елементом розширеного шаблону запиту «Об’єкт запиту», як показано на рисунку 2.7. Клас QueryExecutor подібний до класу UnitOfWork у дизайні Repository, але він не забезпечує єдиної точки доступу до всіх об’єктів запиту; натомість він працює як їх виконавець, відображаючи сукупний зв'язок між класом та інтерфейсом IQueryObject. Цей клас також включає всі фундаментальні методи і зареєстрований як однотонний в основному класі програми. Проте передбачити всі елементи запиту на етапі проектування архітектури дуже важко. Їх реалізація відбувається одночасно з програмною реалізацією системи, тому розділ 3 дипломного проекту містить опис реалізації кожного об'єкта запиту.

В результаті розширений шаблон оформлення запитів є якісною заміною громіздкого репозиторію. Це легко перевірити, кожен об’єкт розділений, і він виконує лише чітко визначені дії із зазначеними даними бізнес-логіки.

Модуль менеджерів – це модуль рівня сервісу, який виконує сервісні функції. Менеджери зазвичай розширюють функціональність вбудованих функцій бібліотек, щоб уникнути зайвих фрагментів коду в контролерах. Це дозволяє використовувати як вбудовані, так і сторонні бібліотеки, а також основні функції. Менеджери реєструються як синглон в основному класі програми.

Рівень відображення програми – це верхній рівень трирівневої монолітної конструкції веб-програми. Він відповідає за показ даних і створення веб-сторінок. У випадку з Web Api, він відповідає за коректну обробку запитів і видачу ресурсів сервера у належному форматі та відправку їх клієнту. Контролер є основним блоком. Це специфічний клас, який приймає запити, обробляє їх і дає відповідь клієнту за допомогою методів дії (Action methods). Контролери є місцем, де реалізується основна функціональність системи. Вони використовують сервісний рівень програмної системи для обробки запитів, що забезпечує основну функціональність. Метод Dependency Injection (DI) використовується для передачі всіх служб контролеру. Це шаблон дизайну для перенесення речей з одного об’єкта на інший без необхідності відтворення їх екземплярів. Цей метод використовується ASP.NET Core для реєстрації служб як одиночних, які згодом можуть бути вбудовані в інші служби або контролери. Шаблон відокремлює побудову залежностей служби від функціональності служби, дозволяючи компонентам бути слабо пов’язаними. Оскільки служби стають все більш автономними, це полегшує модульне тестування.

Крім того, рівень презентації програми розділений на дві частини: сервер і клієнт. Серверний компонент рівня перегляду складається з контролерів. Клієнтська частина програмної системи являє собою окремий веб-додаток, який надсилає запити та відображає клієнтську частину.

2.2.5. Проектування функціональної архітектури

Оскільки це проектування функціональних компонентів програми, функціональна архітектура програмної системи відрізняється від модульної архітектури. Один логічний модуль може використовувати кілька логічних модулів одночасно для отримання однакового результату. Функціональна архітектура – це повний опис і структура функціональності системи, створена з урахуванням технологічних і комерційних потреб, а також ієрархія функцій, їх взаємозалежність і застосування в компонентах системи. Функціональні вимоги моделюються та документуються за допомогою варіантів використання, а

навколо них будується архітектура. В результаті програмна система для публікації, розповсюдження та пошуку рецептів була розбита на три основні функціональні частини.

Одним із найважливіших функціональних модулів є аутентифікація, яка є основою безпеки системи. Другою частиною інформаційної безпеки є перевірка даних користувача на сервері. Першим кроком є підтвердження особи користувача, підтвердивши його логін і пароль. Третій компонент, авторизація, перевіряє привілеї користувачів і визначає, чи можливий доступ до захищених ресурсів.

JWT (JSON Web Token) – це відкритий стандарт (RFC 7519) для встановлення маркерів доступу у форматі JSON, який використовується сучасними додатками для аутентифікації. Сервер генерує токени, які підписуються секретним ключем і надсилаються клієнту, який використовує їх для перевірки їхньої особистості.

Токен складається з трьох частин: заголовка, який надає інформацію про техніку та тип шифрування; корисні дані, які містять дані користувача та час завершення (згорання) токена; та підпис, який підтверджує, що токен не був змінений. Однак використання одного токена для доступу до ресурсів є ризикованим, оскільки якщо він буде добутий зловмисниками, скажімо, за допомогою міжсайтового сценарію (XSS), новий користувач матиме постійний доступ до ресурсів. Зазвичай використовуються два токени: один називається токеном доступу JWT, а інший називається токеном оновлення, який може містити будь-яку кількість символів і бути зашифрований за допомогою будь-якого алгоритму. Використання пари токенів підвищує безпеку, оскільки неможливо використовувати токен доступу без токену оновлення. Якщо токен доступу буде отримно, користувач зможе використовувати його протягом обмеженого часу, але після закінчення терміну дії токена він не зможе отримати доступ до захищених ресурсів без токена. Якщо термін дії вашого маркера доступу закінчується, ви повинні попросити сервер надати вам новий набір маркерів доступу та оновлення. Це дозволить користувачеві продовжувати використовувати систему без повторного входу.

Однак перед проведенням аутентифікації користувач повинен спочатку створити та зберегти свої дані в базі даних. Сучасні онлайн-додатки використовують кілька типів перевірки під час створення користувачів. У дипломній роботі пропонується надіслати на електронну пошту конкретний код доступу для перевірки даних користувача.

Оскільки комп'ютерна система, ймовірно, буде сильно завантажена, запити на додавання та видалення кодів доступу з бази даних можуть мати згубний вплив на продуктивність. В результаті було вирішено встановити користувача в базу даних і записати дані користувача в кеш, де ключем є хешована адреса електронної пошти і пароль (токен перевірки), а значенням є код доступу. Кеш – це сховище даних на основі оперативної пам'яті, яке відповідає на запити значно швидше, ніж бази даних. Крім того, такі репозиторії мають вбудований механізм вимкнення, який видаляє дані, коли закінчується термін їх дії. Ця процедура створення користувача була реалізована за допомогою системи розподілу кешування Redis.

Коли створюється привілейований користувач, наприклад адміністратор або модератор, генерується пароль і доставляється на адресу електронної пошти, яку користувач може використовувати для автентифікації у системі.

Зміна адреси електронної пошти або пароля вимагає підтвердження користувача, надіславши код доступу на адресу електронної пошти, і виконується аналогічно.

Після встановлення та автентифікації звичайного користувача він може робити такі речі, як створювати й керувати новими повідомленнями, оцінювати й зберігати їх, а також коментувати публікації інших користувачів. При створенні публікації спочатку її зміст модерується, тобто адміністратори перевіряють, чи відповідає публікація всім правилам соціальної мережі, і, якщо не відповідає, видаляють публікацію та надсилають користувачеві сповіщення з попередженням. Це перший етап модерації. Існує також постмодерація, що означає, що дані перевіряються після того, як інші люди скаржаться, і вжито відповідних заходів. У цій програмній системі було визначено використання двох методів модерації. Під час публікації рецептів використовується попередня

модерація, постмодерація використовується під час написання коментарів та інших дій, таких як оновлення зображення профілю.

Користувачі також можуть надавати оцінки публікаціям і зберігати їх. Це часті дії, які також потребують оновлення лічильників (кількість лайків і збережених публікацій), які використовуються для замовлення рецептів, а оновлення в реляційній базі даних не є швидким процесом, особливо якщо таблиця має численні індекси.

В результаті було вибрано сховище кешу Redis. Робоча ідея ідентична ідеї вбудованого кешу пам’яті ASP.NET Core. Відмінність полягає в тому, що дані можуть зберігатися у фізичному сховищі в разі ймовірної втрати, наприклад, через переповнення кешу. Redis також може функціонувати як окремий сервер кешу, незалежно від основних процесів програми та знаходиться на іншому фізичному сервері. Це дозволяє підвищити продуктивність програми за рахунок розширення системи кешування та використання її як методу розподілу звантаженості та налаштування доступу.

Cache-Aside і Write-Through є двома основними методами кешування. Найбільш широкий попит має Cache-Aside. Коли програмі потрібно прочитати дані з бази даних, вона спочатку перевіряє кеш, щоб перевірити, чи доступні дані; якщо дані доступні (звернення до кешу), повертаються кешовані дані, а відповідь надсилається клієнту; якщо дані недоступні (промах кешу), в базу даних надсилається запит на отримання даних, потім дані заповнюються в кеш і надсилаються користувачу.

На відміну від Cache-Aside, підхід Write-Through попередньо оновлює кеш відразу після початкового оновлення бази даних. Основна логіка читання даних може бути сформульована так: програма змінює постійне сховище даних, а дані оновлюються в кеші незабаром після цього.

Ці дві тактики часто поєднуються. Це значно покращує продуктивність програми.

Однак для оновлення даних був використаний інший підхід до кешування. Його мета – мінімізувати навантаження на базу даних у разі високої активності та частоти запитів. Це може статися, коли користувач має велику кількість

підписників, і кожен з них задовольняє запит, наприклад, збереження публікації. Буде затримка оновлення лічильника, оскільки, як було зазначено раніше, поле в реляційній базі даних оновлюється повільно. В результаті логіка оновлення даних може бути описана наступним чином:

а) клієнт надсилає запит на оновлення лічильника;

б) дані заносяться у кеш і усі наступні запити виконують збільшення (зменшення) значення цього лічильника у кеші;

в) за деякий час до визначеного терміну інвалідації кешу, усі дані заносяться однією транзакцією у базу даних, виконуючи запит оновлення таблиці у БД, а з кешу видаляються.

При цьому логіка читання відбувається за стратегією Cache-Aside, тобто цілісність даних повністю зберігається.

Якщо під час запису в базу даних виникає помилка, дані кешу копіюються в постійне сховище на фізичному диску, і наступного разу, коли буде спроба записати з кешу в базу даних, спочатку перевіриться постійне сховище на диску, і якщо воно включає даних, вони спочатку вносяться в базу даних. Час вимкнення кешу часто встановлюється на 5-10 хвилин. Ми використали планувальник і сторонню бібліотеку Quartz, щоб створити подію відключення кешу, оскільки Redis не підтримує її обробку, оскільки вона непередбачувана і може виникати як до, так і після відмови.

Однією з функціональних потреб програмної системи є стирання даних. Згідно з політикою великих соціальних мереж, обліковий запис не видаляється миттєво. Спочатку він має позначку «Готовий до видалення», але його можна відновити через шість місяців або будь-який інший період. Обліковий запис остаточно знищується, якщо користувач не відновить обліковий запис до закінчення часу відновлення. Це також робиться за допомогою планувальника завдань, який щоденно перевіряє облікові записи та видаляє ті, термін дії яких закінчився. Під час моніторингу діяльності облікового запису використовується подібний підхід. Якщо користувач неактивний, тобто якщо він або вона не входили в систему протягом тривалого часу (6 місяців), то обліковий запис користувача видаляється назавжди.

Час блокування облікового запису також закінчується за допомогою вищезгаданої процедури, і обліковий запис автоматично розблокується. Модератори також мають можливість розблокування облікового запису.

2.3 Проектування інтерфейсу користувача

Розробка сценаріїв використання, проектування структури інтерфейсу, розробка прототипу інтерфейсу, визначення стилістики, вибір ідеї дизайну, дизайн екранів та анімація інтерфейсу – це всі етапи проектування інтерфейсу користувача.

Використання програмної системи дозволяє створювати індивідуальні сценарії. Оскільки варіанти використання описані в першій частині дипломного проекту, то можна перейти до другого етапу проектування.

Фундаментальна структура інтерфейсу користувача була побудована на основі сценаріїв використання, а саме екранів, що відповідають частинам або модальним вікнам онлайн-додатка. При створенні сценаріїв використання структура інтерфейсу зазвичай представляється схемою основних блоків, яка містить назву та список визначених функцій.

На рисунку 2.8 зображена структура інтерфейсу. «Д» вказує, що це модальне або діалогове вікно, а «Р» означає, що це розділ.

Основними розділами системи є стрічка новин та панель адміністрування. Загалом, функціональність веб-додатка надається у вигляді модальних вікон.

Модальні вікна використовуються для створення нових публікацій, авторизації або реєстрації користувачів, надання розширеної інформації про рецепти та зміни особистої інформації користувача, серед іншого.

Для підтвердження дій користувача, наприклад видалення, використовуються діалогові вікна з необхідним вмістом.

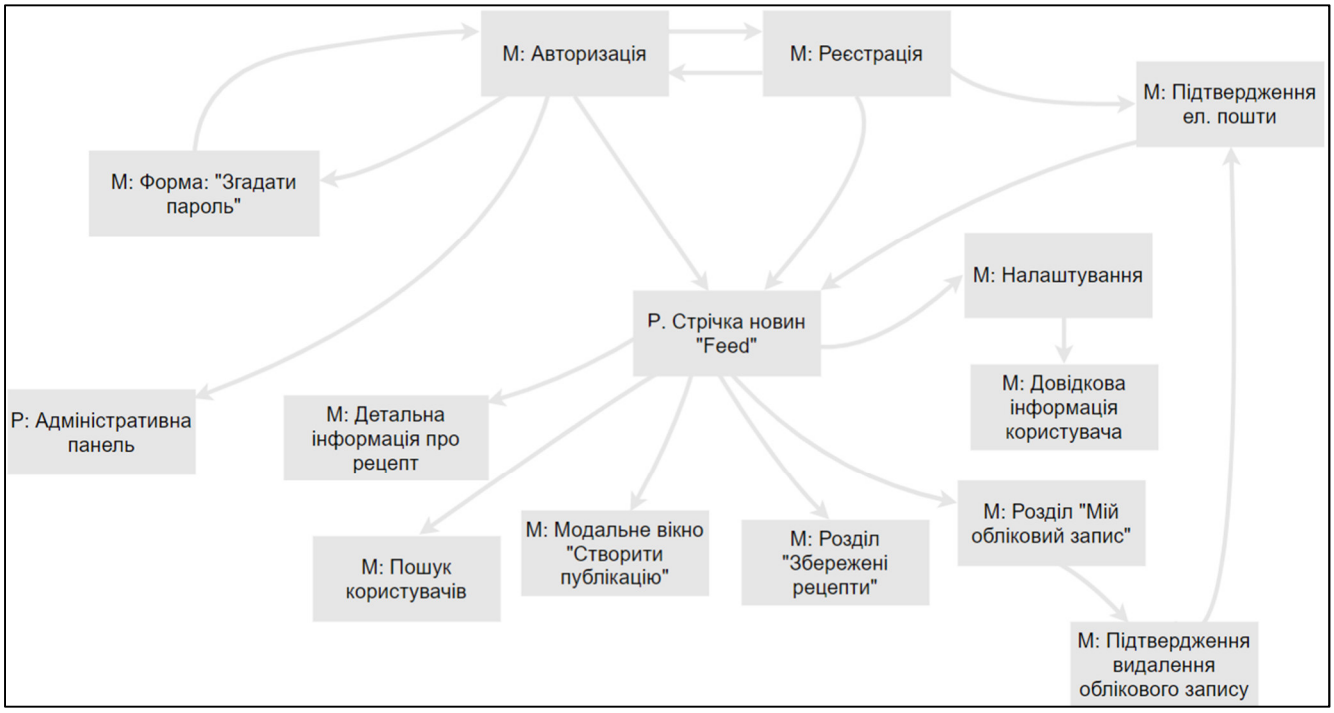


Рисунок 2.8 – Структура інтерфейсу програмної системи

Після цього на основі структури інтерфейсу розробляється прототип кожного структурного підрозділу. Функціонал і розміщення елементів сторінки на сторінці планується за допомогою прототипів. Набори зображень, піктограми, шрифти та функціональні компоненти (кнопки, поля введення тощо) готуються під час вибору стилю, що є наступним кроком у процесі проектування. Вони зазвичай керуються темою програми під час створення ідеї дизайну, і вони вибирають поточну тему, до якої користувачі вже звикли під час використання інших програм. Крім того, тема повинна бути створена з урахуванням особливостей людей з фізіологічними захворюваннями. Наприклад, це може бути дальтонізм, і дуже важливо вибирати кольори, які легко розрізнити навіть люди з вадами зору.

