МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СЕРВІСУ СПОВІЩЕНЬ ЗАСОБАМИ ANGULAR ТА NODE.JS»**

Виконала: студентка 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А. Уахалу
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Уахалу Айї
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка сервісу сповіщень засобами Angular та Node.js

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проєктування.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	23.01.2024	
2.	Збір вихідних даних.	12.02.2024	
3.	Обробка методичних та теоретичних	05.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	23.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	20.06.2024	

Студент _____ А. Уахалу
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка сервісу сповіщень засобами Angular та Node.js»: 60 с., 36 рис., 10 джерел, 2 додатки.

ANGULAR, API, MVC, NODE.JS, NOTIFICATIONS SERVICE, TYPESCRIPT, UML.

Об’єкт дослідження – процес розробки сервісу сповіщень, інструменти для роботи з повідомленнями.

Мета роботи – розробити сервіс сповіщень.

Методи дослідження – моделювання, проєктування, програмний, аналітичний.

У роботі розглянуто проєктування та реалізацію сервісу сповіщень з використанням Angular та Node.js. Застосовано UML для візуалізації системи, зокрема діаграми варіантів використання, діяльності, послідовності та розгортання.

Система надає можливість створювати, переглядати, редагувати та видаляти сповіщення. Реалізовано реєстрацію/автентифікацію користувачів, зберігання сповіщень та метаданих у MongoDB. Проведено юніт- та інтеграційне тестування.

Таким чином, розроблений сервіс сповіщень забезпечує зручний інтерфейс для керування повідомленнями, дозволяючи користувачам легко створювати та налаштовувати сповіщення. Застосування Angular, Node.js та сучасних підходів до розробки програмного забезпечення сприяло створенню ефективної та масштабованої системи.

SUMMARY

Bachelor’s qualifying paper “Developing a Notification Service Using Angular and Node.js”: 60 pages, 36 figures, 10 references, 2 supplements.

ANGULAR, API, MVC, NODE.JS, NOTIFICATIONS SERVICE, TYPESCRIPT, UML.

The object of the study is the process of developing a notification service, tools for working with notifications.

The aim of the study is to develop a notification service.

The methods of research are modeling, design, programming, analytical.

The paper presents a discussion of the design and implementation of a notification service developed using Angular and Node.js. The Unified Modeling Language (UML) was employed to create visual representations of the system, including use case diagrams, activities, sequences, and deployment.

The system enables users to create, view, edit, and delete notifications. It also facilitates user registration and authentication, as well as the storage of notifications and metadata in MongoDB. Unit and integration testing were conducted to ensure the system’s functionality and reliability.

Consequently, the developed notification service provides a user-friendly interface for managing notifications, allowing users to easily create and customize notifications. The use of Angular, Node.js, and modern approaches to software development contributed to the creation of an efficient and scalable system.

ЗМІСТ

ВСТУП

Розробка сервісів сповіщень є актуальним і важливим завданням у сучасному світі, де оперативність інформування, ефективна комунікація та залученість користувачів відіграють ключову роль. Такі сервіси дозволяють швидко повідомляти людей про важливі події, новини, зміни або надзвичайні ситуації, забезпечуючи своєчасне реагування та підвищуючи обізнаність. Вони також є зручним каналом взаємодії між організаціями, компаніями та їхніми клієнтами, співробітниками або партнерами, допомагаючи підтримувати тісний зв’язок та інформувати про нові продукти, послуги чи оновлення.

Крім того, сервіси сповіщень можуть сприяти підвищенню залученості користувачів, формуванню лояльності до бренду або підтримці інтересу до певної діяльності через регулярні повідомлення. Сучасні рішення дозволяють персоналізувати та таргетувати повідомлення для різних цільових груп, враховуючи їхні інтереси, переваги та поведінку, роблячи комунікацію більш релевантною та ефективною. Також не можна недооцінювати переваги автоматизації та ефективності, які забезпечують сервіси сповіщень, зменшуючи витрати часу та ресурсів на ручну розсилку повідомлень великій кількості людей.

Виходячи з цього, було створено сервіс сповіщень, який дозволяє швидко та зручно створити нове сповіщення та керувати ним. Для реалізації сервісу використовувалися сучасні технології та інструменти, такі як Angular та Node.js, що забезпечують високу продуктивність та надійність системи.

Актуальність дослідження: актуальність теми зумовлена необхідністю створення ефективного та зручного інструменту для сповіщень, який відповідатиме сучасним вимогам та забезпечить швидке й надійне інформування користувачів.

З огляду на це, можна виділити наступні цілі і задачі нашого дослідження:

Мета: розробити сервіс сповіщення.

Задачі:

- сформувати вимоги до сервісу;
- спроєктувати та побудувати архітектуру сервісу;
- реалізувати сервіс сповіщень;
- протестувати роботу сервісу.

Об’єкт дослідження: процес розробки сервісу сповіщень, інструменти для роботи з повідомленнями.

Предмет дослідження: фреймворк Angular та середовище Node.js.

Методи дослідження: моделювання, проєктування, програмний, аналітичний.

Перший розділ присвячено збору та аналізуванню вимог до сервісу, огляду інструментів розробки, опису предметної області та системи.

У другому розділі розглянуто проєктування додатку, наведено опис прецедентів та побудовано UML-діаграми.

Третій розділ присвячено реалізації та тестуванню роботи сервісу, наведено керівництво користувача, яке описує процес роботи із сервісом сповіщень.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Система – сервіс сповіщень, створений засобами Angular та Node.js.

Angular – це фреймворк, заснований на компонентно-орієнтованому підході.

Node.js – це однопоточне кросплатформне середовище виконання з відкритим вихідним кодом і бібліотека, яка використовується для запуску вебдодатків, написаних на JavaScript, поза браузером клієнта.

ДВВ – Діаграма Варіантів Використання чи Use Case Diagram.

ДД – Діаграма Діяльності.

ДП – Діаграма Послідовності.

ДР – Діаграма Розгортання.

Гість – користувач, який не зареєстрований або неавторизований у системі.

Авторизований користувач – користувач, який увійшов до системи та може взаємодіяти з її функціоналом.

1.1.2 Технічні терміни

NoSQL БД – документо-орієнтована база даних.

MVC – архітектурний шаблон «модель-представлення-контролер».

Framework – це набір інструментів, бібліотек та правил, який використовується для створення програмних додатків.

Компонент – структурна одиниця Angular.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати можливість створювати та керувати сповіщеннями.

Експлуатаційне призначення системи – система може експлуатуватися авторизованими користувачами та гостями системи.

Мета створення системи – розробка сервісу сповіщень.

1.2.2 Загальні функціональні можливості системи

Система має надавати гостям такі можливості:

- реєстрація в системі;
- вхід до системи.

Система має надавати авторизованим користувачам такі можливості:

- створення/перегляд/видалення/редагування категорії сповіщення;
- створення/перегляд/видалення/редагування сповіщення;
- вибір способу сповіщення/ категорії сповіщення;
- вихід з системи.

1.3 Нефункціональні вимоги

1.3.1 Інтерфейс користувача

Система повинна мати гнучкий та зрозумілий інтерфейс, який забезпечуватиме коректне відображення на різних пристроях та зручність використання для користувачів різних категорій.