Для прискорення розробки програми запропоновано об’єднати всі три процеси проектування в один. Material Design був обраний як ідея дизайну, оскільки він популярний і зручний для користувачів через очевидні особливості інтерфейсу.

У лівій частині кожної сторінки програми є навігаційне меню. Верхня панель і основний компонент – стрічка новин – розташовані на головній сторінці.

Навігаційне меню має п’ять розділів: основна частина – стрічки новин, розділ для авторських публікацій, «Збережені рецепти», налаштування та особистий кабінет користувача.

Дописи інших користувачів з'являються в панелі новин. Вікно пошуку рецептів розташоване на верхній панелі. Рецепт можливо знайти за його назвою. Вікно сортування рецептів також розташоване поруч із полем пошуку. Можна сортувати за датою публікації, кількістю вподобань, калорійністю, складністю та часом приготування.

Панель з найактивнішими користувачами розташована вгорі, поруч з нею є кнопка пошуку додаткових користувачів. При натисканні на нього відкриється модальне вікно пошуку користувача.

Доступний пошук за іменем користувача. Кнопка підписки знаходиться праворуч. Доступ до публікацій та сторінки загальнодоступної інформації користувача можна отримати, натиснувши на його ім’я.

Картки використовуються для надання інформації. Зображення профілю автора та його назва відображаються у верхній частині картки. Деталі рецепту (час приготування, калорії тощо) також представлені в скороченому вигляді. Нижче наведено зображення рецепта з кнопками оцінки, коментування та збереження, а також короткий опис рецепта, дата його публікації та кнопка «Деталі», яка відкриває модальне вікно з додатковою інформацією про страву. Воно містить кулінарне відео, список інгредієнтів і категорій рецептів, етапи приготування, описи та коментарі.

Форму для заповнення необхідних даних для публікації рецепта можна знайти в розділі для публікації. Є можливість створити індивідуальний рецепт і зберегти його у збережених рецептах, не публікуючи, на додаток до стандартного рецепта.

Доступна можливість змінювати та видаляти дані, а також оновлювати та видаляти рецепти на сторінці «Мій обліковий запис», яка також містить статистику (кількість підписників, рецепти тощо).

Діалогове вікно підтвердження з’являється, коли ви натискаєте посилання видалити обліковий запис.

У розділі «Збережені дописи» зберігаються всі рецепти користувача. Книги рецептів також можна використовувати для організації публікацій. Для класифікації використовується кількість рецептів у книзі та дата її виходу. Решта розділів та елементів інтерфейсу по суті ідентичні описаним.

Після отримання відповідної ролі лише модератори мають доступ до адміністративної панелі. Він має таблиці для додавання, редагування та видалення категорій рецептів, а також панель керування публікаціями користувачів. У цьому прикладі є перегляд опублікованих публікацій, а також кнопки для публікації та вилучення рецептів. Також є посилання для перегляду скарг споживачів, а також налаштування блокування людей, якщо їх занадто багато.

Кнопку для системного адміністратора можна знайти в рядку навігаційного меню. Якщо користувач не має права, кнопки для створення та збереження рецептів вимкнені. Коли ви натискаєте кнопку облікового запису, з’являється вікно авторизації. Якщо неавторизована особа спробує оцінити або зберегти публікацію, з’явиться це діалогове вікно.

2.4 Аналіз та вибір сучасних технологій для реалізації клієнтської частини веб-додатку

Для розробки інтерфейсу користувача – клієнтської сторони програмної системи було досліджено кілька відомих технологій, включаючи бібліотеку React і фреймворк Angular.

React – це платформа, розроблена Facebook, яка дозволяє розробляти інтерфейс користувача для односторінкових програм (SPA), які не потребують перезавантаження. Головною перевагою бібліотеки є швидкість, з якою можна створити додатки, а також готові ресурси та шаблони, які можна використовувати замість створення власних. React покращує продуктивність, використовуючи дерево елементів віртуальної веб-сторінки (DOM). Стандартний DOM оновлюється довго. Щоб вирішити цю проблему, React представив віртуальний DOM. Віртуальний DOM є повністю заснованим на

пам’яті аналогом DOM веб-браузера. Програма працює швидше, оскільки лише один вузол моделі документа змінюється, коли ви змінюєте будь-який компонент.

Створення складних програм може бути складним і займати багато часу, оскільки React – це лише бібліотека. React просто звертається до рівня інтерфейсу програми, йому не вистачає визначеної архітектури програми та повного набору інструментів (таких як служби, модулі, директиви тощо). В результаті React підходить лише для крихітних проектів.

Angular, як і React, є інженерною платформою з відкритим кодом для створення інтерфейсів користувача. Ця платформа використовує як одну зі своїх функцій об’єктно-орієнтовану мову програмування TypeScript, а не JavaScript. Це значно полегшує підтримку та тестування продукту. Крім того, на відміну від бібліотеки React, фреймворк Angular має безліч вбудованих функцій у вигляді сервісів, модулів, компонентів і директив, які допомагають побудувати фундаментальну архітектуру та принципи розробки веб-додатків, скорочуючи час розробки. Angular використовує ієрархічну реалізацію залежностей і візуалізатор Ivy, який використовує підхід до дерева для оновлення стану компонента. Це сприятливо впливає на продуктивність програми. RxJs, бібліотека для роботи з даними, також включена.

Його недоліком є складність створення та розуміння. Платформа Angular була обрана для створення інтерфейсу користувача програмної системи, оскільки вона зазвичай використовується для великих, масштабованих корпоративних додатків.

2.5 Висновки

В результаті аналізу існуючої клієнт-серверної архітектури, нові онлайн-додатки все частіше використовуються як серверний компонент Web API.

Також описувалися переваги та недоліки розробки архітектури монолітного та мікросервісного сервера, причому монолітний стиль розглядається як хороший варіант для розробки мінімально життєздатного

продукту.

За основу була обрана система управління реляційною базою даних PostgreSQL, а сервер Redis виступає як рівень кешу після оцінки багатьох типів сховищ даних. Також побудовано концептуальну модель системи.

У процесі модульного проектування були визначені основні програмні пакети, створена об’єктна модель, а «Об’єкт запиту» визначено як базовий шаблон проектування на рівні сервісу монолітного типу серверної архітектури.

Було проведено аналіз функціональних вимог як наслідок функціонального проектування, і на їх основі ґрунтувався опис первинних процесів у системі, а також основних методів кешування в контексті конкретних проектних рішень.

Також були створені основні аспекти користувальницького інтерфейсу, досліджені нові технології для впровадження та обрана платформа Angular для реалізації клієнта.

3 РЕАЛІЗАЦІЯ ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ РОБОТИ З РЕЦЕПТАМИ

3.1 Створення бази даних програмної системи

Фактична база даних – головне сховище, яке міститиме організовані дані – будується і створюється, коли визначена базова модель програмної системи та обрана система управління реляційною базою даних. Для початку діаграма сутність-зв’язок використовується для створення таблиць і зв’язків між ними. Зовнішній та первинний ключі встановлені. Отримали таблиці, які відповідають суті концептуальної моделі після виконання всіх процедур нормалізації. Після цього кроку побудови бази даних проводиться перевірка таблиць із значною кількістю записів. Як наслідок аналізу були обрані наступні таблиці: Post (яка містить повідомлення користувачів у нескінченній кількості) та будь-які пов’язані таблиці з даними стосовно «Багато до багатьох», наприклад post_action, яка описує дії користувача. Таблиця User, яка зберігає дані користувача, а також пов’язані таблиці, може містити багато даних. Таблиці, за якими будуть побудовані індекси, були обрані після завершення дослідження. Індекси – це спеціальні таблиці в реляційній базі даних, наприклад PostgreSQL, які використовуються для прискорення пошуку інформації у базі даних. Коли створюється індекс і призначається стовпцю, усі дані в таблиці, а також дані, які будуть додані в майбутньому, будуть відсортовані за певними критеріями на основі типу індексу. У PostgreSQL існують різні типи індексів: B-tree (бінарне дерево), Hash (хеш-індекс), GiST і GIN (для повнотекстового пошуку). Для різних видів індексів використовуються різні методи впорядкування даних. Основна перевага індексів полягає в тому, що вони прискорюють вибірку даних. Однак, оскільки швидкість запису та оновлення може знизитися, це слід враховувати при розробці моделі індексу. Обираються стовпці таблиць і вирішується, чи використовувати для них індекс. Також, якщо необхідно, обирається тип індексу. Індекси також можна створювати, охоплюючи багато стовпців одночасно. В результаті було проведено ширше дослідження та створено модель індексу бази даних.

Запит для створення індексу B-tree наприклад на стовпець last_publication_time у таблиці User виглядає так:

```
CREATE INDEX last_publication_time_idx ON "User" USING btree
("LastPublicationTime" DESC, "UserId" DESC)
```

Індексу дається ім’я і визначається, до яких стовпців він буде застосований під час його створення. Створюючи індекс B-tree також потрібно відсортувати дані (у порядку зростання або спадання) для пошуку. Аналогічно було побудовано інші індекси. Стовпці таблиці та основні індекси, які посилаються на них, наведені в таблиці 3.1.

Таблиця 3.1 – Список індексів

Таблиця	Назви стовпців	Тип індексу
User	LastPublicationTime, UserId	B-tree (desc, desc)
	UserName, FirstName, LastName	GIN
Post	PostTitle	GIN
	CreatedAt, Id	B-tree (desc, desc)
	PreferenceCount, CreatedAt, Id	B-tree (desc, desc, desc)
RecipeBook	Name	GIN
Comment	CreatedAt, Id	B-tree (desc, desc)

База даних має два типи індексів, як показано на схемі: B-tree та GIN. Перший використовується для прискорення вибірки та сортування сторінок даних. GIN є оригінальним методом для прискорення повнотекстового пошуку. Ця програмна система використовує його для прискорення пошуку рецептів за іменем або користувачів за іменем, прізвищем та іменем користувача.

Розбиття на сторінки – це метод вибірки даних поетапно. «Зміщене» розбиття на сторінки є найпоширенішим типом. Це робиться шляхом вказівки обмеження, наприклад кількості рядків для вибору та кількості рядків, які

потрібно пропустити з початку вибірки. Крім того, інформація має бути упорядкована. При роботі з величезними обсягами даних цей метод розбиття на сторінки неефективний, оскільки кожен крок збільшує кількість рядків, які необхідно пропустити.

Keyset – це ще одна форма розбиття на сторінки. Він визначає умови, за яких будуть вибрані рядки, які будуть включені в блок розбиття на сторінки. Інформація має бути упорядкована. Основна перевага полягає в тому, що, на відміну від «офсетного», цей стиль розбиття на сторінки не використовує метод пропуску рядків. Це підвищує швидкість зразка. Якщо немає необхідності розділяти блоки даних на окремі, чітко визначені сторінки, використовується ця пагінація. Це задовольняє потреби програмної системи. В результаті його було обрано в якості основної техніки сторінки. Індекси зворотного порядку для асоційованих стовпців B-tree були встановлені для підвищення частоти вибірки даних (таблиця 3.1). Зразок запиту звичайних даних розбиття на сторінку виглядає так:

```
SELECT * FROM "Post" WHERE ("Post"."CreatedAt", "Post"."Id") <
(dateMark, idMark) AND "Post"." CreatedAt" < initialDateTime
```

Оскільки може виникнути обставина, коли час створення двох публікацій однаковий, цей запит порівнює дані за допомогою кортежу, і існує додаткова умова для унікальної ідентифікації. Поля dateMark та idMark — це дата в ідентифікаторі останньої публікації з попереднього зразка сторінок, а initialDateTime – це час початку процесу вибірки, всі наступні сторінки в цьому циклі зразка будуть обмежені цією датою, щоб уникнути читання нових даних, які можуть не відповідати конкретним умовам і ситуації «брудного читання», тобто повторного читання тих самих даних із бази даних.

Усі складні запити та запити з вхідними параметрами були згруповані в процедури та функції. Це знижує дублювання коду. Під час розробки функціональної архітектури програмної системи формуються функції та процеси.

3.2 Реалізація серверної частини програмної системи

Можна почати впроваджувати програмну систему після побудови бази даних і дотримання запропонованої модульної архітектури. При створенні нового проекту WebApi базовий шаблон створюється за допомогою інструментів Microsoft – редактора коду Visual Studio. Startup.cs є основним файлом, який містить всю необхідну логіку, необхідну для запуску програми. Основні параметри проекту зберігаються у файлі appsettings.json. Підключення до бази даних і налаштування сервера Redis, а також розміщення маркера дозволу — все це має терміни. Зазвичай використовуються два файли: один для розробки (appsettings.development.json) і один для розгортання (appsettings.deployment.json) (appsettings.production.json). Перший використовує локальні налаштування, а другий використовує специфічні для хостингу налаштування. Контролери знаходяться в папці Controllers.

Було обрано встановлення бібліотеки NLog, яка використовується для запису помилок та зберігання файлів на фізичному диску, для виявлення будь-яких несправностей у програмному забезпеченні.

Необхідні пакети були сконструйовані відповідно до модульної конструкції, а саме Application – основний модуль програми, який виробляється при запуску нового веб-проекту, Entities - моделі бізнес-логіки сутності. Services є об’єктами рівня служби, тоді як Utils – це додаткова функціональність. Кожен пакет являється відокремленою бібліотекою.

Проектування моделей бізнес-логіки є першим кроком у реалізації сервера. Ці моделі є відображеннями таблиць реляційної бази даних.. Абстрактний клас PostgresEntityBase успадковується всіма класами. Це об’єднуючий клас без атрибутів. Дата формування об’єкта зберігається в класі DatedEntity. Він успадковується всіма класами, які використовують значення дати створення. Усі інші класи мають атрибути, еквівалентні рядкам таблиці бази даних.

Після створення моделі бізнес-логіки був побудований контекст бази

даних – точка доступу для управління запитами до бази даних. Клас PostgresContext представлений контекстом, який виглядає так:

```
public class PostgresContext : DbContext
{
public PostgresContext(DbContextOptions<PostgresContext>
options): base(options)
  {
    ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior
    .NoTracking;
  }
protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)=> optionsBuilder.LogTo(Console.WriteLine);

protected override void OnModelCreating(ModelBuilder modelBuilder)
  {
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<UserFollow>()
      .HasKey(u => new { u.FollowerId, u.FollowingId });

    /*..............*/
  }
  public DbSet<User> Users { get; set; }
  /*..............*/
}
```

Клас ChangeTracker бібліотеки Entity Framework Core пропонує функцію підвищення продуктивності, яка видаляє відстеження змін моделі.

Функція OnConfiguring включає в себе засоби для публікації журналів запитів до бази даних, які виконують EF Core. Це дозволяє виявляти та перезаписувати запити, створені помилково системою ORM, що може мати згубний вплив на продуктивність.

Явні посилання на складені первинні ключі встановлюються для системи ORM у функції OnModelCreating для сутностей зі складеним первинним ключем.

Під час створення запитів до бази даних наведені нижче набори даних заповнюються значеннями з бази даних.

За допомогою підходу до реалізації залежностей створений контекст даних реєструється в головному файлі Startup.cs. Однак, щоб мінімізувати накопичення коду в основному файлі, було вирішено створити всі реєстрації сторонніх служб в окремому файлі Extensions.cs.

Після цього було створено шаблон «Об’єкт запиту», який є частиною сервісного рівня архітектури сервера. Інтерфейс IQueryExecutor і клас, який його реалізує, складають цей пакет. У головному файлі також записується інтерфейс та його реалізація. Це виглядає наступним чином:

```
public interface IQueryExecutor
{
  Task<T> Execute<T>(IQueryObject<T> query, bool autoSave = true);

  Task<T> CachedRead<T>(IQueryObject<T> query, string cacheKey)
    where T : class;

  IQueryable<T> GetBy<T>(Expression<Func<T, bool>> condition)
    where T : PostgresEntityBase;

  Task<T> GetSingleBy<T>(Expression<Func<T, bool>> condition)
    where T : PostgresEntityBase;

  Task<bool> ExistsBy<T>(Expression<Func<T, bool>> condition)
    where T : PostgresEntityBase;

  Task<T> Create<T>(T entity, bool autoSave = true)
    where T : PostgresEntityBase;
   Task CreateFew<T>(IEnumerable<T> entities, bool autoSave =
true)
     where T : PostgresEntityBase;

   Task Update<T>(T entity, bool autoSave = true)
     where T : PostgresEntityBase;

   Task Delete<T>(T entity, bool autoSave = true)
     where T : PostgresEntityBase;
}
```

GetBy, GetSingleBy, ExistsBy, Create, CreateFew, Update, Delete – узагальнені методи із шаблону «Репозиторій», які викликають основні методи обробки операцій з базою даних, включених до EF Core при їх реалізації. Функція Execute використовується для виконання нестандартних запитів до бази даних за допомогою спеціальних класів, які називаються об’єктами запиту, які реалізують інтерфейс IQueryObject. Функціональність проведення запитів до бази даних інкапсульована у всіх об’єктах, що реалізують цей інтерфейс. Вони виконуються з використанням таких методів EF Core, як FromSqlInterpolated і ExecuteSqlInterpolated. Оскільки запит виражається в чистому SQL, а не автоматично створюється фреймворком ORM, їх використання дає вам повний

контроль над ним. Це дозволяє керувати даними в базі даних з більшою гнучкістю, оскільки ви можете виконувати запити, які не підтримуються основними інструментами EF Core, наприклад, запити на виклик функцій або процедур, або створювати таблиці (наприклад, повнотекстові пошукові запити, що містять оператори лише для PostgreSQL).

Далі наведено приклад об’єкта запиту оновлення профільного зображення у обліковому записі користувача:

```
public readonly struct UpdateUserPicture : IQueryObject<int>
{
    private readonly Guid id;
    private readonly string pictureBlobKey;

public UpdateUserPicture(Guid id, string pictureBlobKey = null)
{
    this.id = id;
    this.pictureBlobKey = pictureBlobKey;
}
 public async Task<int> Execute(PostgresContext context)
 {
   FormattableString sql = $@"UPDATE ""User"" SET ""PictureKey"" =
   {pictureBlobKey} WHERE ""UserId"" = {id}";

   return await context.Database.ExecuteSqlInterpolatedAsync(sql);
 }
}

await executor.Execute(new UpdateUserPicture(CurrentUserId,
fileName));
```

Було обрано використовувати структури типу "значення", а не "посилання" як об'єкти запиту замість класів. Це вказує на те, що новий екземпляр структури буде багатошаровим, а скоріше складеним, як класи. Як наслідок, не буде зайвих викликів до збирача сміття, що покращить швидкість роботи програми.

Окремий проект Contract містить усі інтерфейси. Структура пакетів цього проекту відображає модульну конструкцію програми.

Менеджери були сформовані для виконання більш широких обов'язків системи. Це класи в сервісному рівні модульної архітектури.

LoggerManager – це компонент, який використовується для відстеження потенційних помилок.

Токен авторизації доступу та токен оновлення генеруються та

налаштовуються за допомогою AuthManager. Тут встановлюється термін дії токена доступу. З міркувань безпеки встановлено 15-хвилинний час. Оскільки він використовується для перевірки автентичності токена доступу, час оновлення токена оновлення може бути довшим (7 днів). Ідентифікатор користувача та роль включені в зашифровані дані токена під назвою Payload, які визначають, які частини інтерфейсу мають бути захищені, а які доступні на стороні клієнта.

MemoryCacheManager – надає оперативну пам’ять Redis для керування тимчасовим сховищем. Включаються основні методи доступу до даних, включаючи читання, оновлення та видалення даних.

AWSS3BucketManager – керує файлами в хмарному сховищі Amazon S3. Файли конфігурації appsettings.json надають інформацію про дозвіл, необхідни1 для безпечного доступу до S3.

Використовуючи залежності реалізації, всі менеджери реєструються в основному класі програми.

Функціональна архітектура програмної системи реалізована після попередніх дій. Для задоволення корпоративних потреб розроблена система контролерів. Кожен контролер взаємодіє з однією або кількома моделями бізнес-логіки. Реалізовано логіку даних та кінцеві точки кожного методу. Зазвичай інший рівень, який називається Business Services Layer, відокремлює всю цю логіку від контролерів, але було вибрано не робити цього, оскільки це погіршить продуктивність програми.

Метод контролера з назвою ActionMethod приймає клієнтський запит і відповідає даними та номером статусу, який вказує, чи був запит клієнта успішним. Для методів з параметрами, які забороняють клієнтам подавати порожні значення, додано атрибут. Папка Attributes містить усі атрибути, надані в модулі Application. Щоб запобігти надмірним викликам, кілька методів надають атрибути, які встановлюють максимальну кількість запитів на хвилину.

Аргументи методу в контролері часто є об’єктами, надісланими клієнтом для оновлення або отримання ресурсу, які не відповідають структурі моделі бізнес-логіки. Крім того, рішення може відрізнятися від класів об’єктної моделі.

В результаті був створений шаблон DTO (Data Transfer Object). Він являє собою тимчасовий об’єкт, який або перетворюється на модель для зберігання бази даних, або при необхідності доставляється клієнту. DTO було перетворено в модель бізнес-логіки і навпаки за допомогою сторонньої бібліотеки Automapper.

Вхід, вихід і запит – це три види DTO, які часто зустрічаються у відповідних пакетах. При передачі даних клієнту вхідні об’єкти є класами, які надсилаються як аргументи методам контролера. Вони серіалізовані в класи моделі бізнес-логіки та зберігаються в базі даних. На відміну від цього, вихідні об’єкти – це класи, які доставляються клієнту як відповідь. У них є дані з бази даних. Об’єкти запиту, також відомі як об’єкти запиту, – це класи, надані клієнтом як вхідні об’єкти, але вони не впливають на базу даних; замість цього вони визначають вимоги до конкретного зразка запиту з бази даних. Прості об’єкти та об’єкти запиту на розбивку сторінок – це два типи об’єктів запиту. Для таких об’єктів був створений базовий абстрактний клас BasePaginationRequest, який містить три атрибути, які характеризують зразок запиту розбиття на сторінки. Ось як це робиться:

```
public abstract class BasePaginationRequest<T> where T : struct
{
   public T? IdMark { get; set; }

   [Required]
   [Range(3, 1000)]
    public int PagingLimit { get; set; }

    public virtual bool FirstPagination() => IdMark == null;
}
```

IdMark – це поле, яке представляє унікальний ідентифікатор кінцевого значення попереднього кроку пагінації. Цим значенням визначається наступний блок розбиття на сторінки. Обмеження вибірки даних визначається PaginLimit. Також є техніка FirstPagination, яка оцінює, чи є крок розбиття на сторінки першим у циклі вибірки та чи потрібно враховувати всі вищезазначені вимоги.

Клас DateTimeBasePaginationRequest, який розширює базовий клас розбиття на сторінки, розширюється всіма DTO, які використовують розбиття на сторінки на основі об’єктів. Нижче наведено, як це було реалізовано:

```
public class DateTimeBasedPaginationRequest<T> :
BasePaginationRequest<T> where T : struct
{
  [Required]
  [StringLength(maximumLength: 256, MinimumLength = 2)]
  public string SearchTerm { get; set; }

  public DateTime? InitialDateTime { get; set; }

  public DateTime? DateTimeMark { get; set; }

  public override bool FirstPagination()
            => base.FirstPagination() || InitialDateTime == null
||
               DateTimeMark == null;
}
```

InitialDateTime – відповідно до запитів, ініційованих клієнтом, це значення представляє час початку циклу, який може включати багато сторінок. Це ускладнить отримання даних, які були створені після того, як користувач запустив цикл отримання даних.

DateTimeMark – показує час публікації останнього запису попередньої сторінки для визначення наступного.

Пошуковий запит користувача представлений SearchTerm.

Усі об’єкти запиту DTO містять виключно інструкції для надсилання даних користувачеві. Серед них є фільтри рецептів.

Методи контролера можуть мати як відкритий, так і закритий доступ. В останньому сценарії метод позначений властивістю Authorize, що вказує на те, що він доступний лише для авторизованого користувача. Також є елемент фільтра, який ідентифікує користувача, який зробив запит. Це може бути звичайний користувач або модератор. Модератор має повноваження змінювати категорії, видаляти повідомлення користувачів, які порушують стандарти та рекомендації соціальної мережі, а також визначати людей, які отримали скарги від інших користувачів.

Техніка кешування охоплює кілька методів у контролерах. Техніка рецептури, наприклад, використовує підхід кешування, який кешує дані

протягом обмеженого часу. Для цього використовуйте метод CachedRead, наведений нижче:

```
public async Task<T> CachedRead<T>(IQueryObject<T> query, string
cacheKey) where T : class
{
    T data = await cache.Get<T>(cacheKey, true);

    if (data == null)
    {
         data = await query.Execute(context);
         await cache.Set(cacheKey, data);
    }

    return data;
}
```

Якщо дані кешуються, метод повертає їх, якщо дані не кешуються, метод запитує базу даних і заповнює кеш даними.

Відкладені оновлення даних є ще одним методом кешування. Наприклад, він використовується, щоб відстежувати, скільки людей вподобали вас. Дані спочатку записуються в кеш перед введенням в базу даних, коли користувач виконує цю операцію. Він також використовує метод Cached-Aside для синхронізації даних після того, як вони були кешовані.

Зміни даних відкладаються за допомогою планувальника завдань. Для його реалізації використовувався Quartz.net. Він дозволяє створювати завдання і запускати їх через певні проміжки часу. Для цього була розроблена фабрика об’єктів для ініціалізації як планувальника, так і об’єктів завдання. У наведеному нижче списку є приклад завдання, яке перевіряє користувачів, які не були активними або не ввійшли в систему протягом шести місяців, і видаляє їх:

```
[DisallowConcurrentExecution]
public class MarkUserAsRemovedJob : JobBase
{
   private readonly IQueryExecutor executor;
   private readonly ILoggerManager logger;

   public MarkUserAsRemovedJob(IQueryExecutor executor,
                               ILoggerManager logger)
   {
       this.executor = executor;
```

```
        this.logger = logger;
    }

   public override async Task Execute(IJobExecutionContext
context)
    {
       try
       {
         await executor.Execute(new UpdateMarkUserAsRemoved());
       }
       catch (Exception ex)
       {
          logger.LogError($"{nameof(MarkUserAsRemovedJob)} |
{DateTime.Now} \n {ex.Message}");
       }
     }}
```

Вся логіка виконується у функції Execute, яка надсилає відповідний об’єкт запиту та запускає процедуру бази даних, яка позначає користувачів для видалення. Поточний час надається як параметр і порівнюється з часом останньої активності користувача. Реалізація залежностей реєструє це завдання в контейнері інверсії. Вираз Cron використовується для визначення інтервалу часу для завдань. Тест проводитиметься один раз на день, вночі, коли навантаження на сервер мінімальне. ScheduleServices містить усі відкладені завдання, які також є частиною сервісного рівня модульної архітектури.

Після реєстрації на електронну адресу користувача надходить номер підтвердження. Бібліотека MailKit використовується для надсилання електронних листів. Короткий буквено-цифровий код створюється, зберігається протягом короткого періоду та надається користувачеві через електронну пошту під час реєстрації користувача, відновлення пароля або відновлення облікового запису. Код доступу – це ключ кешу, а значення – хешована електронна пошта, створена за допомогою хеш-бібліотеки BCrypt. Дані перевіряються з тими, що знаходяться в кеші, і знищуються, коли користувач вводить код підтвердження. Ось який вигляд має даний код:

```
[AllowAnonymous]
[HttpPost("sendAccessCode")]
[ServiceFilter(typeof(ValidationFilterAttribute))]
public async Task<IActionResult> SendAccessCode([FromBody]
                              UserForSendingAccessCodeDto
userData)
```

```
{
   string accessCode = Hashing.GenerateCharSet();
   string locale = userData.Locale ?? userData.Language;

   string hashedEmail =
   BCrypt.Net.BCrypt.HashPassword(userData.Email)

   if (await cache.KeyExists(userData.Email))
   {

     ModelState.AddModelError(Constants.AccessCodeAlreadySentKey,
     Constants.AccessCodeAlreadySentMsg);
     return BadRequest(ModelState);

   }

var emailBody = Mailing.GetWelcomeEmailBody(locale, accessCode);

await Mailing.SendEmail(configuration, userData.UserName,
userData.Email, emailBody);

   if (!await cache.KeyExists(accessCode))
    {
      await cache.Set(userData.Email, string.Empty);
      await cache.Set(accessCode, hashedEmail);
    }

return Ok();
}
```

Наповнення листа залежить від місцезнаходження клієнта. Пакет Utils включає всі шаблони для написання листів. Цей пакет також включає інструменти генерації коду доступу, а також усі константи.

3.3 Реалізація клієнтської частини програмної системи

Як рішення для побудови клієнтського розділу веб-додатка (front end) була обрана платформа Angular. При реалізації використовується компонентна модель. Тобто кожен компонент або набір компонентів інтерфейсу користувача. Компонент складається з трьох частин: шаблону розмітки HTML, файлу стилю CSS або іншого файлу на основі CSS, препроцесора та класу компонента TypeScript. Спеціальні директиви фреймворку пов'язують шаблон і клас

компонентів. Усі компоненти групуються в логічно пов’язані модулі, які потім можна об’єднати.