1.3.2 Підтримка браузерів

Необхідно гарантувати належну роботу системи в останніх версіях популярних браузерів, таких як Mozilla Firefox, Google Chrome, Safari, Microsoft Edge та Opera, щоб забезпечити широку доступність для користувачів.

1.3.3 Вимоги до продуктивності

Завантаження сторінок має відбуватися не довше однієї секунди, а процеси створення та отримання сповіщень не повинні негативно впливати на загальну продуктивність системи.

1.3.4 Вимоги до безпеки

Доступ до функціоналу системи має бути обмежений лише для авторизованих користувачів з метою захисту даних та попередження несанкціонованого використання.

Система не повинна дозволяти користувачам вводити дані, які можуть бути використані як експлойти.

1.4 Опис предметної області

Предметною областю є розробка сервісу сповіщень. Дана система повинна надавати користувачам можливість створювати сповіщення та взаємодіяти з ними. Процес взаємодії з системою залежить від прав користувача. Для гостя доступні лише функції створення профіля та входу до системи. Після цього він, як користувач, може створювати сповіщення та налаштовувати їх.

Процес створення сповіщення відбувається наступним чином. Користувач авторизується в системі, та натискає на пункт меню «Категорії». Система відображає інтерфейс керування категоріями, де користувач може створити категорії сповіщень. Після створення категорії, користувачу необхідно обрати пункт меню «Сповіщення». Система відображає інтерфейс керування сповіщеннями, де для створення нового сповіщення користувачу необхідно натиснути на кнопку «Додати нове сповіщення», заповнити форму та натиснути на кнопку «Створити сповіщення». Система зберігає дані сповіщення до БД та відображає список сповіщень користувача.

1.5 Опис системи

Створення сервісів сповіщень є актуальним і важливим завданням у сучасному світі, де оперативність інформування, ефективна комунікація та залученість користувачів відіграють ключову роль. Такі сервіси дозволяють швидко повідомляти людей про важливі події, новини, зміни або надзвичайні ситуації, забезпечуючи своєчасне реагування та підвищуючи обізнаність. Вони також є зручним каналом взаємодії між організаціями, компаніями та їхніми клієнтами, співробітниками або партнерами, допомагаючи підтримувати тісний зв’язок та інформувати про нові продукти, послуги чи оновлення.

Крім того, сервіси сповіщень можуть сприяти підвищенню залученості користувачів, формуванню лояльності до бренду або підтримці інтересу до певної діяльності через регулярні повідомлення. Сучасні рішення дозволяють персоналізувати та таргетувати повідомлення для різних цільових груп, враховуючи їхні інтереси, переваги та поведінку, роблячи комунікацію більш релевантною та ефективною. Також не можна недооцінювати переваги автоматизації та ефективності, які забезпечують сервіси сповіщень, зменшуючи витрати часу та ресурсів на ручну розсилку повідомлень великій кількості людей.

З огляду на це було створено сервіс сповіщень, який дозволяє швидко та зручно створити нове сповіщення та керувати ним.

1.6 Опис інструментів розробки

1.6.1 Angular

Angular – це фреймворк з відкритим вихідним кодом для створення клієнтських вебдодатків. Розроблений і підтримуваний компанією Google, Angular базується на TypeScript – надбудові над JavaScript, що додає статичну типізацію та інші можливості для поліпшення продуктивності розробки і підтримки коду [1].

Архітектура побудована на основі модульного дизайну, де додаток поділяється на логічні блоки – NgModules. Це полегшує масштабування, повторне використання коду і тестування. Ключовим архітектурним шаблоном є підхід з одностороннім потоком даних – Angular Services [2, 5].

Angular пропонує широкий спектр можливостей, включаючи двостороннє зв’язування даних, маршрутизацію на основі URL, роботу з формами, анімаціями, HTTP-клієнтом, сервісами, ін’єкцією залежностей та інструментами для тестування. Бібліотека RxJS інтегрована в Angular і використовується для роботи з асинхронними потоками даних [1, 3, 5].

Angular CLI забезпечує потужне середовище для створення, розробки, тестування та розгортання додатків.

1.6.2 Node.js

Node.js – це відкрита багатоплатформна середа виконання JavaScript, що базується на движку V8 від Google і використовує підхід безперервного циклу подій (event-driven). Вона дозволяє створювати масштабовані мережеві додатки завдяки своїй асинхронній моделі неблокуючого вводу/виводу, яка забезпечує високу продуктивність та можливість ефективно обробляти кілька запитів одночасно [8].

Особливістю Node.js є те, що вона написана на C++, але використовує JavaScript як скриптову мову, надаючи розробникам можливість писати як серверну, так і клієнтську частину додатку однією мовою програмування.

Node.js має багатий набір вбудованих модулів, які розширюють функціональність JavaScript і надають APIs для роботи з файловою системою, мережевими сокетами, HTTP-запитами та іншими аспектами. Величезна екосистема відкритих бібліотек (пакетів), керованих системою пакетів npm, полегшує повторне використання коду [6, 9].

Node.js широко використовується для створення різноманітних додатків, таких як вебсервери, проксі, API-сервери, мікросервіси, прослушники черги повідомлень тощо, особливо для роботи в реальному часі, обробки великих обсягів даних, потоків даних, ігрових серверів та інших високопродуктивних та масштабованих вебдодатків.

1.6.3 MongoDB

MongoDB – це популярна система керування базами даних NoSQL з відкритим вихідним кодом, яка використовує документо-орієнтовану модель зберігання даних [7].

Розглянемо її основні характеристики.

MongoDB не використовує звичайні для реляційних баз даних таблиці і схеми, натомість вона зберігає дані у вигляді документів JSON-подібного формату BSON. Документи можуть мати різну структуру, що забезпечує високу гнучкість моделювання даних. Документи групуються у колекції, які є аналогом таблиць в реляційних базах даних [7].

База даних масштабується горизонтально шляхом розподілу даних між різними вузлами кластера, що забезпечує високу продуктивність та доступність. MongoDB підтримує автоматичну репліку та відмовостійкість [7].

Замість структурованої мови запитів SQL, MongoDB використовує прості

бінарні протоколи на основі JSON для запитів та модифікації даних. Це робить її зручною для використання з сучасними мовами програмування [7].

MongoDB має гнучку систему індексування, включаючи підтримку геопросторових індексів, що дозволяє ефективно зберігати та запитувати географічні координати [7].

Вона широко використовується у вебдодатках, мобільних сервісах, систем обробки даних великих розмірів та інших сферах завдяки своїй гнучкості, горизонтальній масштабованості та продуктивності.

Хоча MongoDB не підтримує транзакційні властивості ACID, вона пропонує інші функції, як от агрегація даних, тригери, представлення даних і багато іншого [7].

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

UML (Unified Modeling Language) є стандартизованою графічною мовою моделювання, яка широко використовується для візуалізації, специфікації, конструювання та документування об’єктно-орієнтованих програмних систем [10].

Застосування UML під час розробки системи забезпечує візуальне представлення різних аспектів системи, таких як структура, поведінка, архітектура та взаємозв’язки між компонентами, що полегшує спільне розуміння проєкту між усіма залученими сторонами, включаючи розробників, аналітиків та зацікавлені сторони.

За допомогою UML можна моделювати вимоги до системи, функціональні вимоги та бізнес-процеси, а також передавати ці вимоги від зацікавлених сторін до розробників, забезпечуючи основу для подальшого проєктування системи.