Angular також підтримує використання сервісів на додаток до компонентів. Це унікальні класи, які виконують специфічні для служби функції і можуть бути реалізовані в компонентах із використанням визначеного методу реалізації залежностей. Служби HTTP зазвичай використовуються для прийому запитів і повернення об’єкта, відомого як об’єкт, який можна спостерігати. Необхідно підписатися на цей об’єкт за допомогою методу subscribe, щоб отримати дані з сервера. RxJs бібліотека, включена в структуру, надає ці можливості. Це бібліотека реактивного програмування, яка дозволяє використовувати декларативні міркування для організації роботи з подіями та асинхронним кодом. Вона створений із шаблоном дизайну Observer, який дозволяє певним речам спостерігати та реагувати на події в інших елементах. Ця бібліотека містить кілька корисних методів для обробки підписок і оновлення даних.

Визначення базових моделей і створення базових сервісів є першими кроками в реалізації клієнтської складової. Для поділу всіх моделей використовувалися моделі запиту та відповіді. Перший відповідає за створення запитів сервера, а останній відповідає за серіалізацію відповіді сервера. Інтерфейси є у всіх моделях. Наприклад, парадигма зберігання розшифрованих даних мтокенів доступу виглядає так:

```
export interface IUserAuthResponseDto {
  userId: string,
  userType: UserType
}
```

Він містить такі параметри, як ідентифікатор і тип користувача. Тип Response має свою модель. Моделі запитів розробляються так само, як і DTO запитів.

Модель маршрутизації створюється після створення моделей бізнес-логіки. Це метод маршрутизації в програмі, який дозволяє направляти запити до веб-програми, який містить певні ресурси. У цьому веб-додатку було вирішено побудувати два маршрути: один для звичайних користувачів, а інший для

модераторів. Щоб звичайний користувач не мав доступу до панелі адміністратора, був створений клас під назвою GuardObject, який забороняє доступ до адміністративного розділу програмного забезпечення. Він отримує інформацію від токена доступу та визначає, які користувачі мають доступ до яких розділів.

Потім було реалізовано сервіси. Вони поділяються на дві категорії: Служби HTTP безпосередньо відповідають за надсилання запитів до сервера та обробку відповідей. Оскільки сервер доставляє те саме повідомлення, коли виникає проблема зі статусом 500, був створений клас для перехоплення відповідей та їх обробки до того, як вони ввійшли в службу. Аналогічно, перехоплювач зворотної обробки відповідає за встановлення спеціального заголовка, що містить токен, який є тоеном доступу, який надається серверу як заголовок запиту, щоб отримати дані захищених ресурсів. Цей перехоплювач створює такий заголовок і доставляє його на сервер для кожного дозволеного клієнтського запиту. Нижче показано частину коду:

```
@Injectable({ providedIn: 'root' })
export class UserService {
   constructor(
      private httpClient: HttpClient,
      private tokenSettingsService: TokenSettingsService) {}

    getUserInfo(): Observable<ICurrentUserResponse> {
         return this.httpClient.get<ICurrentUserResponse>
                (environment.baseApiUrl + ApiUrls.UserInfo) }
}
```

Властивість Injectable вказує, що послуга може бути введена в будь-який модуль програми. Функція getUserInfo надсилає серверу об’єкт відповідного запиту. Оскільки коренева адреса сервера на хостингу буде відрізнятися від локальної адреси, вона зберігається у файлах середовища. Існує також перелік, який містить усі кінцеві точки сервера.

Служби помічників – це інша форма обслуговування. У додатку вони виконують допоміжні функції. До них належать такі функції, як служба сповіщень, яка відображає вікно сповіщень для користувача, і служба локального зберігання, яка взаємодіє з локальним сховищем веб-браузера. В м

Служби сповіщень будуть виступати інструментом взаємодії кінцевого користувача із системою «Розумний будинок». Тобто це означає що зі сторони клієнтського додатку не потрібно ніякої додаткової реалізації сервісів чи застосування сторонніх бібліотек для взаємодії із зовнішнім світом оскільки система свовіщень налаштована таким чином, що відбувається реакція веб-клієнту на подію із веб-серверу самого додатку, який в свою чергу обробляє інформацію зі сторонньої системи.

Store Services – це інша форма обслуговування. Вони відповідають за загальний стан данихс системи, оскільки, коли компонент видаляється з дерева DOM, усі пов’язані з ним дані також видаляються. Ці служби також відповідають за доступ до даних з будь-якого компонента. Веб-додаток реалізовує єдиний глобальний сервіс для зберігання даних користувача, до якого мають бути доступні всі компоненти.

Після реалізації сервісів відповідно до проекту інтерфейсу виконується проектування та створення компонентів та модулів програми. Опис основних компонентів подано далі.

Nav Bar – компонент для панелі навігації.

Search Input – вказує поле для пошуку. Під час введення тексту користувачеві надається інформація, пов’язана з пошуковим запитом. Однак, щоб зменшити навантаження на систему, дані передаються не відразу, а після введення тексту. Для цього використовується додатковий сервіс Debouncer.

Main Feed – показує понель коментарів користувачів. Стороння бібліотека Ngx-Scroller надає технології віртуалізації даних для контейнера прокручування. Оскільки дані кешуються і відображається лише видима область, цей підхід дозволяє зберігати велику кількість даних у контейнері прокручування без шкоди для швидкості програми. Він також дозволяє налаштувати події, щоб знати, коли користувач закінчив прокручування. Якщо так, завантажте свіжі дані, використовуючи механізм розбиття сторінок, який зараз діє на сервері. Служба Endless Scrolling була розроблена як допоміжна служба для виконання цих дій.

Filter Panel – панель, що містить усі доступні фільтри. Є кнопки для

застосування та очищення фільтра. Його можна знайти в правій частині програми.

Filter Multiple Select – компонент множинного вибору. Ним представлені усі категорії програми.

Filter Chips – вікно для пошуку складових.

Post Card – пропонує вікно з публікацією, яка містить фотографію страви та основну інформацію про рецепт, а також кількість лайків, збереження та коментарів.

Post Details – панель, яка показує вичерпну інформацію про страву, таку як категорії, інгредієнти, методи приготування, відео на YouTube (якщо є) та коментарі. Віртуалізація також використовується в контейнері коментарів.

Далі реалізовано базовий компонент – діалогове вікно, та форми:

– Login Form – вікно для авторизації зареєстрованого користувача.
– Registration Form – вікно для реєстрації.
– Reset Password Form – вікно відновлення паролю.
– Restore Account Form – вікно відновлення облікового запису.
– Create Post Form – вікно для додавання нового допису.
– User Identity Form – вікно для зміни облікового запису.

При натисненні на кнопку «Додати рецепт» на навігаційній панелі відкривається відповідне діалогове вікно, що показане на рисунку 3.2.

Далі на рисунках 3.2 – 3.4 наведено приклади основних форм.

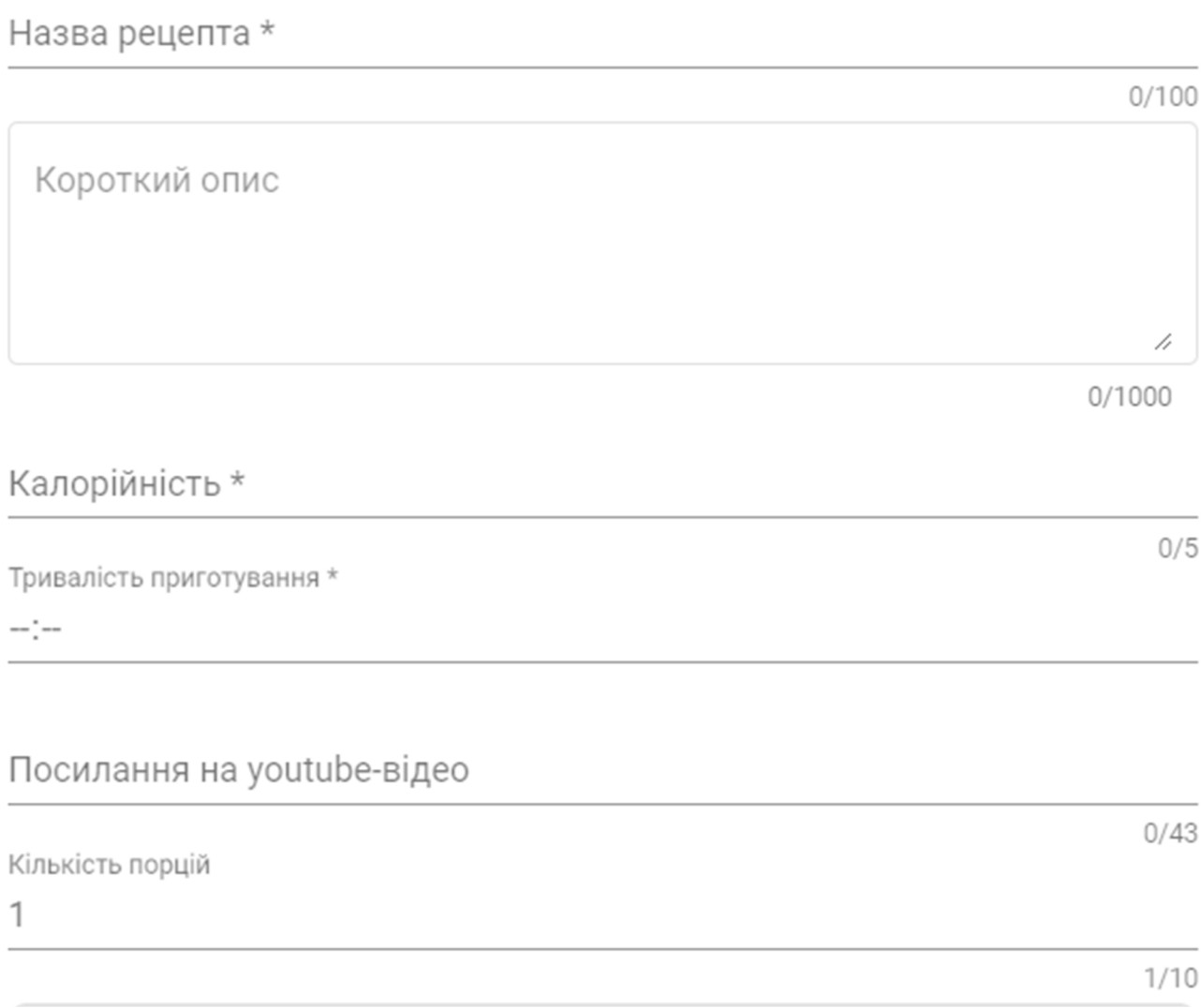


Рисунок 3.2 – Форма створення публікації

Усі натискання на елементи, які бачать лише зареєстровані користувачі, відкриють діалогове вікно входу, показане на рисунку 3.3, якщо користувач не ввійшов у систему.

Якщо користувач забув пароль свого облікового запису, він може змінити його, натиснувши посилання «Змінити пароль».

Якщо користувач ще не зареєстрований у системі, він може зареєструватись натиснуши на посилання «Зареєструватися».

Відкриється вікно реєстрації, що подане на рисунку 3.4.

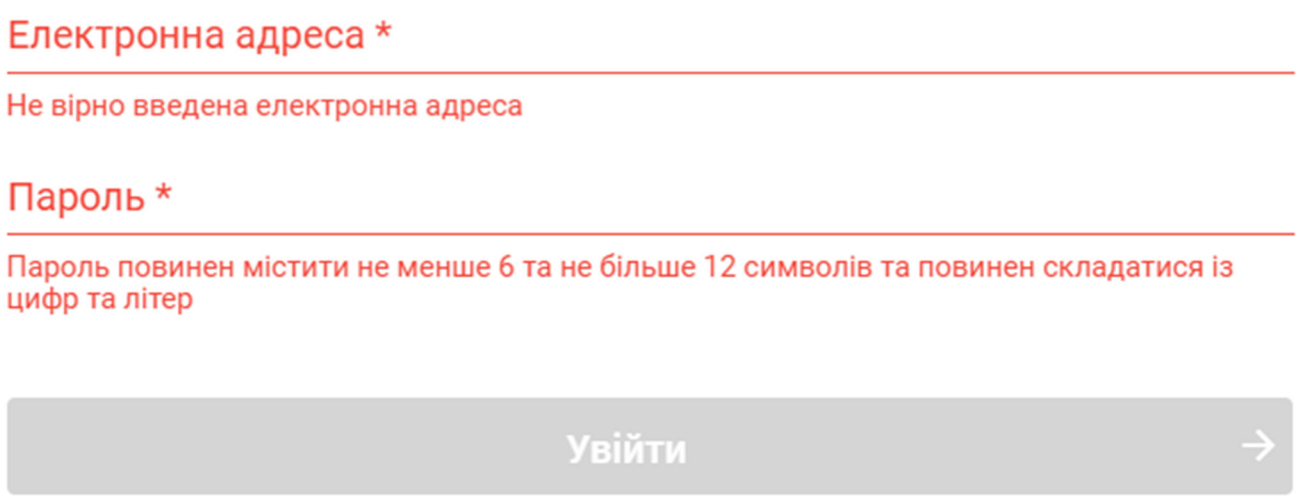


Рисунок 3.3 – Форма авторизації

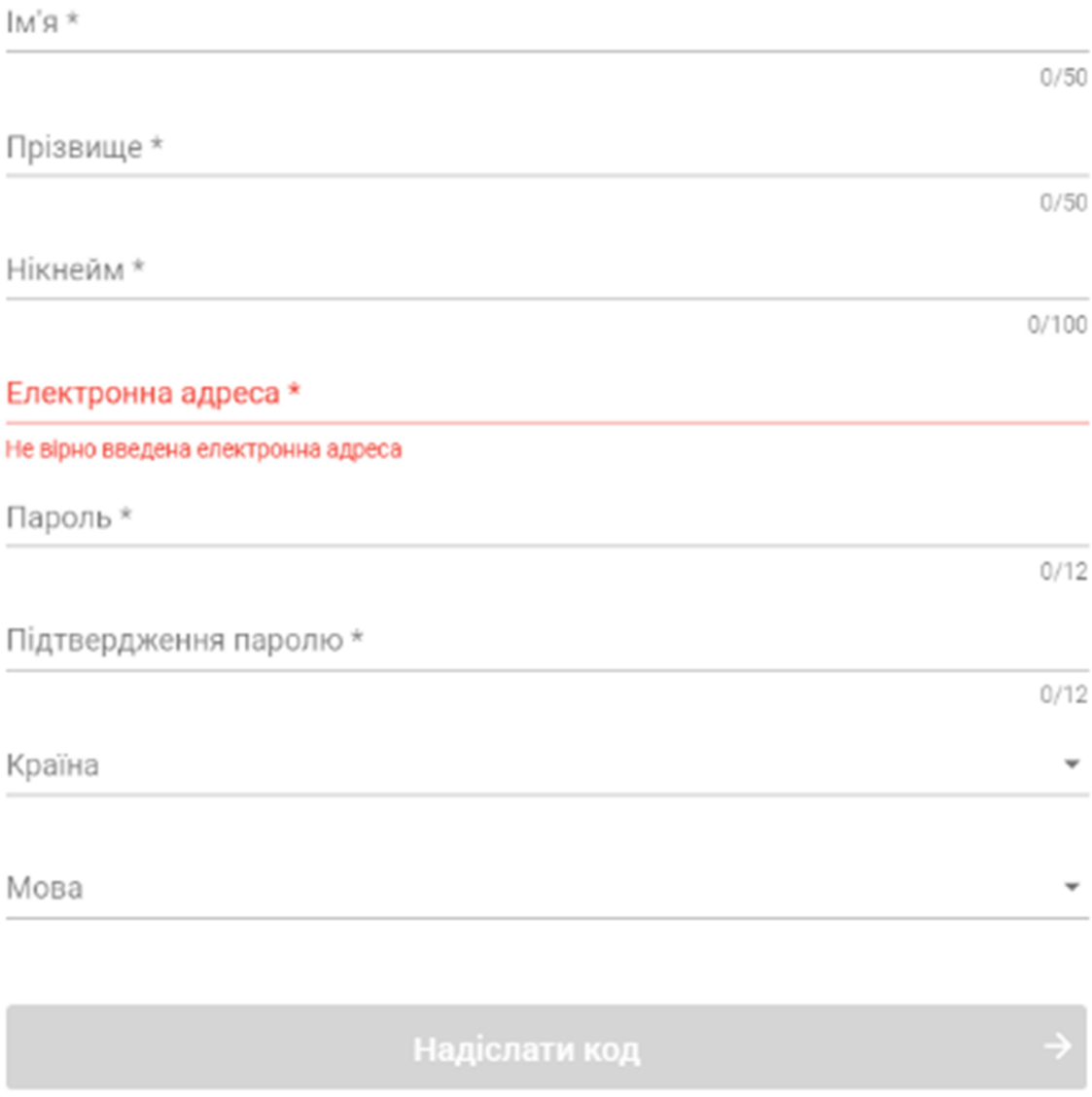


Рисунок 3.4 – Форма реєстрації користувача

Користувач має змогу редагувати облікові дані та змінювати зображення профілю.