UML забезпечує структурований підхід до розробки програмного забезпечення, дозволяючи розробникам моделювати та візуалізувати різні аспекти системи на різних етапах її життєвого циклу, включаючи моделювання класів, компонентів, розгортання та поведінки системи.

Діаграми UML є стандартизованими та зрозумілими для всіх учасників проєкту, що полегшує комунікацію та обмін знаннями між ними, а також можуть використовуватися як частина документації системи, допомагаючи майбутньому обслуговуванню та розширенню системи.

Крім того, існує велика кількість інструментів, що підтримують UML, включаючи засоби для створення діаграм, зворотного проєктування коду та генерації коду з діаграм, які можуть підвищити продуктивність розробки та забезпечити узгодженість між моделями та кодом.

2.2 Діаграма варіантів використання

Діаграма варіантів використання (use case diagram) є одним з основних видів діаграм в уніфікованій мові моделювання UML. Вона використовується для візуального представлення функціональних вимог системи та описує взаємодію між системою та її акторами (користувачами або зовнішніми системами) [10].

Розглянемо основні елементи ДВВ.

Актори – зовнішні сутності, які взаємодіють із системою, такі як користувачі, інші системи або пристрої. Вони зображуються у вигляді фігурок людини або іншого символу [10].

Варіанти використання – послідовності дій або сценаріїв, які можуть бути ініційовані актором і представляють певну функціональність системи. Вони зображуються у вигляді овалів з назвою всередині [10].

Асоціації – лінії, що з’єднують акторів і варіанти використання, які вказують на можливість взаємодії між ними [10].

Включення (include) – відношення між варіантами використання, коли функціональність одного варіанту використання включається в інший [10].

Розширення (extend) – відношення між варіантами використання, коли один варіант використання може розширювати або доповнювати функціональність іншого за певних умов [10].

Діаграма варіантів використання забезпечує високорівневе уявлення про функціональність системи та її межі. Вона допомагає зрозуміти вимоги до системи, визначити ролі та обов’язки акторів, а також виявити можливі області функціональності, які можуть бути пропущені або недостатньо описані.

Ця діаграма є корисним інструментом для комунікації між зацікавленими сторонами, аналітиками, дизайнерами та розробниками системи, допомагаючи їм спільно зрозуміти і узгодити функціональні вимоги до системи.

На рисунку 2.1 представлена діаграма варіантів використання сервісу сповіщень.

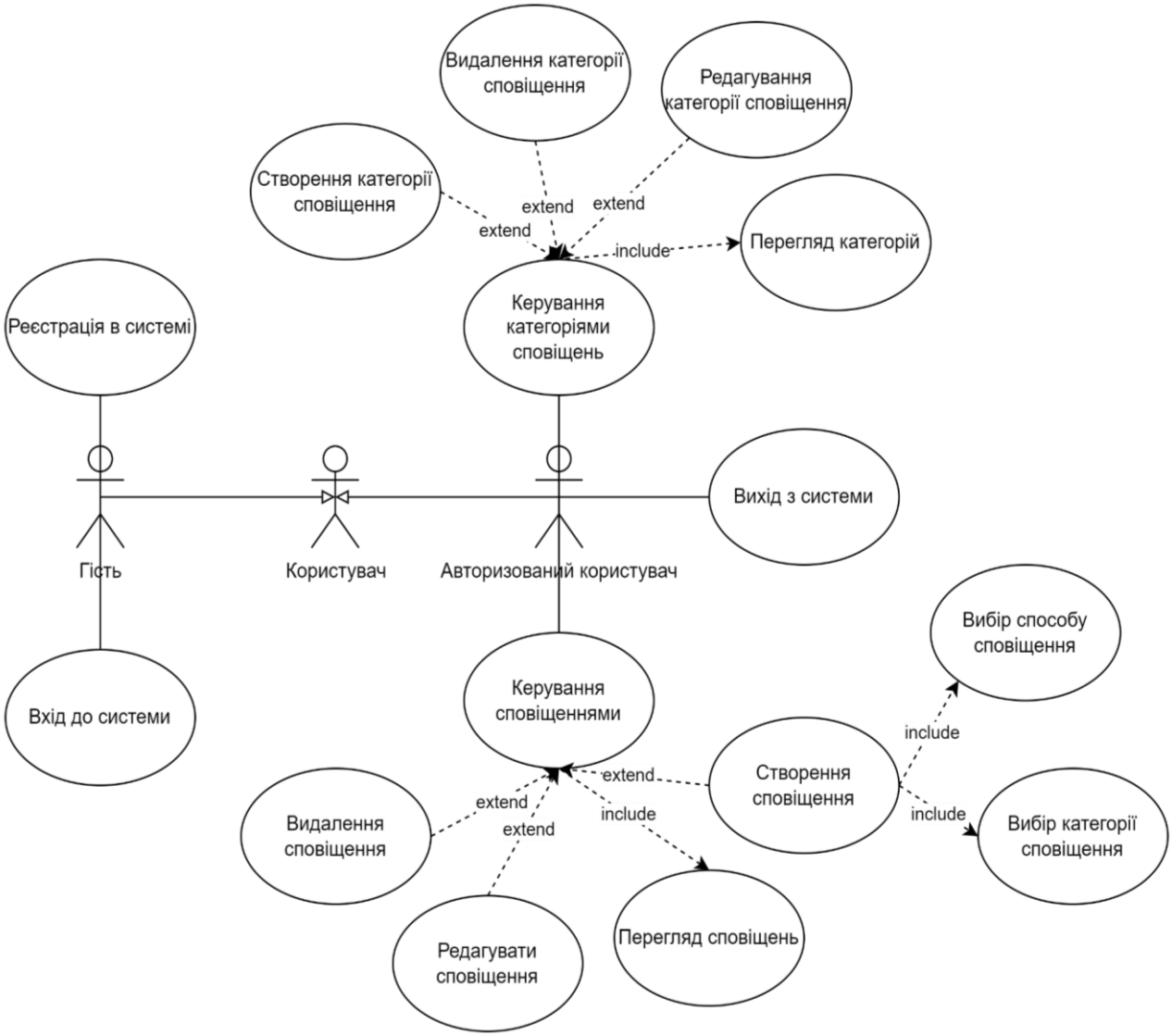

Рисунок 2.1 – Діаграма варіантів використання

На діаграмі представлено акторів «Гість» та «Авторизований користувач», кожен з яких може взаємодіяти з системою на різних рівнях. Також представлено актора «Користувач», який включає в себе прецеденти інших акторів.

Виділено 1 основний варіант використання – «Створення сповіщення». Після того, як користувач переходить до пункту меню «Сповіщення», система відображає інтерфейс керування сповіщеннями. Користувач натискає на кнопку «Додати нове сповіщення», заповнює форму відповідними даними та натискає на кнопку «Створити сповіщення». Система зберігає дані до БД та відображає список сповіщень користувача.

Варіанти використання є ключовим інструментом для визначення та опису

функціональних можливостей програмної системи з точки зору її кінцевих користувачів або зовнішніх суб’єктів (акторів). Кожен варіант використання представляє певний сценарій або послідовність дій, які актор може виконати в межах взаємодії з системою для досягнення конкретної мети чи отримання певного результату. Таким чином, варіанти використання описують різні способи застосування функціоналу системи її користувачами для виконання своїх завдань та задоволення потреб.