Для цього необхідно клікнути на іконку користувача, що знаходиться останньою на навігаційній панелі зліва.

Відкриється модальне вікно, що подане на рисунку 3.5.

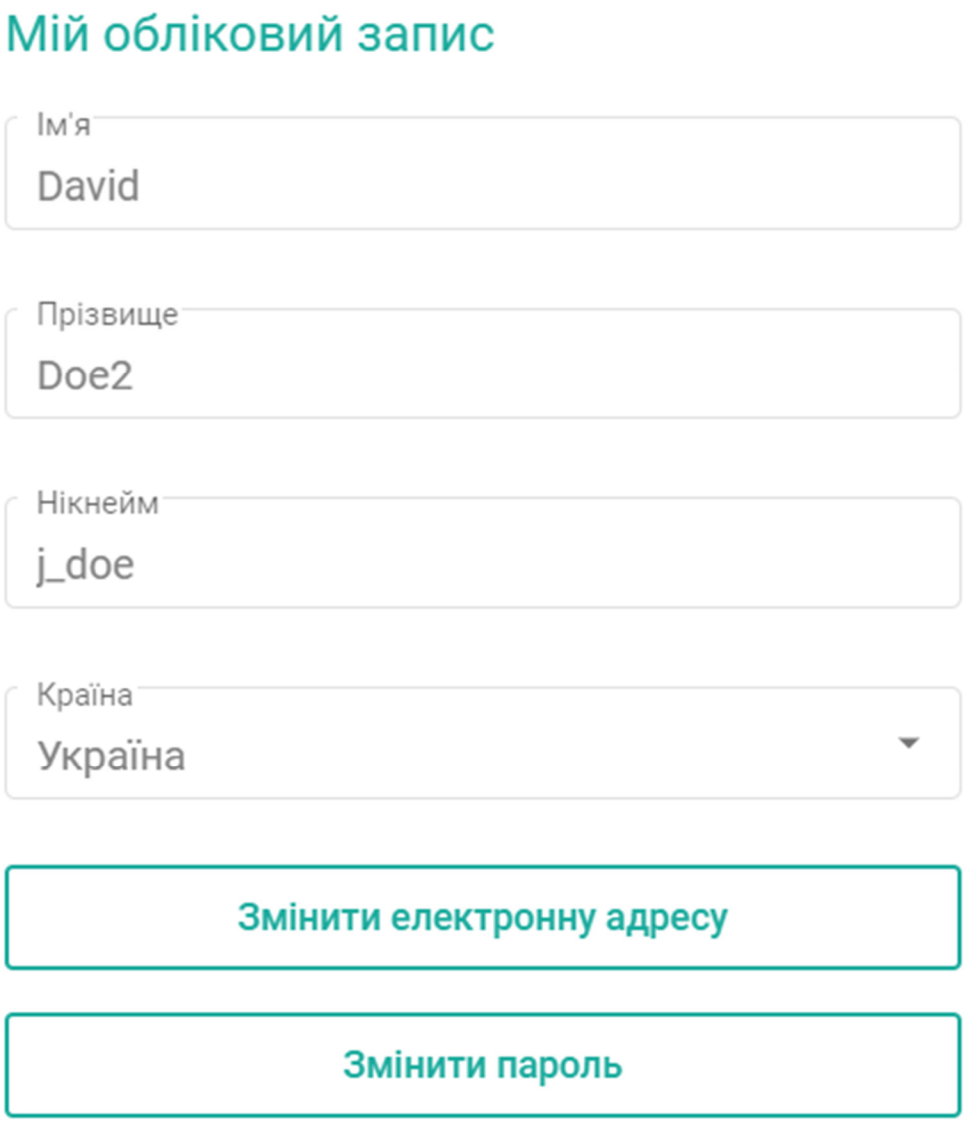


Рисунок 3.5 – Форма редагування облікового запису користувача

Користувач може вийти з облікового запису або видалити його, змінити електронну пошту та пароль.

Повернувшись на головну сторінку, при кліку на зображення або назву рецепту на картці рецепту відкриється вікно із детальною інформацією, що

містить відео, якщо його завантажив користувач, список необхідних та опціональних інгредієнтів, короткий опис рецепту, етапи приготування та категорії, до яких належить рецепт.

Також є можливість перегляду коментарів інших користувачів та створення своїх.

Щоб створити коментар, необіхдно написати текст та натиснути на кнопку із іконкою плюса.

Натискання символу кошика на самому блоці коментарів дозволяє видалити коментарі. Крім того, якщо часто створюються коментарі, система видасть попередження і на короткий час заборонить це зробити.

Проект розгортається за допомогою сервера, коли клієнтська частина завершена, а всі важливі згорнуті файли зберігаються в певній папці на сервері під назвою wwwroot.

3.4 Висновки

В результаті програмна система була впроваджена на етапі проектування та задокументовано ключові етапи її розробки.

Базу даних було побудовано з використанням концептуальної моделі, завершено серверну реалізацію модульної та функціональної архітектури, а також клієнтську частину веб-системи.

4 ТЕСТУВАННЯ ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ РОБОТИ З РЕЦЕПТАМИ

4.1 Аналіз та вибір методів тестування програмного забезпечення

Практика пошуку несправностей програмного забезпечення та оцінки його продуктивності в кількох контекстах розгортання відома як тестування програмного забезпечення. Тестування є важливим елементом процесу розробки програмного забезпечення, оскільки воно допомагає виявляти та усувати помилки.

Функціональне та нефункціональне тестування – це дві форми тестування. Функціональне тестування відповідає за те, щоб усі функції програмної системи працювали належним чином. Так звані тестові випадки (test case) створюються за допомогою діаграми варіантів використання і складаються з набору тестів, які відповідають функціональній діяльності програмної системи.

Тестування безпеки додатків, засноване на фундаментальних концепціях безпеки додатків, також включається в функціональне тестування. Це стосується безпеки, цілісності та доступності даних. Він також перевіряє рівень захисту програми від зловмисних атак, таких як XSS, XSRF, і ймовірних ін’єкцій у кодову базу програми. У результаті тестування безпеки виявлені та усунені критичні вразливості програмного продукту.

Компоненти програми також можна перевірити на сумісність з іншими програмними модулями під час функціонального тестування.

Нефункціональне тестування відноситься до тестів, які використовуються для визначення характеристик програмного забезпечення або того, як система повинна працювати. Тести на навантаження, продуктивність та стрес-тести є прикладами нефункціональних тестів. Усі ці тести оцінюють продуктивність програмної системи та допомагають визначити й усунути місця, де продуктивність програми погіршується.

Модульне тестування гарантує, що окремі відокремлені модулі програмної системи працюють належним чином. Серед іншого вони можуть включати класи бази даних, функції та процедури. Модульне тестування проводиться протягом

життєвого циклу розробки програмного забезпечення.

Загалом тестування системи використовується для підтвердження як функціональних, так і нефункціональних вимог. На програмній системі проводиться комплексне тестування, яке включає тестування окремих компонентів (юніт-тестування), а також пов’язаних компонентів, а також тестування ресурсів системи, комбінацій даних, що вводяться користувачем, зручності використання та інших факторів. Набір тестових випадків, що відповідають системним прецедентам, розвивається протягом такого тестування.

Під час інтеграційного тестування програмного забезпечення приділяється найбільша увага відносинам між компонентами системи. Наприклад, якщо архітектура в стилі сервера є мікросервісами, то кожна із служб спочатку тестується незалежно, а потім перевіряється взаємодія мікросервісів, і чи є тести успішними. Крім того, взаємодія системних модулів оцінюється на двох рівнях: компонентний, тобто оцінка взаємодії компонентів всередині однієї системи, і системний, тобто тестування взаємодії окремих систем, як у мікросервісній архітектурі. Також можна використовувати тестування інтеграції, щоб виявити проблеми з інтерфейсом системи, з операційною системою або апаратним забезпеченням.

Інтеграційне тестування також виконується в іншій послідовності. Існують три види тестування: знизу вгору, зверху вниз і вибуховим. Тестування знизу вгору передбачає організацію та тестування всіх процедур і функцій окремо. Потім створюється наступний рівень: структури та класи програмної системи збираються та тестуються разом. Однак ця стратегія підходить лише тоді, коли майже всі модулі були створені та готові до використання та взаємодії. Краще використовувати тестування зверху вниз для системи з численними компонентами, які знаходяться лише на стадії проектування, тобто не реалізовані. Це досягається шляхом спочатку тестування функціонування модулів верхнього рівня, а потім заміни модулів низького рівня на «заглушки» або об’єкти, які імітують функціональність справжніх модулів. Потім об’єкти-заглушки замінюються справжніми компонентами під час впровадження модулів.

Приймальні випробування є завершальним етапом, і він визначає, чи відповідає система критеріям приймання і чи досягає випробуваний продукт необхідного рівня якості.

Оскільки вбудована система програмного забезпечення є клієнт-серверним додатком, необхідно перевірити як серверні, так і клієнтські компоненти, а також всю систему. Для досягнення цієї мети використовувався підхід до тестування знизу вгору, в якому кожен компонент системи тестується незалежно перед тим, як буде перевірена вся система. При створенні нового програмного модуля це дає змогу переконатися, що всі раніше згенеровані модулі працюють належним чином, що зменшує час, необхідний для тестування системи у разі введення нових логічних компонентів, які потенційно можуть призвести до недоліків програми.

4.2 Тестування серверної частини програмної системи

Оскільки практично всі модулі програми реалізовані, було запропоновано використовувати тестування знизу вгору при тестуванні реалізованої комп’ютерної системи.

Серверна сторона тестується, щоб знайти проблеми в ключовому елементі програми клієнт-сервер, оскільки сервер реалізує основні функції, функції безпеки та алгоритми керування навантаженням.

Тестування проводилося поетапно, починаючи з низькорівневих модулів і переходячи до компонентів високого рівня. Для емуляції реальної бази даних були використані бібліотека NUnit і база даних в пам'яті.

Для всіх тестів у проекті створено окремий каталог Test Projects, який містить проекти для тестування серверної частини функціональності програми. Він також пропонує основні параметри тестового середовища, такі як об’єкти-заглушки та класи, які реплікують поведінку системи Entity Framework ORM. Ці об’єкти називаються «макетами», і їх функціональність еквівалентна функціональності реальних об’єктів. Синглтони використовуються для прикраси

класів обслуговування. Крім того, був створений базовий клас BaseTestClass, який містить усі основні налаштування тесту.

Спочатку було виконане модульне тестування об’єктів запиту. Для цього було створені мок-об’єкти, що симулювали поведінку реальних сутностей бізнес-моделі. Лістинг типового тест-кейсу модульного тестування об’єкту запиту створення користувача подано у лістингу:

```
[Test]
public async Task CreateUsersWithCorrectData()
{
  Guid id = Guid.NewGuid();

  User user = UserMock.Instance.UserForAdd;
  UserIdentity userIdentity = UserMock.Instance.UserIdentityForAdd;
  user.UserId = id;
  userIdentity.Id = id;

  await QueryExecutor.Execute(new CreateUser(userIdentity, user));

  User createdUser = await QueryExecutor.GetSingleBy<User>(x =>
                            x.UserId == id);

  Assert.IsNotNull(createdUser);
  Assert.AreEqual(id, createdUser.UserId);
}
```

Перевірка ідентифікатора новоствореного користувача, який має збігатися з ідентифікатором, створеним раніше, і загалом, якщо в базі даних є щойно створений користувач, є одним із способів визначити, чи ефективно спрацювала процедура. Якщо це так, метод успішно запущено. Базовий клас реєструє QueryExecutor і Context. Структура інших тестів схожа.

Після тестування служб було перевірено контролери або кінцеві точки програми.

Для перевірки роботи контролерів, зокрема кінцевих точок програми, використовується метод моделювання запитів. Це було досягнуто за допомогою фреймворку HttpMock. Він має можливості для обробки змодельованих запитів сервера, але були створені «заглушки» контексту даних і оцінені методи дій у контролерах програмної системи, щоб запобігти зміні даних із бази даних.

4.3 Системне тестування та верифікація програмного забезпечення

Після тестування серверної частини, модульне тестування клієнтської частини та повне автоматизоване тестування системи було реалізовано за подібним прикладом.

Окремі компоненти та функціональні частини, у тому числі як служби, класи трансформаторів, допоміжні класи та класи бізнес-логіки, часто тестуються під час модульного тестування клієнтського розділу програми, що використовує платформу Angular. Модульні тести на клієнті ґрунтуються на тих же міркуваннях, що й модульні тести на сервері. Дані компонента замінюються, і перевіряється, чи він справний і чи виконує свої функції. Його замінено копією, яка не передає дані на сервер, а натомість реплікує поведінку виконання HTTP-запитів за допомогою вбудованих інструментів для служб, які використовують клієнт http. Для проведення модульного тестування було використано основу для проведення модульних тестів Karma та бібліотеку для їх реалізації – Jasmine. Кожен тест починається з блоку опису, який пояснює тестовий акт, за ним слідує, який описує сам тест. Нижче наведено приклад тесту компонента, який відповідає за представлення видавничої картки, яка перевіряє, чи завантажено компонент.

```
describe('PostCardComponent', () => {
   let component: PostCardComponent
   let fixture: ComponentFixture<PostCardComponent>


beforeEach(async () => {
    await TestBed.configureTestingModule({
        declarations: [ PostCardComponent ]
}).compileComponents()
       })
```

Наскрізні тести виконуються після завершення модульних тестів. Вони ретельно тестують систему. Щоб мінімізувати зміни даних у типовому середовищі розробки, ми використовуємо окреме середовище, розроблене виключно для тестування. Тестування E2e було реалізовано за допомогою фреймворка Protractor. Gin використовує моделювання дій користувача для

автоматизованого тестування програмної системи. Шаблон оформлення об’єкта сторінки також використовується в автоматизованому тестуванні. Цей клас забезпечує підключення до вузлів дерева html, які можуть бути використані, щоб дізнатися більше про те, що тестується. Для кожного компонента, функціональність якого була протестована на модуль, створюються об’єкти сторінки. Protractor потребує спеціального сервера під назвою Selenium для проведення тестів, який може працювати як локально, так і віддалено.

Після проведення тестування було створено таблицю, що містить загальний вигляд структури тест-кейсів. Вона подана у табл. 4.1.

Таблиця 4.1 – Основні тест-кейси

Вимоги	Модуль додатку	Підмодуль додатку	Вихідні дані	Очікуваний результат
1	2	3	4	5
R1	Модальне вікно	Форма авторизації	1. Клацнути на кнопку авторизації. 2. Ввести логін та пароль 3. Натиснути на кнопку.	1. З’являється модальне вікно та форма авторизації. 2. Валідація даних 3. Вікно закривається, сторінка перезавантажується, доступні усі дії для зареєстрованого користувача в залежності від його ролі.

Продовження таблиці 4.1 – Основні тест-кейси

R2	Модальне вікно	Форма реєстрації	1. Клацнути на кнопку авторизації. 2. Перейти на форму реєстрації по відповідному посиланню 3. Ввести необхідні особисті дані. 4. Натиснути кнопку «Надіслати код». 5. Вставити надісланий код. 6. Натиснути на кнопку «Завершити реєстрацію».	1. З’являється модальне вікно, з’являється форма авторизації, 2. З’являється форма реєстрації. 3. Валідація даних 4. Сервер відправляє код на пошту користувача. 5. Валідація коду 6. Закривається вікно, перезавантажується сторінка, доступні усі дії для зареєстрованого користувача в залежності від його ролі.
R3	Модальне Вікно	Форма публікації рецептів	Підготовка: зберегти картинку страви на робочому столі. 1. Клацнути на пункт меню «Додати рецепт». 2. Заповнити форму основною інформацією про рецепт. 3. Клацнути на кнопку завантаження картинки рецепту. 4. Вибрати підготовлений файл зі списку. 5. Натиснути кнопку «Далі».	1. З’являється модальне вікно та форма з публікацією рецептів. 2. Валідація даних форми основних даних про рецепт. 3. Відкривається вікно із вибором зображення. 4. Зображення відображається у формі. 5. З’являється форма категорій рецепту. 6. Валідація даних. 7. Валідація таблиці інгредієнтів. 8. З’являється форма створення етапів пригоутвання.

Продовження таблиці 4.1 – Основні тест-кейси

1	2	3	4	5
R3	Модальне Вікно	Форма публікації рецептів	6. Заповнити форму категорій рецепту. 7. Заповнити таблицю з інгредієнтами. 8. Натиснути кнопку «Далі». 9. Заповнити форму із етапами приготування. 10. Натиснути кнопку «Завершити публікацію».	9. Валідація даних. 10. Вікно закривається, втсановлюється фільтр «Мої рецепти» та оновлюється стрічка публікацій.
R4	Стрічка публікацій	Збережені рецепти	1. Клацнути на пункт меню «Збережені рецепти».	1. Панель фільтрів закривається, відкривається панель із книгами рецептів
R5	Модальне вікно	Редагування облікового запису	Підготовка: зберегти профільне зображення на робочому столі. 1. Клацнути на пункт меню «Мій обліковий запис». 2. Редагувати облікові дані. 3. Клацнути на кнопку завантаження профільного зображення. 4. Вибрати підготовлений файл. 5. Натиснути кнопку «Зберегти дані».	1. Відкривається модальне вікно із формою редагування облікових даних. 2. Валідація даних 3. Відкривається вікно для вибору зображення. 4. Вибраний файл відображається на формі. 5. Дані посилаються на сервер, вікно закривається та сторінка перезавантажується

Кінець таблиці 4.1 – Основні тест-кейси

R6	Стрічка публікацій	Панель фільтрів	1. Клацнути на кнопку «Показати панель фільтрів». 2. Вибрати фільтри. 3. Натиснути кнопку. «Застосувати фільтри». 4. Натиснути кнопку «Очистити фільтри». 5. Натиснути кнопку «Застосувати фільтри».	1. Відображається панель фільтрів справа. 2. Відображаються відповідні фільтри. 3. Стрічка фільтрується та оновлюється. 4. Поля на панелі фільтрів очищуються. 5. Публікації відображаються у звичайному режимі.
R7	Адмін. панель	Створення категорій	1. Клацнути на кнопку «Мій обліковий запис» 2. Авторизуватися у ролі модератора 3. Вибрати потрібну категорію 4. Ввести дані нової категорії 5. Створити категорію	1. З’являється модальне вікно із формою авторизації. 2. Вікно закривається, відбувається переміщення на сторінку адміністративної панелі. 3. Таблиця із категоріями фільтрується відповідно до вибраної. 4. Валідація даних 5. Дані додаються у таблицю

Оскільки програмна система буде вивантажена на платформі Amazon Beanstalk, було проведено поверхневе тестування, щоб забезпечити належну функціональність програми.

Коли потрібно перевірити лише ключові компоненти програмної системи, це називається «Smoke Testing». Щоб виявити та усунути недоліки у віддаленому середовищі, зазвичай достатньо виконати прості автоматизовані тести.

Після виконання тест-кейсів було створено звіт про результати тестування, що подано у таблиці 4.2.

У таблиці 4.2 наведені деякі із основних результатів проходження тест-кейсів.

Таблиця 4.2 – Звіт про результати виконання деяких тест-кейсів

ID	Опис тест-кейса	Вхідні значення	Реальні вихідні дані	Пройдено
1	2	3	4	5
TC1	Авторизація у системі	1. Логін користувача 2. Пароль користувача	1. Токен доступу та оновлення не пусті та валідні 2. Наявна інформація про користувача вірна	Так, Очікувані та реальні вихідні дані однакові
TC2	Реєстрація користувача	1. Ім’я 2. Прізвище 3. Нікнейм 4. Ел. адреса 5. Пароль 6. Країна	1. Токен доступу та оновлення не пусті та валідні	Так, Очікувані та реальні вихідні дані однакові

Кінець таблиці 4.2 – Звіт про результати виконання деяких тест-кейсів

1	2	3	4	5
ТС2	Реєстрація користувача	7. Мова 8. Код доступу	2. Отримана інформація про користувача не порожня та відповідає даним з форми реєстрації	Так, Очікувані та реальні вихідні дані однакові
ТС3	Публікація рецепту	1. Назва рецепту 2. Короткий опис 3. Калорійність 4. Тривалість Приготування 5. Посилання на відео 6 Додаткова інф. 9. Зображення рецепту 10. Категорії рецепту 11. Інгредієнти 12. Етапи приготування	1. Створена публікація містить усю необхідну інформацію, що відповідає інформації, що створив користувач 2. Встановлений фільтр на відображення тільки рецептів поточного користувача	Так, Очікувані та реальні вихідні дані однакові
ТС1 Inv	Авторизація у системі	1. Неправильний логін користувача 2. Неправильний Пароль користувача	1. Токени доступу не надсилаються 2. Відображається помилка «Користувача не знайдено»	Так, Очікувані та реальні вихідні дані однакові

4.4 Висновки

Отже, б уло проведено тестування із різними наборами даними. Усі тести виконались успішно.

Система також пройшла оптимізаційне тестування, яке передбачає моніторинг навантаження на базу даних та її складові компоненти, такі як таблиці. Для цього ми використали планувальник запитів і перевірили більшість запитів і характеристик продуктивності. Планувальник показує, скільки часу потрібно для виконання запиту, і дає змогу визначити запити, які не працюють належним чином. Деякі запити були покращені на етапі тестування, щоб підвищити швидкість роботи програмної системи.

Також була випробувана робота сховища даних Redis, яка продемонструвала ефективність кешування в додатку.

В результаті тестування впровадженої комп’ютерної системи було виявлено, що всі тести пройшли задовільно. В ході випробувань були виявлені дефекти та ефективно усунені. Основна функціональність програми розроблена відповідно до вимог програмного забезпечення, і більшість з них повністю робочі.

ВИСНОВКИ

Соціальні мережі стають широко використовуваними платформами, які дозволяють користувачам обмінюватися інформацією. Соціальні мережі можуть надавати різноманітні послуги залежно від свого виду, але найважливішими є пошук і розповсюдження інформації. Соціальні мережі вузької тематики об’єднують людей на основі спільних інтересів і зосереджені на одній проблемі. Вони не завжди популярні, але вони можуть бути корисними, особливо якщо допомагають задовольняти основні потреби людини.

Як наслідок, кінцевим результатом дипломного проекту є досліджений та реалізований веб-базований метод автоматизації публікації, розповсюдження та пошуку рецептів, реалізовано веб-базований метод взаємодії із системою управління «Розумний дім». В ході дослідження було основні компоненти системи «Розумний дім»; досліджено існуючі методи побудови систем «Розумний дім». На основі вивчення та детального аналізу у сфері соціальних мереж та програмних систем було визнано ключові переваги та недоліки існуючого програмного забезпечення. Було виявлено, що сьогодні на ринку дуже мало кулінарних соціальних мереж, а доступні програмні рішення – звичайні веб-сайти без специфічних функцій соціальних мереж та не мають особливо привабливого або сучасного інтерфейсу, що підкреслює актуальність теми розробленого програмного забезпечення. На основі отриманих даних було визначено функціональні та нефункціональні потреби програми та надано ключові варіанти використання розробленої системи.

Переваги та недоліки стилів архітектури монолітного та мікросервісного програмного забезпечення були вивчені та визнані на етапі проектування програмного забезпечення. Хоча було продемонстровано, що мікросервіси є надійною альтернативою для побудови потенційно сильно завантажених систем, монолітна серверна архітектура була обрана на основі типу продукту, часу розробки та складності реалізації додатку.

Під час детального проектування було проаналізовано переваги та недоліки кількох типів сховищ даних, а в якості головного сховища було обрано

систему управління реляційною базою даних, керуючись цілісністю, надійністю, узгодженістю та швидкістю доступу до даних.

Сутність системи та їх властивості були виявлені на етапі розробки моделі сутність-відношення, і дані були нормалізовані, а зв’язки зведені до третьої нормальної форми, однак деякі зв’язки були частково денормалізовані, щоб мінімізувати погіршення швидкодії.

Основні характеристики популярних інструментів розробки були визначені під час аналізу поточних систем управління реляційними базами даних, а база даних PostgreSQL була обрана на основі значущих факторів, включаючи швидкість, цінову політику та ряд функцій управління масивами даних.

Також було визначено характеристики трирівневої архітектури, а також системи пакетів даних та об’єктної моделі класів. Визнано ключові переваги та недоліки найпопулярніших шаблонів для створення сервісного рівня програми, і визначено, що шаблон «Об’єкт запиту» є надійним вибором для реалізації завдяки його здатності виконувати нетривіальні запити, гнучкості та простоті тестування.

Необхідні вимоги до програмного забезпечення були оцінені протягом побудови функціональної архітектури, а можливі зони погіршення швидкодії програми були визначені та усунені за допомогою методів управління системами з високим навантаженням. Це було досягнуто завдяки алгоритмам кешування в контексті певних дизайнерських рішень, використовуючи систему тимчасового зберігання Redis.

Під час виконання проекту користувальницького інтерфейсу, а також порівняння сучасних технологій побудови клієнтського компонента програми були визначені ключові частини та модальні вікна програмної системи, а в якості фронтенд-фреймворка обрана платформа Angular.

Під час впровадження програмної системи була побудована база даних відповідно до концептуальної моделі, а також система індексації та базові запити до бази даних. Після цього був реалізований модульний і функціональний дизайн сервера. Основні компоненти та сервіси програми, які відповідають

запропонованій системі модальних вікон і розділів, були розроблені під час створення клієнтської частини веб-додатку.

Основні тестові приклади були створені під час тестування програмного забезпечення, були виявлені та усунені помилки в системі, а в результаті надано звіт про результати тестування. Проведено модульне тестування серверів і клієнтів, а також тестування всієї системи за допомогою автоматизованих тестів.

Як наслідок кваліфікаційної роботи було створено програмну систему для автоматизації публікації, розповсюдження та пошуку рецептур різних страв. Фундаментальна функціональність програми побудована відповідно до специфікацій програмного забезпечення, а результати практичних випробувань підтверджують ефективність програми.

Об’єктом дослідження є процес автоматизації роботи з рецептами.

Предметом дослідження є метод та технологія автоматизації роботи з рецептами.

Метою кваліфікаційної роботи є підвищення ефективності та рівня автоматизації роботи з рецептами за рахунок інтеграції в реальний світ за допомогою фізичних отримувачів інформації з оточуючого світу.

У кваліфікаційній роботі проведено аналіз соціальних мереж, було порівняно монолітний і мікросервісний стиль розробки, розглянуто методології та інструменти для високонавантаженого системного адміністрування та рекомендовані шляхи їх покращення, було розглянуто використання реляційних баз даних і баз даних NoSQL, а також їх переваги та недоліки. Характеристики front-end фреймворків були визначені під час побудови клієнтського компонента системи. Система напряму взаємодіє з «Розумним будинком», що дає можливість використовувати соціальну мережу у помешканні як єдиний механізм, який автоматизує пошук страв та буде синхронізований з процесами, які кожна людина виконує щодня.

Наукова новизна отриманих результатів:

1) набули подальшого розвитку метод та технологія автоматизації публікації, поширення та пошуку рецептів за рахунок використання фізичних

давачів, що допомагають зменшити витрати часу на базові людські потреби, підвищити ефективність та скоротити час приготування їжі.

Практична цінність отриманих результатів. В результаті виконання кваліфікаційної роботи були розроблені метод та технологія автоматизації роботи з рецептами, а також методи налаштування системи «Розумний будинок» для синхронізації її із соціальною мережею та наташтування веб-серверу для аналізу даних, отриманих із реального світу.

Результатом вирішення завдань кваліфікаційної роботи є повноцінна комп'ютерна система для автоматизації, публікації та поширення кулінарних рецептів, яка має інтеграцію з системою «Розумний будинок».