Головна мета варіантів використання – змоделювати та задокументувати вимоги до поведінки та функціональних можливостей системи з погляду акторів, які нею оперують. Вони визначають конкретні сценарії використання різних функцій системи, кроки чи дії, необхідні для досягнення певного результату. Послідовність дій у варіанті використання описує, як актор взаємодіє з системою для виконання конкретного завдання або реалізації потреби.

Таким чином, варіанти використання забезпечують зрозумілий спосіб представлення функціональних вимог до програмного забезпечення, дозволяючи розробникам, аналітикам та замовникам узгодити спільне бачення щодо функціоналу системи та способів його використання кінцевими користувачами або зовнішніми учасниками (див. рис. 2.2 – 2.3).

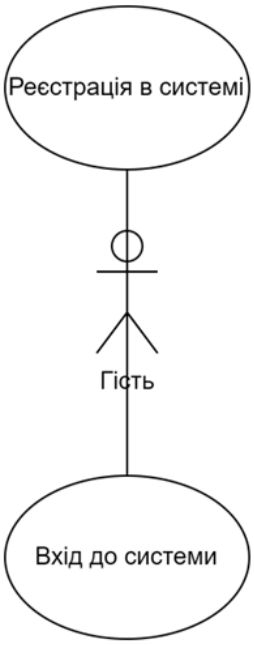

Рисунок 2.2 – ДВВ «Гість»

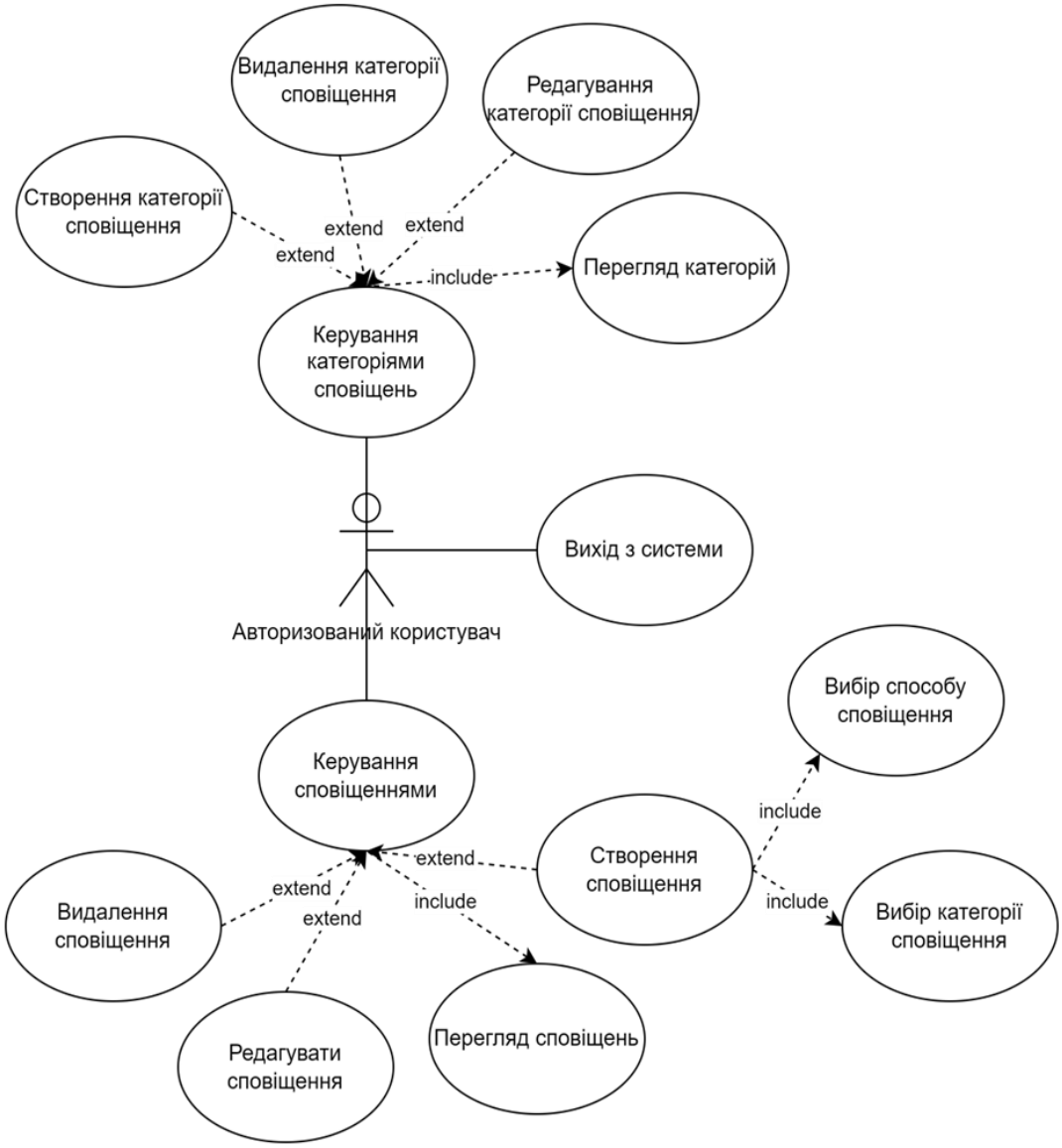

Рисунок 2.3 – ДВВ «Авторизований користувач»

2.2.1 Опис варіантів використання

Прецедент «Реєстрація в системі».

Призначення: даний варіант використання надає можливість користувачу зареєструватися в системі.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач переходить на головну сторінку сервісу та обирає пункт «Зареєструватися». Система відображає форму реєстрації. Після введення необхідних даних, користувач натискає на кнопку «Реєстрація», система створює

новий обліковий запис, зберігає його параметри та переадресовує на сторінку входу до системи.

Виняткова ситуація 1: невалідні дані – система відображає повідомлення про помилку, користувач може повторно ввести дані.

Виняткова ситуація 2: такий користувач вже зареєстрований – система відображає повідомлення про помилку, користувач може змінити дані.

Виняткова ситуація 3: помилка сервера – система відображає відповідне повідомлення.

Прецедент «Вхід до системи».

Призначення: даний варіант використання надає можливість користувач входити до системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач переходить на головну сторінку сервісу. Система відображає форму входу до системи. Після введення електронної пошти та пароля, користувач натискає на кнопку «Вхід», система відображає особистий кабінет користувача.

Передумова: перед початком виконання даного варіанта використання користувач повинен зареєструватися в системі.

Виняткова ситуація 1: невалідні дані – система відображає повідомлення про помилку, користувач може повторно ввести дані.

Виняткова ситуація 2: такого користувача не існує – система відображає повідомлення про помилку, користувач може змінити дані або зареєструватися.

Виняткова ситуація 3: помилка сервера – система відображає відповідне повідомлення.

Прецедент «Вихід з системи».

Призначення: даний варіант використання надає можливість користувачу виходити з системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на пункт меню «Вихід». Система видаляє дані про сесію користувача та відображає сторінку входу.

Прецедент «Керування категоріями сповіщень».

Призначення: даний варіант використання надає можливість користувачу керувати категоріями сповіщень.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на пункт меню «Категорії». Система відображає інтерфейс керування категоріями сповіщень.

Передумова: перед початком виконання даного варіанта використання користувач повинен увійти до системи.

Виняткова ситуація 1: сесія користувача вичерпана – система відображає головну сторінку.

Прецедент «Створення категорії сповіщень».

Призначення: даний варіант використання надає можливість користувачу створювати нові категорії сповіщень.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на кнопку «Додати нову категорію». Система відображає форму створення нової категорії. Користувач може ввести дані в відповідні поля. Після введення даних, користувач натискає на кнопку «Створити категорію». Система зберігає дані категорії в БД та відображає сторінку керування категоріями.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися на сторінці керування категоріями.

Виняткова ситуація 1: невалідні дані – система відображає повідомлення про помилку, користувач може повторно ввести дані.

Виняткова ситуація 2: сесія користувача вичерпана – система відображає головну сторінку.

Виняткова ситуація 3: категорія вже існує – система відображає відповідне повідомлення, користувач може повторно ввести дані.

Виняткова ситуація 4: помилка сервера – система відображає відповідне повідомлення, дані не зберігаються.

Прецедент «Редагування категорії сповіщень».

Призначення: даний варіант використання надає можливість користувачу редагувати створені категорії сповіщень.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на кнопку «Редагувати» у рядку відповідної категорії. Система відображає форму редагування категорії. Користувач може змінити дані. Після редагування даних, користувач натискає на кнопку «Оновити категорію». Система оновлює дані категорії в БД та відображає сторінку керування категоріями.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися на сторінці керування категоріями, а також повинна бути створена принаймні одна категорія.

Виняткова ситуація 1: невалідні дані – система відображає повідомлення про помилку, користувач може повторно ввести дані.

Виняткова ситуація 2: сесія користувача вичерпана – система відображає головну сторінку.

Виняткова ситуація 3: категорія вже існує – система відображає відповідне повідомлення, користувач може повторно ввести дані.

Виняткова ситуація 4: помилка сервера – система відображає відповідне повідомлення, дані не оновлюються.

Прецедент «Видалення категорії сповіщень».

Призначення: даний варіант використання надає можливість користувачу видалити категорію сповіщень.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на кнопку «Видалити» у рядку відповідної категорії. Система видаляє дані категорії з БД та відображає сторінку керування категоріями.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися на сторінці керування категоріями, а також повинна бути створена принаймні одна категорія.

Виняткова ситуація 1: сесія користувача вичерпана – система відображає

головну сторінку.

Виняткова ситуація 2: помилка сервера – система відображає відповідне повідомлення, дані не видаляються.

Прецедент «Перегляд категорій».

Призначення: даний варіант використання надає можливість користувачу переглядати список створених ним категорій.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на пункт меню «Категорії». Система відображає список категорій, а також інструментарій взаємодії з ними.

Передумова: перед початком виконання даного варіанта використання користувач повинен авторизуватися в системі.

Виняткова ситуація 1: сесія користувача вичерпана – система відображає головну сторінку.

Виняткова ситуація 2: помилка сервера – система відображає відповідне повідомлення.

Прецедент «Керування сповіщеннями».

Призначення: даний варіант використання надає можливість користувачу керувати сповіщеннями.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на пункт меню «Сповіщення». Система відображає інтерфейс керування сповіщеннями.

Передумова: перед початком виконання даного варіанта використання користувач повинен увійти до системи.

Виняткова ситуація 1: сесія користувача вичерпана – система відображає головну сторінку.

Прецедент «Створення сповіщення».

Призначення: даний варіант використання надає можливість користувачу створювати нові сповіщення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на кнопку «Додати нове сповіщення». Система

відображає форму створення нового сповіщення. Користувач може ввести дані в відповідні поля, які включають в себе назву, опис, вибір категорії та типу сповіщення, дата та час. Після введення необхідних даних, користувач натискає на кнопку «Створити сповіщення». Система зберігає дані в БД та відображає сторінку керування сповіщеннями.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися на сторінці керування сповіщеннями та повинна бути створена принаймні одна категорія.

Виняткова ситуація 1: невалідні дані – система відображає повідомлення про помилку, користувач може повторно ввести дані.

Виняткова ситуація 2: сесія користувача вичерпана – система відображає головну сторінку.

Виняткова ситуація 3: помилка сервера – система відображає відповідне повідомлення, дані не зберігаються.

Прецедент «Редагування сповіщення».

Призначення: даний варіант використання надає можливість користувачу редагувати створені сповіщення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на кнопку «Редагувати» у рядку відповідного сповіщення. Система відображає форму редагування сповіщення. Користувач може змінити дані. Після редагування необхідних даних, користувач натискає на кнопку «Оновити сповіщення». Система оновлює дані в БД та відображає сторінку керування сповіщеннями.

Передумова: перед початком виконання даного варіанта використання повинно бути створено принаймні одне сповіщення.

Виняткова ситуація 1: невалідні дані – система відображає повідомлення про помилку, користувач може повторно ввести дані.

Виняткова ситуація 2: сесія користувача вичерпана – система відображає головну сторінку.

Виняткова ситуація 3: помилка сервера – система відображає відповідне

повідомлення, дані не оновлюються.

Прецедент «Видалення сповіщення».

Призначення: даний варіант використання надає можливість користувачу видалити сповіщення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на кнопку «Видалити» у рядку відповідного сповіщення. Система видаляє дані категорії з БД та відображає сторінку керування сповіщеннями.

Передумова: перед початком виконання даного варіанта використання повинно бути створено принаймні одне сповіщення.

Виняткова ситуація 1: сесія користувача вичерпана – система відображає головну сторінку.

Виняткова ситуація 2: помилка сервера – система відображає відповідне повідомлення, дані не видаляються.

Прецедент «Перегляд сповіщень».

Призначення: даний варіант використання надає можливість користувачу переглядати список створених ним сповіщень.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на пункт меню «Сповіщення». Система відображає список сповіщень, а також інструментарій взаємодії з ними.

Передумова: перед початком виконання даного варіанта використання користувач повинен авторизуватися в системі.

Виняткова ситуація 1: сесія користувача вичерпана – система відображає головну сторінку.

Виняткова ситуація 2: помилка сервера – система відображає відповідне повідомлення.

Прецедент «Вибір категорії сповіщення».

Призначення: даний варіант використання надає можливість користувачу вибрати категорію під час створення або редагування сповіщення.

Основний потік подій: даний варіант використання починає виконуватися,

коли користувач або створює, або редагує сповіщення. Система надає можливість обрати одну із створених користувач категорій.

Прецедент «Вибір способу сповіщення».

Призначення: даний варіант використання надає можливість користувачу вибрати спосіб оповіщення під час створення або редагування сповіщення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач або створює, або редагує сповіщення. Система надає можливість обрати один із наявних способів сповіщення.

2.3 Діаграма діяльності

Діаграма діяльності є одним з видів діаграм в уніфікованій мові моделювання UML, який застосовується для візуального представлення потоку керування та послідовності дій в системі. Вона дозволяє візуалізувати кроки або завдання, які мають бути виконані під час певного процесу або реалізації певного варіанта використання. На діаграмі діяльності окремі дії зображуються у вигляді прямокутників із заокругленими кутами, а напрямок їх виконання позначається стрілками, які утворюють потік керування [10].