За темою кваліфікаційної роботи опубліковано тези доповіді та взято участь у Всеукраїнській науково-практичній конференції «Актуальні проблеми комп'ютерних наук», що проходила 15-16 жовтня 2021 р. в Хмельницькому національному університеті:

1) Рей К., Ковтонюк І., Грищук І. Дослідження методів керування ресурсами кіберфізичної системи «Розумний будинок». Збірник наукових праць за матеріалами Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук» АПКН–2021 (Хмельницький, 15-16 жовтня 2021). С. 191-193.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЬ

1. Lucas L. JavaFx Special Effects.London: *Appress*, 2017. – 432 с.
2. Pajankar A. Raspberry Pi Supercomputing and Scientific Programming. Munich: *Appress*, 2017. – 234 с.
3. Bloch J. Effective Java. London: Addison-Wesley, 461.
4. Рецепти. Кулінарні рецепти з фото: перші і другі блюда, салати, випічка, десерти, коктейлі. URL: http://cook.i.ua (дата звернення: 15.11.2021).
5. Elise Bauer.Simply Recipes. URL: https://www.simplyrecipes.com (дата звернення: 15.11.2021).
6. Клієнт-серверна архітектура. URL: https://training.qatestlab.com/ blog/technical- articles/client -server-architecture (дата звернення: 15.11.2021).
7. Larry Kerschberg. dblp: Advanced Transaction Models and Architectures. Computer Science bibliography. URL: https://dblp.uni-trier.de/db/books/collections/JajodiaK97.html (дата звернення: 15.11.2021).
8. Server Architectures .Concurrent Programming for Scalable Web Architectures. URL: http://berb.github.io/diploma-thesis/original/042_serverarch.html (дата звернення: 15.11.2021).
9. S. Sumathi, S. Esakkirajan. Fundamentals of Relational Database Management Systems. M.: Springer-Verlag, Berlin, Heidelberg, 2007.
10. What is NoSQL? Nonrelational Databases, Flexible Schema Data Models URL: https://aws.amazon.com/nosql/?nc1=h_ls (дата звернення: 15.11.2021).
11. What Is a Graph Database? URL: https://aws.amazon.com/nosql/graph/?nc1=h_ls (дата звернення: 15.11.2021).
12. Mark Drake A Comparison Of Relational Database Management Systems URL:https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems (дата звернення: 15.11.2021).
13. About PostgreSQL URL: https://www.postgresql.org/about.
14. .NET Fundamentals URL: https://docs.microsoft.com/en-us/dotnet/fundamentals (дата звернення: 15.11.2021).

15. Introduction to ASP.NET Core URL: https://docs.microsoft.com/en-us/aspnet/core/introduction-to aspnetcore?view=aspnetcore-5.0 (дата звернення: 15.11.2021).

16. Atul Gupta, Sudhanshu Hate, Andrew Siemer. ASP.NET 4 Social Networking . – M.: Packt Publishing, 2011.

17. Jon P Smith. Is the repository pattern useful with Entity Framework Core? The Reformed Programmer. URL:https://www.thereformedprogrammer.net/is-the-repository-pattern-useful-with-entity-framework-core (дата звернення: 15.11.2021).

18. Caching strategies. Amazon ElastiCache. URL: https://docs.aws.amazon.com/AmazonElastiCache/latest/mem-ug/Strategies.html (дата звернення: 15.11.2021).

19. Redis Documentation URL:https://redis.io (дата звернення: 15.11.2021).

20. Pros and Cons of ReactJS. javatpoint. URL: https://www.javatpoint.com/pros-and-cons-of-react (дата звернення: 15.11.2021).

21. Pros and Cons of Angular Development Framework URL: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of- angular-development (дата звернення: 15.11.2021).

22. Typed JavaScript at Any Scale URL: https://www.typescriptlang.org (дата звернення: 15.11.2021).

23. What is Angular? URL:https://angular.io/guide/what-is-angular (дата звернення: 15.11.2021).

24. The RxJS library URL:https://angular.io/guide/rx-library (дата звернення: 15.11.2021).

25. Different Testing Types with Details [Electronic resource] / Types of Software Testing. – Available from: https://www.softwaretestinghelp.com/types-of-software-testing (дата звернення: 15.11.2021).

26. Protractor – end-to-end testing for AngularJS URL: https://www.protractortest.org. (дата звернення: 15.11.2021).

27. An Overview of Home Automation Systems URL: https://ieeexplore.ieee.org/document/7791223/.

28. Granzer W. P. Security in Building Automation Systems Munich: *Appress*, 2018. – 578 c.

29. Dragoicea M. A Service Oriented Simulation Architecture for Intelligent Building Management. . London: *Appress*, 2017. – 654 c.

30. Carnell J. Spring Microservices in Action . Munich: *Appress*, 2017. – 384 c.

31. Dewailly L. Building a RESTful Web Service with Spring .Munich: *Appress*, 2017. – 675 c.

32. Moises M. Learn Microservices with Spring Boot. London: *Appress*, 2017. – 385 c.

33. Bell C. Beginning Sensor Networks with Arduino and Raspberry Pi Munich: *Appress*, 2017. – 576 c.

34. Pethuru R. The Internet of Things. Munich: *Appress*, 2017. – 365 c.

35. Nazarko S. Raspberry Pi Media Center. London: *Appress*, 2018. – 108 c.

36. P. Jain, A. Sharma and L. Ahuja, The Impact of Agile Software Development Process on the Quality of Software Product, *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2018, pp. 812-815, doi: 10.1109/ICRITO.2018.8748529.

37. M. Kuhrmann et al., Hybrid Software Development Approaches in Practice: A European Perspective, *IEEE Software*, vol. 36, no. 4, pp. 20-31, July-Aug. 2019, doi: 10.1109/MS.2018.110161245.

38. T. Tekbulut, N. Canbaz and T. Ö. Kaya, Machine Learning Application in LAPIS Agile Software Development Process, *Turkish National Software Engineering Symposium (UYMS),* 2020, pp. 1-6, doi: 10.1109/UYMS50627.2020.9247069.

39. N. Ramasubbu and C. Kemerer, Integrating Technical Debt Management and Software Quality Management Processes: A Framework and Field Tests, *IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 883-883, doi: 10.1145/3180155.3182529.

40. V. Augustine, J. Hudepohl, P. Marcinczak and W. Snipes, Deploying Software Team Analytics in a Multinational Organization, *IEEE Software*, vol. 35, no. 1, pp. 72-76, January/February 2018, doi: 10.1109/MS.2017.4541044.

41. M. Singh, N. Chauhan and R. Popli, A Framework For Transitioning Of Traditional Software Development Method To Distributed Agile Software Development, *International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT),* 2019, pp. 1-4, doi: 10.1109/ICICT46931.2019.8977654.

42. P. Hohl, M. Stupperich, J. Münch and K. Schneider, An Assessment Model to Foster the Adoption of Agile Software Product Lines in the Automotive Domain, *IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC),* 2018, pp. 1-9, doi: 10.1109/ICE.2018.8436325.

43. S. A. Ruk, M. F. Khan, S. G. Khan and S. M. Zia, A survey on Adopting Agile Software Development: Issues & Its impact on Software Quality, *IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS),* 2019, pp. 1-5, doi: 10.1109/ICETAS48360.2019.9117324.

44. A. Iftikhar, S. Musa, M. Alam, M. M. Su'ud and S. M. Ali, A survey of soft computing applications in global software development, *IEEE International Conference on Innovative Research and Development (ICIRD),* 2018, pp. 1-4, doi: 10.1109/ICIRD.2018.8376330.

45. J. Berglind Söderqvist, L. Lindlöf and L. Trygg, Inter-Team Coordination in Agile Development: Learning from Non-Software Contexts, *IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE),* 2019, pp. 69-70, doi: 10.1109/CHASE.2019.00024.

46. N. Rashid, S. U. Khan, H. U. Khan and M. Ilyas, Green-Agile Maturity Model: An Evaluation Framework for Global Software Development Vendors, *IEEE Access*, vol. 9, pp. 71868-71886, 2021, doi: 10.1109/ACCESS.2021.3079194.

47. R. Biddle, A. Meier, M. Kropp and C. Anslow, Poster: Sources of Satisfaction in Agile Software Development, *IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion),* 2018, pp. 333-334.

48. F. Hayat, A. U. Rehman, K. S. Arif, K. Wahab and M. Abbas, The Influence of Agile Methodology (Scrum) on Software Project Management, *20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence,*

Networking and Parallel/Distributed Computing (SNPD), 2019, pp. 145-149, doi: 10.1109/SNPD.2019.8935813.

49. J. Kumar and V. Garg, Security analysis of unstructured data in NOSQL MongoDB database, *International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN),* 2017, pp. 300-305, doi: 10.1109/IC3TSN.2017.8284495.

50. A. Sharma and P. Kaur, A Multitenant Data Store Using a Column Based NoSQL Database, *Twelfth International Conference on Contemporary Computing (IC3)*, 2019, pp. 1-5, doi: 10.1109/IC3.2019.8844906.

51. V. Sachdeva and S. Gupta, Basic NOSQL Injection Analysis And Detection On MongoDB, *International Conference on Advanced Computation and Telecommunication (ICACAT),* 2018, pp. 1-5, doi: 10.1109/ICACAT.2018.8933707.

52. B. F. P. de Oliveira, M. d. C. Victorino and M. Holanda, Data Warehouse Based on NoSQL: a literature mapping, *16th Iberian Conference on Information Systems and Technologies (CISTI),* 2021, pp. 1-6, doi: 10.23919/CISTI52073.2021.9476293.

53. S. Wang, Y. Zhu, F. Wang and Z. Qian, The selection of NoSQL database in E-government system of China meteorological administration, *IEEE 2nd International Conference on Big Data Analysis (ICBDA),* 2017, pp. 105-109, doi: 10.1109/ICBDA.2017.8078786.

54. R. Macedo et al., A Practical Framework for Privacy-Preserving NoSQL Databases, *IEEE 36th Symposium on Reliable Distributed Systems (SRDS),* 2017, pp. 11-20, doi: 10.1109/SRDS.2017.10.

55. Jaglale J. Sping 4 Cookbook / Jerome Jaglale. – Munich: Pack Publishing, 2014. – 234 c.

56. Bloch J. Effective Java / Joshua Bloch. – London: Addison-Wesley, 461.

ДОДАТОК А

(обов’язковий)

КОПІЇ ТЕЗ ДОПОВІДІ НА КОНФЕРЕНЦІЇ

1) Рей К., Ковтонюк І., Грищук І. Дослідження методів керування ресурсами кіберфізичної системи «Розумний будинок» // Збірник наукових праць за матеріалами Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук» АПКН–2021 (Хмельницький, 15-16 жовтня 2021). С. 191-193.

УДК 004.031.42: 004.896

Рей К. С., Ковтонюк І. П., Грищук І. І.

Хмельницький національний університет

ДОСЛІДЖЕННЯ МЕТОДІВ КЕРУВАННЯ РЕСУРСАМИ КІБЕРФІЗИЧНОЇ СИСТЕМИ «РОЗУМНИЙ БУДИНОК»

Кіберфізична система «Розумний будинок» має забезпечувати комфорт, безпеку та ресурсозбереження, а також розпізнавати конкретні події та реагувати на них. В роботі проведено дослідження відомих методів керування ресурсами (зокрема, освітленням та контролем витоку води і газу) кіберфізичної системи «Розумний будинок».

The cyberphysical system "Smart Home" must provide comfort, security and resource conservation, as well as recognise and respond to specific events. The study of known methods of resource management (in particular, lighting and control of water and gas leakage) of the cyberphysical system "Smart Home" was conducted.

Кіберфізична система «Розумний будинок» дозволяє звільнити користувачів від левової частки побутових проблем. Це ціла інфраструктура, що перетворює середньостатистичну житлову площу в автономну екосистему.

Кіберфізична система «Розумний будинок» здатна: регулювати опалення та водопостачання; керувати освітленням житлової площі; забезпечувати охорону і відоспостереження; контролювати витік води і газу; підтримувати необхідний рівень вологості та температури в приміщеннях; керувати побутовою технікою [1-3].

Метою роботи є дослідження відомих методів керування ресурсами (зокрема, освітленням та контролем витоку води і газу) кіберфізичної системи «Розумний будинок».

В загальному вигляді структуру підсистеми освітлення кіберфізичної системи «Розумний будинок» зображено на рисунку 1 [4]. Така підсистема складається з модуля керування освітленням, модуля виявлення присутності, модуля контролю освітленням та модуля керування розетками, які об'єднані між собою за допомогою комутаційного середовища.

Модуль керування освітленням забезпечує ручне та автоматичне локальне керування освітленням. Модуль виявлення присутності визначає присутність людини в приміщенні та ввімкнення освітлення лише в тих приміщеннях, де помічена присутність. Модуль контролю освітленості вимірює природнє освітлення і вмикає освітлення лише в тих місцях, де є брак природного освітлення. Модуль управління розетками забезпечує можливість керування електроприладами з використанням безпровідної технології. Описана підсистема освітлення

кіберфізичної системи «Розумний будинок» керується з мобільного телефону за допомогою спеціального мобільного додатку.

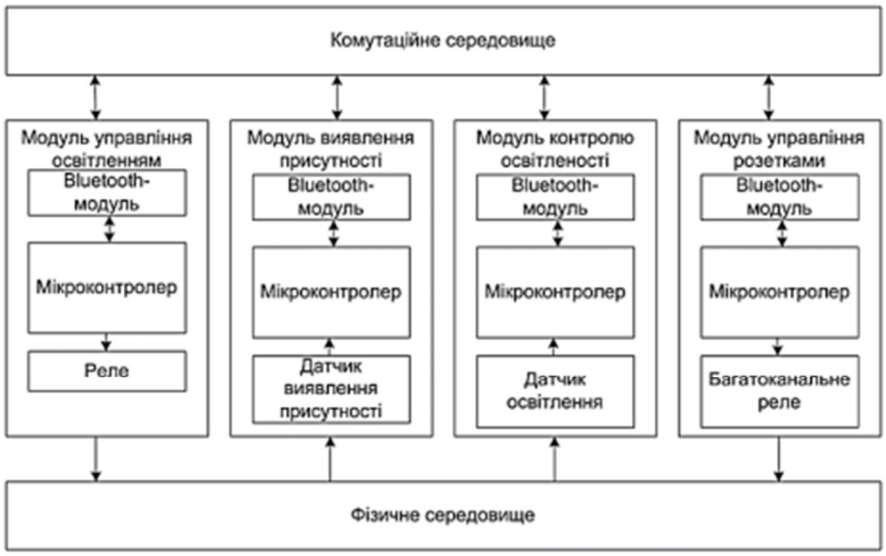


Рисунок 1 – Загальна структура підсистеми освітлення кіберфізичної системи «Розумний будинок» [4]

Рисунок 2 – Схема використання датчиків протікання води у підсистемі контролю витоку води і газу в кіберфізичній системі «Розумний будинок» [5]

Підсистема контролю витоку води і газу в кіберфізичній системі «Розумний будинок» забезпечує можливість закривати/відкривати клапани водо- і газопостачання при виявленні протікань в процесі контролю якості повітря, тобто автоматично блокувати подачу води та/або газу. Крім цього, така підсистема забезпечує правильну вентиляцію приміщень, дозволяє ввімкнення системи світло- та звукооповіщення, передачу сигналу на пульт оперативного чергового, а також передавати сигнал про аварію власнику житлового приміщення на мобільний телефон.

Схема використання датчиків протікання води у підсистемі контролю витоку води і газу в кіберфізичній системі «Розумний будинок» представлена на рисунку 2.