Діаграма також підтримує моделювання альтернативних шляхів виконання, паралельних потоків, розгалужень і злиття через використання спеціальних вузлів, таких як рішення, вилки та з’єднання. Об’єктні потоки дозволяють представляти передачу об’єктів, даних або повідомлень між діями в процесі їх виконання. Важливими елементами є початковий та кінцевий вузли, які відповідно вказують на точку входу та виходу з послідовності дій.

Діаграма діяльності є гнучким і потужним інструментом для моделювання та документування різноманітних процесів, включаючи бізнес-процеси, робочі потоки, реалізацію варіантів використання, а також логіку окремих методів. Вона допомагає візуально аналізувати складні процеси, виявляти потенційні вузькі місця або неефективності, а також забезпечує ефективну комунікацію між

різними зацікавленими сторонами, такими як бізнес-аналітики, розробники та користувачі системи.

На рисунках 2.4 – 2.5 наведено діаграму діяльності прецеденту «Створити сповіщення».

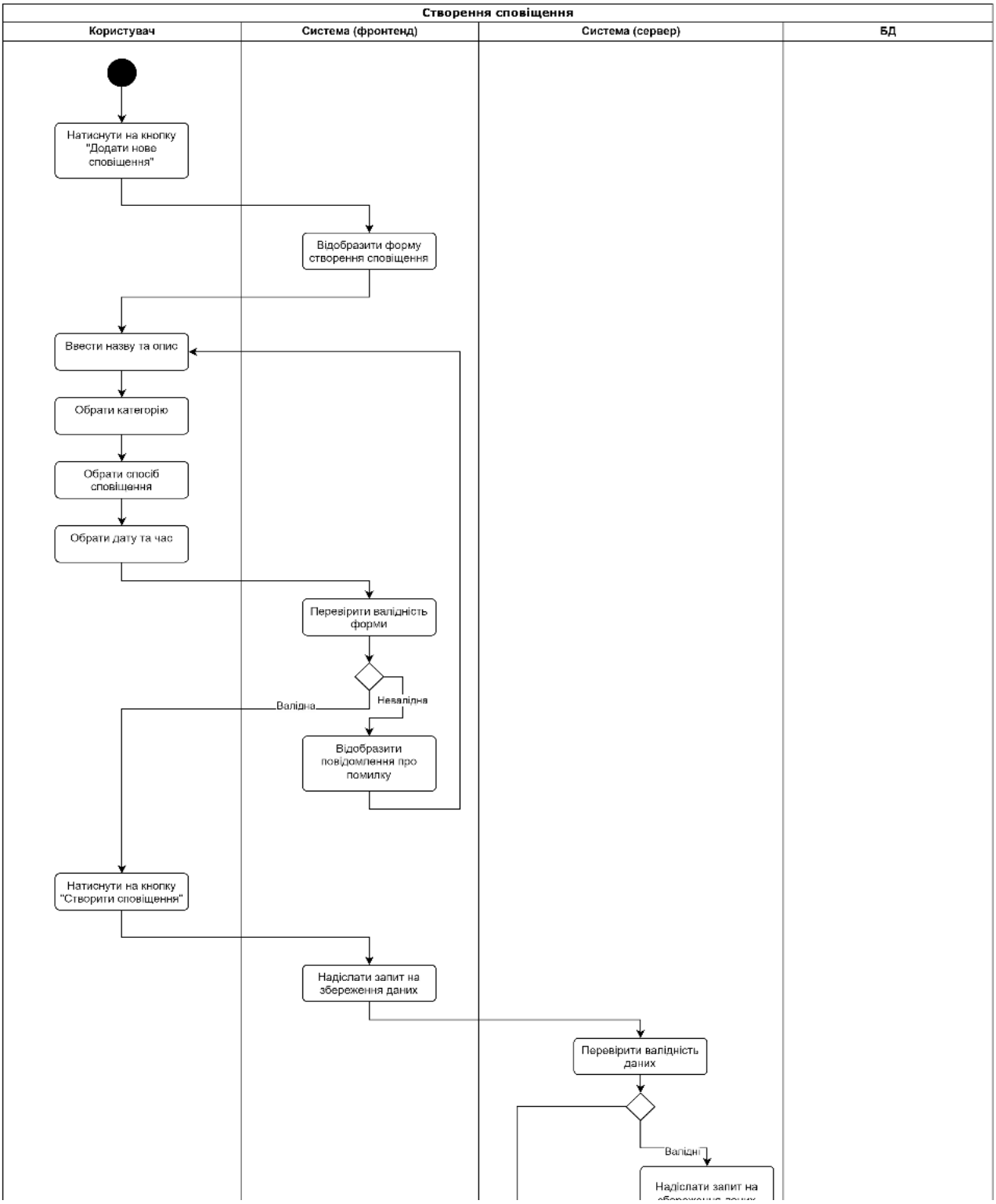

Рисунок 2.4 – Діаграма діяльності (1 частина)

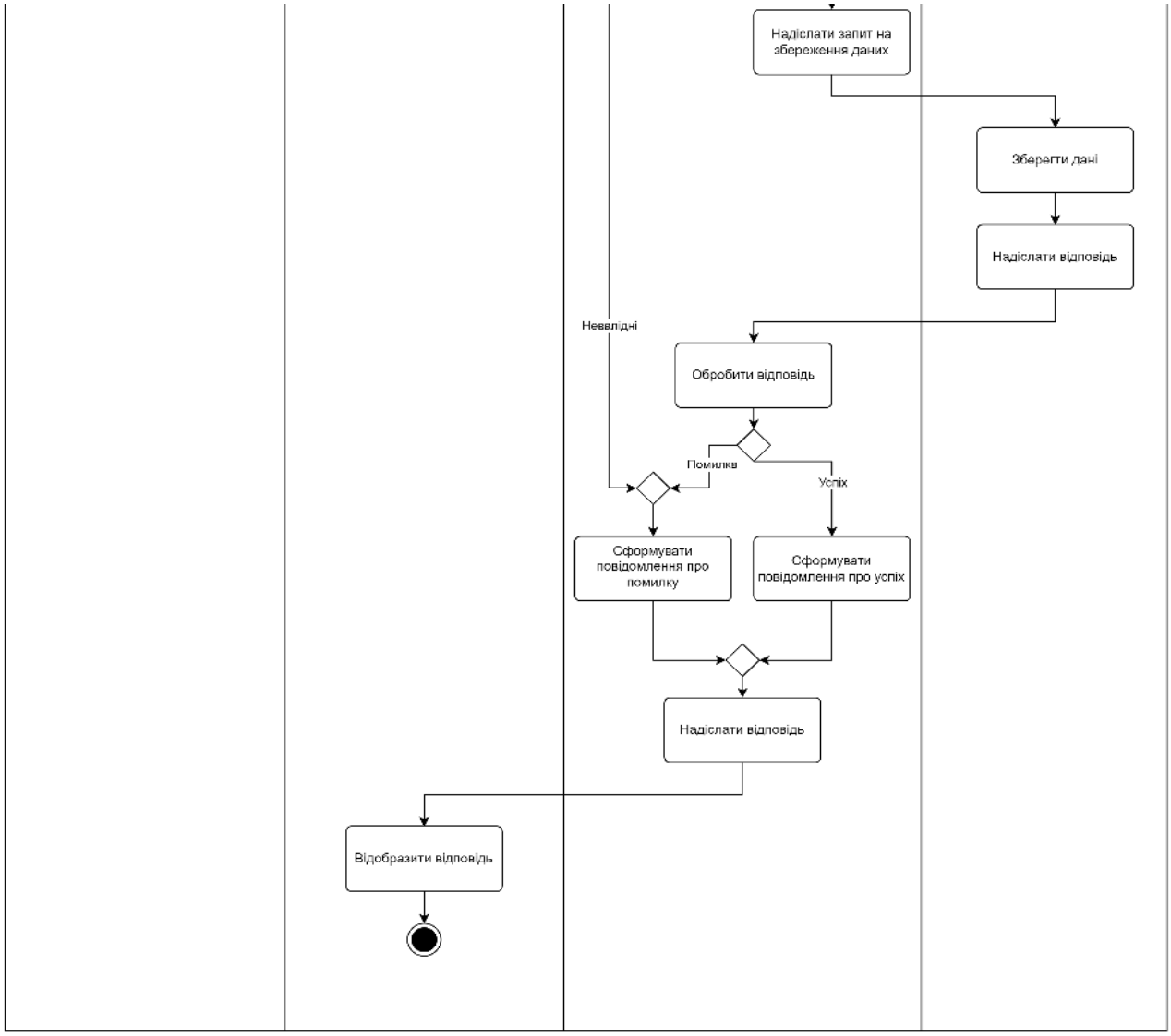

Рисунок 2.5 – Діаграма діяльності (2 частина)

2.4 Діаграма послідовності

Діаграма послідовності є одним з видів діаграм взаємодії в уніфікованій мові моделювання UML. Вона використовується для візуального представлення взаємодії між об’єктами в системі в часовій послідовності. Діаграма послідовності допомагає зрозуміти порядок обміну повідомленнями між об’єктами та внутрішню поведінку системи під час виконання певного сценарію або варіанту використання [10].

На рисунку 2.6 описана діаграма послідовності прецеденту «Створити сповіщення».

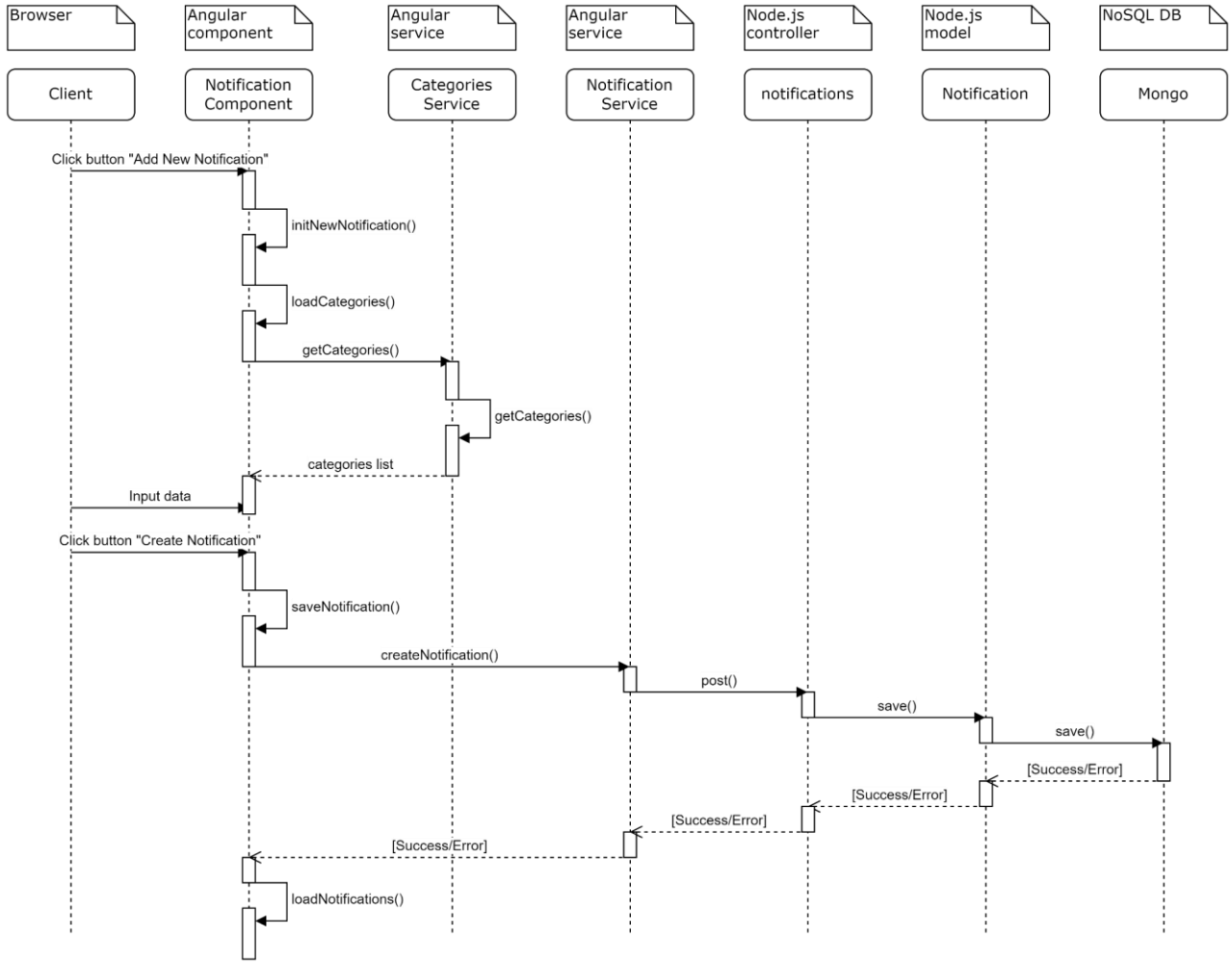

Рисунок 2.6 – ДП «Створити сповіщення»

На вертикальній осі діаграми послідовності розташовані об’єкти або учасники взаємодії, які називаються лініями життя. Лінії життя представляють існування об’єктів протягом певного періоду часу. Горизонтальна вісь відображає часову послідовність, в якій повідомлення передаються між об’єктами. Самі повідомлення зображуються як горизонтальні стрілки між лініями життя об’єктів, що вказують на відправника та отримувача повідомлення, а також його напрямок [10].

Діаграма послідовності дозволяє візуалізувати не лише обмін повідомленнями між об’єктами, але й внутрішню поведінку окремих об’єктів, таку як виклик методів або внутрішні операції. Ця інформація може бути представлена на діаграмі за допомогою фрагментів взаємодії, які охоплюють певну частину послідовності повідомлень.

Ця діаграма є особливо корисною для аналізу та документування складних сценаріїв взаємодії між компонентами системи, виявлення потенційних проблем або вузьких місць у комунікації між об’єктами, а також для полегшення розуміння поведінки системи різними зацікавленими сторонами, такими як розробники, тестувальники та архітектори.

Діаграма послідовності також може включати додаткові елементи та конструкції для більш детального представлення різних аспектів взаємодії. Наприклад, можна використовувати умовні або циклічні фрагменти для моделювання розгалужень або циклів у послідовності повідомлень. Крім того, можна застосовувати конструкції паралельних фрагментів для демонстрації одночасного виконання кількох потоків взаємодії. Також діаграма послідовності дозволяє показувати створення та знищення об’єктів під час взаємодії, а також використовувати відповідні символи для позначення асинхронних повідомлень або викликів операцій із зворотним викликом [10].

2.5 Діаграма розгортання

Діаграма розгортання в UML (Unified Modeling Language) використовується для візуалізації архітектури системи з точки зору апаратного забезпечення, на якому вона працює. Цей тип діаграми ілюструє, як різні частини програмного забезпечення розподілені між фізичними чи віртуальними вузлами, такими як сервери, комп’ютери або віртуальні машини. Вузли з’єднані мережевими лініями, що показують спосіб комунікації між ними [10].

Крім того, на діаграмі представлені артефакти, такі як виконувані файли, бібліотеки або бази даних, які розміщені на цих вузлах. За допомогою діаграми розгортання можна визначити вимоги до апаратного забезпечення, оптимізувати розподіл ресурсів і глибше зрозуміти взаємозв’язки та залежності в комплексних системах, а також деталізувати аспекти зв’язку між компонентами, включаючи протоколи та версії програмного забезпечення [10].

На рисунку 2.7 наведено діаграму розгортання системи.

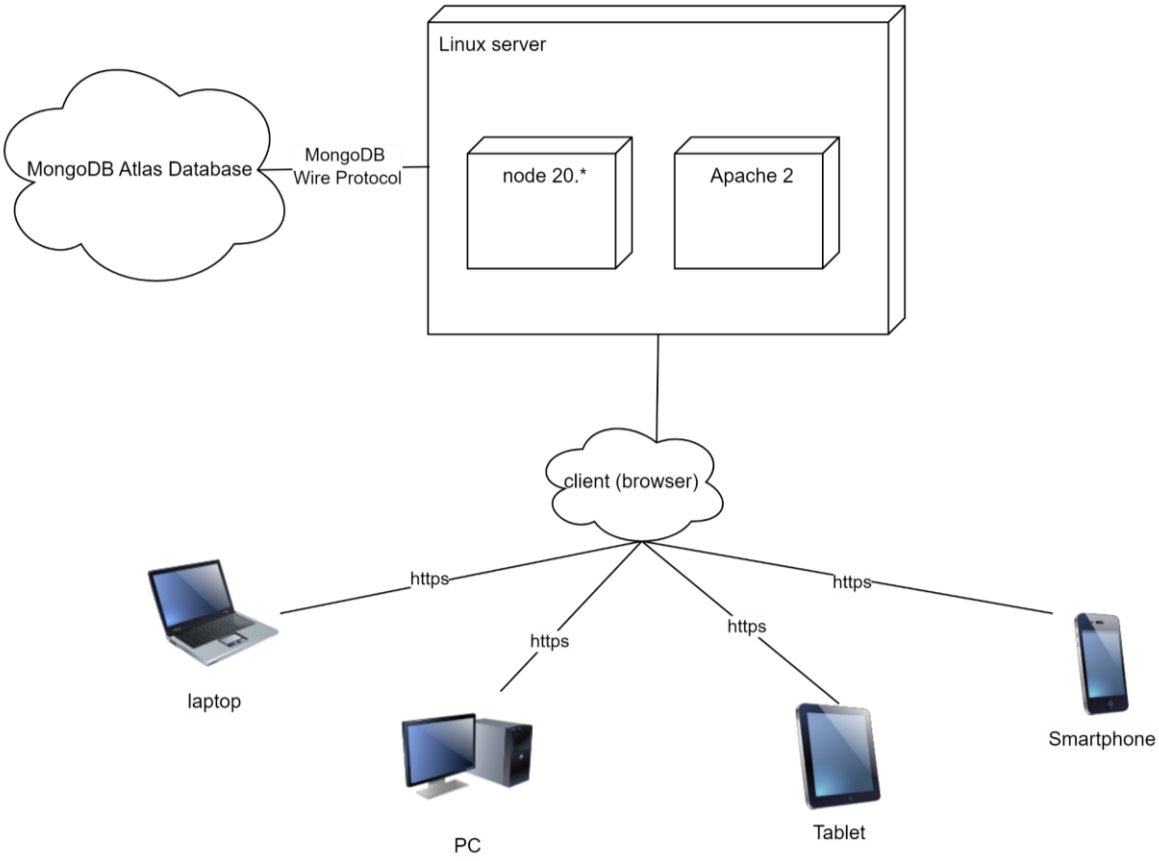

Рисунок 2.7 – Діаграма розгортання

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Для реалізації було використано фреймворк Angular та платформу Node.js.

RxJS – це бібліотека, що реалізує принципи реактивного програмування JavaScript. Заснована на об’єктах типу Observable, вона спрощує написання та контроль асинхронного та подійного коду [4].

MongoDB Atlas Database – це хмарна служба для розгортання, керування та масштабування розподілених сховищ даних MongoDB [7].

3.2 Основні класи системи

Оскільки основним прецедентом системи є «Створення сповіщення», то основними класами системи будуть ті, що надають можливість вводити дані про сповіщення та зберегти їх до БД.

Головними класами системи зі сторони фронтенду є:

- NotificationsComponent – відповідає за взаємодію зі сповіщеннями (створення, видалення, редагування, перегляд);
- NotificationService – відповідає за надсилання HTTP запитів на сервер, що стосуються колекції notifications;
- CategoryService – відповідає за надсилання HTTP запитів на сервер, що стосуються колекції categories.

Головними класами системи зі сторони серверу є:

- routes/notifications – відповідає за обробку роутів та надсилання відповідей на фронтенд;
- models/Notification – відповідає за взаємодію з БД.

Нижче приведено код моделі Notification (рис. 3.1).

```
const mongoose = require('mongoose');

const notificationSchema = new mongoose.Schema({
  title: { type: String, required: true },
  description: { type: String },
  category: { type: mongoose.Schema.Types.ObjectId, ref: 'Category' },
  notifyDate: { type: Date, required: true },
  notifyMethods: { type: [String], enum: ['email', 'dashboard'], required: true },
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: false },
  status: { type: String, enum: ['pending', 'sent'], default: 'pending' }
});

module.exports = mongoose.model('Notification', notificationSchema);
```

Рисунок 3.1 – Модель Notification

Детально ознайомитися з кодом серверної частини можна в додатку А.

3.3 Налаштування MongoDB Atlas Database

Для зберігання даних про користувачів, категорії та сповіщення використовується MongoDB Atlas Database. Це дозволяє пришвидшити роботу системи, а також забезпечує безпеку даних.

Для створення БД на MongoDB Atlas Database необхідно або увійти до особистого кабінету на cloud.mongodb.com, або створити обліковий запис, якщо він відсутній. Після входу, необхідно натиснути на кнопку «Create cluster» (рис. 3.2) [7].

Рисунок 3.2 – Кнопка «Create cluster»

Далі необхідно вибрати опції кластера, від цього буде залежати його ціна та швидкодія. Сервіс пропонує використовувати заготовлені шаблони опцій, щоб обрати оптимальні для користувача налаштування. На цьому етапі необхідно задати назву, провайдера та регіон кластера (рис. 3.3) [7].