Отже, система «Розумний будинок» має забезпечувати комфорт, безпеку та ресурсозбереження, а також розпізнавати конкретні події та реагувати на них. В роботі проведено дослідження відомих методів керування ресурсами (зокрема, освітленням та контролем витоку води і газу) кіберфізичної системи «Розумний будинок».

Перелік посилань

1. Мельник А. О. Кіберфізичні системи: проблеми створення та напрями розвитку. Вісник Нац. ун-ту «Львівська політехніка». 2014. № 806. С. 154–161.
2. Ерфан П.. Микитин Г. Кіберфізичні системи в проектуванні розумних будинків. Студентська науково-технічна конференція : збірник тез доповідей, м. Львів, жовтень 2019 р. Львів, 2019. С. 176–178.
3. Lea P. Internet of Things for Architects: Architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security. Birmingham: Packt Publishing, 2018. 524p.
4. Франков Д.А. Кіберфізична система освітлення «Розумного будинку». Зв'язок. 2019. №5. С. 55-59.
5. Калінін Д. В. Електронні системи розумного будинку з підвищеною ефективністю: дипломна робота на здобуття ступеня бакалавра: 171 / Київ, 2021. 67 с.

ДОДАТОК Б

(обов'язковий)

ПРЕЗЕНТАЦІЯ ДОПОВІДІ

Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

Рей Костянтин Сергійович

Метод та технологія автоматизації роботи з рецептами

Науковий керівник – доктор техн. наук, професор Говорущенко Т.О.

Хмельницький - 2022

Мета і задачі дослідження

- Метою кваліфікаційної роботи є підвищення ефективності та рівня автоматизації роботи з рецептами за рахунок інтеграції в реальний світ за допомогою фізичних отримувачів інформації з оточуючого світу.
- Об'єктом дослідження є процес автоматизації роботи з рецептами.
- Предметом дослідження є метод та технологія автоматизації роботи з рецептами.

Мета і задачі дослідження

Поставлена мета досягається розв'язанням таких основних задач:

- аналіз особливості та специфіки предметної області соціальної мережі;
- аналіз монолітної та мікросервісної архітектури;
- аналіз методів та інструментів по управлінню високонавантаженими системами та шляхи їх оптимізації;
- аналіз використання реляційних, документо-орієнтованих та графових баз даних при реалізації програмної системи та реалізувати серверну частину програмної системи;
- максимізувати зручність використання системи та налаштувати її для взаємодії із системою «Розумний будинок».

Результатом вирішення завдань кваліфікаційної роботи є повноцінна комп'ютерна система для автоматизації, публікації та поширення кулінарних рецептів, яка має інтеграцію з системою «Розумний будинок».

Наукова новизна отриманих результатів

Набули подальшого розвитку метод та технологія автоматизації публікації, поширення та пошуку рецептів за рахунок використання фізичних давачів, що допомагають зменшити витрати часу на базові людські потреби, підвищити ефективність та скоротити час приготування їжі.

Практична цінність отриманих результатів

В результаті виконання кваліфікаційної роботи були розроблені метод та технологія автоматизації роботи з рецептами, а також методи налаштування системи «Розумний будинок» для синхронізації її із соціальною мережею та наташтування веб-серверу для аналізу даних, отриманих із реального світу.

Соціальна мережа напряму взаємодіє з «Розумним будинком», що дає можливість використовувати її у помешканні як єдиний механізм, який автоматизує пошук страв та буде синхронізований з процесами, які кожна людина виконує щодня.

Актуальність теми

Актуальність теми полягає в тому, що сьогодні в Інтернеті існує велика кількість кулінарних сайтів, але повноцінних соціальних мереж з такою тематикою практично немає. Соціальні мережі є не тільки інструментом для обміну інформацією між користувачами, а й готовими платформами для ведення власного бізнесу. Крім того, приготування їжі є одним із пріоритетів більшості людей. І якщо ми об'єднаємо це, то отримаємо потужну систему, призначену для пошуку та обміну рецептами. Така система в першу чергу повинна бути швидкою та оптимізованою, а оскільки вона розрахована на велику кількість користувачів, то й високо завантажена. Існує багато технологій і практик для створення та обслуговування таких систем, але вибрати найкращу не завжди легко. Тому ми пропонуємо реалізувати таку програмну систему та на її основі проаналізувати та дослідити основні принципи оптимізації високонавантажених систем.

Аналіз предметної області

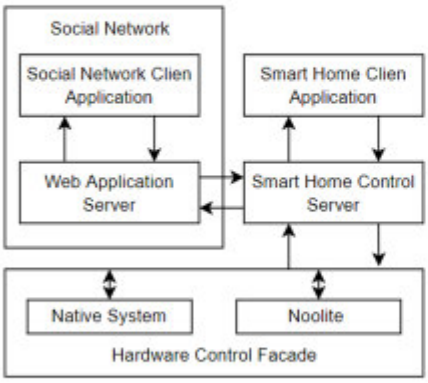
Усі веб-додатки та будь-які Інтернет-сервіси побудовані за принципом архітектури «Клієнт-сервер». Ця архітектура заснована на двох основних складових: клієнті і сервері.

У зв'язку з популярністю API серверів виникли й розвиваються різноманітні архітектурні підходи до створення інтерфейсів прикладного програмного забезпечення та управління ними. Найпопулярнішим з них є REST (Representational State Transfer).

Результати проведеної роботи

Беручи до уваги той факт, що на даний момент системи розумних будинків мають широкий спектр можливостей, є бездротовими та мають власні вмотновані механізми взаємодії зі сторонніми системами то веб-базований метод взаємодії між такою системою та веб-сервером додатку підходить найкраще.

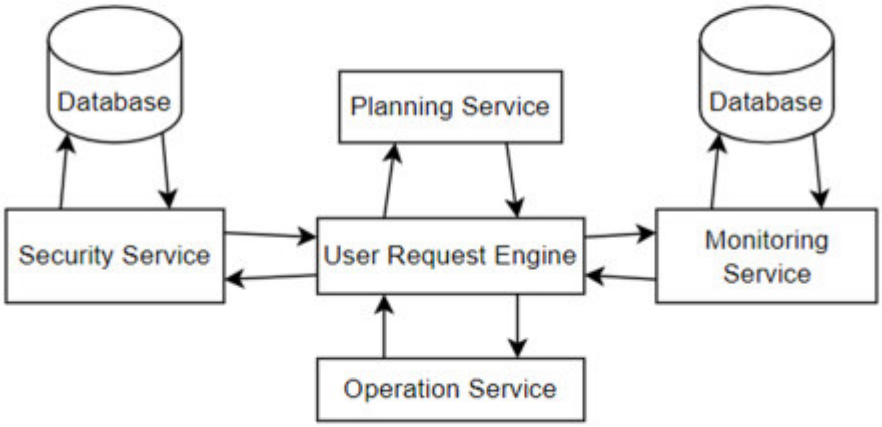
Архітектура компютерної системи автоматизації публікації



Як видно з рисунку, модуль Web Application Server соціальної мережі взаємоємодіє з модулем Smart Home Control Server системи «Розумний будинок» за допомогою REST API по протоколу HTTP, тобто це означає, що системи можуть існувати окремо одна від ондої, або бути зкомпоновані в одним механізм через серверну взаємодію.

Система має двосторонню взаємодію між сервером соціальної мережі та між сервером розумного будинку.

Структура компонентів Smart Home Control Server



Операція виконання дії

Кінцевий користувач може змінювати стан приладів через клієнтський додаток з рецептами і навпаки, система «Розумний будинок» може взаємодіяти із соціальною мережею.

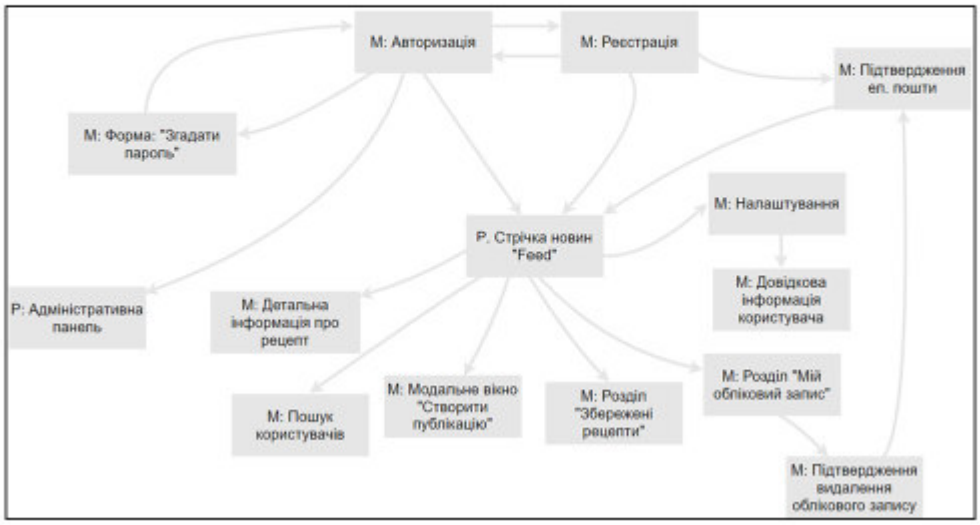
Алгоритм виконання дії для системи «Розумний дім» полягає в наступному: відбувається перевірка доступності вибраного фізичного пристрою, відбувається надсилання запиту з параметрами, що змінюють стан вибраного пристрою відповідно до вибору користувача.

Планувальник дій

Соціальна мережа включає рецепти із часом приготування. Тобто, обравши страву, відправиться запит до системи «Розумний будинок», та автоматично встановиться таймер на приладі приготування.

За ці функції відповідає плануальний дій системи «Розумний будинок».

Структура інтерфейсу програмної частини системи



Публікація

- За темою кваліфікаційної роботи опубліковано тези доповіді та взято участь у Всеукраїнській науково-практичній конференції «Актуальні проблеми комп'ютерних наук», що проходила 15-16 жовтня 2021 р. в Хмельницькому національному університеті:
- 1) Рей К., Ковтонюк І., Грищук І. Дослідження методів керування ресурсами кіберфізичної системи «Розумний будинок». Збірник наукових праць за матеріалами Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук» АПКН–2021 (Хмельницький, 15-16 жовтня 2021). С. 191-193.

Висновки

Кінцевим результатом дипломного проекту є досліджений та реалізований веб-базований метод автоматизації публікації, розповсюдження та пошуку рецептів, реалізовано веб-базований метод взаємодії із системою управління «Розумний дім». В ході дослідження було основні компоненти системи «Розумний дім»; досліджено існуючі методи побудови систем «Розумний дім». На основі вивчення та детального аналізу у сфері соціальних мереж та програмних систем було визнано ключові переваги та недоліки існуючого програмного забезпечення. Було виявлено, що сьогодні на ринку дуже мало кулінарних соціальних мереж, а доступні програмні рішення – звичайні веб-сайти без специфічних функцій соціальних мереж та не мають особливо привабливого або сучасного інтерфейсу, що підкреслює актуальність теми розробленого програмного забезпечення. На основі отриманих даних було визначено функціональні та нефункціональні потреби програми та надано ключові варіанти використання розробленої системи.

ДЯКУЮ ЗА УВАГУ

UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1011214308

Дата перевірки:
17.05.2022 10:02:43 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
17.05.2022 10:07:06 EEST

ID користувача:
100005591

Назва документа: **Рей_Метод та технологія автоматизації роботи з рецептами**

Кількість сторінок: **88** Кількість слів: **18438** Кількість символів: **141314** Розмір файлу: **1.25 MB** ID файлу: **1011106808**

26.5%
Схожість

Найбільша схожість: **25.3%** з джерелом з Бібліотеки (ID файлу: **1008215685**)

0.31% Джерела з Інтернету	38	Сторінка 90
26.4% Джерела з Бібліотеки	129	Сторінка 90

0.54% Цитат

Цитати	10	Сторінка 91

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Замінені символи	255

UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1011224863

Дата перевірки:
17.05.2022 18:56:03 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
17.05.2022 19:26:22 EEST

ID користувача:
100005591

Назва документа: **Рей_Метод та технологія автоматизації роботи з рецептами_2**

Кількість сторінок: **88** Кількість слів: **18213** Кількість символів: **139681** Розмір файлу: **1.24 MB** ID файлу: **1011116676**

13.5%
Схожість

Найбільша схожість: **12.4%** з джерелом з Бібліотеки (ID файлу: **1008215685**)

0.22% Джерела з Інтернету	31	Сторінка 90
13.4% Джерела з Бібліотеки	110	Сторінка 90

0.55% Цитат

Цитати	10	Сторінка 91

Не знайдено жодних посилань

74.3%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

Немає вилучених Інтернет-джерел

74.3% Вилученого тексту з Бібліотеки	1	Сторінка 91

Модифікації

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

Замінені символи	240

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 17.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 11%**

ID: 103552 Название: Метод та технологія автоматизації роботи з рецептами Добавлено в БД: 2022-05-17 Авторы: Рей К. С. Руководители: Говорущенко Т. О. Консультанты: Оппоненты:	Документ		Суммарное совпадение по Базе Данных	
	Символы	Лексемы	Символы	Лексемы
	130623	1170	24460 (19%)	276 (24%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы
91748	Название: Програмна система для автоматизації публікації, поширення та пошуку кулінарних рецептів Добавлено в БД: 2021-06-01 Авторы: В.О. Бойко Руководители: Л.П. Бедратюк Консультанты: Оппоненты:	22795 (17.0%)	259 (22.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Рей Костянтин Сергійович

Тема: Метод та технологія автоматизації роботи з рецептами

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг дипломної роботи:

Кількість сторінок записки 120 с.

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи підвищення ефективності та рівня автоматизації роботи з рецептами за рахунок інтеграції в реальний світ за допомогою фізичних отримувачів інформації з оточуючого світу.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі проведено дослідження предметної області та виконано постановку задачі. В другому розділі проведено проектування технології автоматизації роботи з рецептами. В третьому розділі реалізовано технологію автоматизації роботи з рецептами. Набули подальшого розвитку метод та технологія автоматизації публікації, поширення та пошуку рецептів за рахунок використання фізичних давачів, що допомагають зменшити витрати часу на базові людські потреби, підвищити ефективність та скоротити час приготування їжі. В четвертому розділі виконано тестування технології автоматизації роботи з рецептами.

4. Позитивні сторони роботи: вдале поєднання програмної та апаратної реалізації у технології автоматизації роботи з рецептами

5. Негативні сторони роботи: розроблений метод не має математичного формалізованого представлення.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на середньому науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: 3.75/С (добре).

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

Мартинюк Валерій Володимирович

зав. каф. АКІТ, д.т.н., професор

"___"_____2022 р. _____(підпис)

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Метод та технологія автоматизації роботи з рецептами

Автор: Рей Костянтин Сергійович

Спеціальність: 123 – Компютерна інженерія

Освітня програма: освітньо-наукова

Науковий керівник: Говорущенко Т.О., д.т.н., професор

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:

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

1) запозичення не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами;
4) в якості запозичень в окремих місцях системою зафіксовано фрагменти лістингів програмного коду, які є автоматично генерованим кодом і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
5) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ідентичності/схожості, складає 13.5% і адресується до 141 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Т. О. Говорущенко

Гарант ОП _____ О. С. Савенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко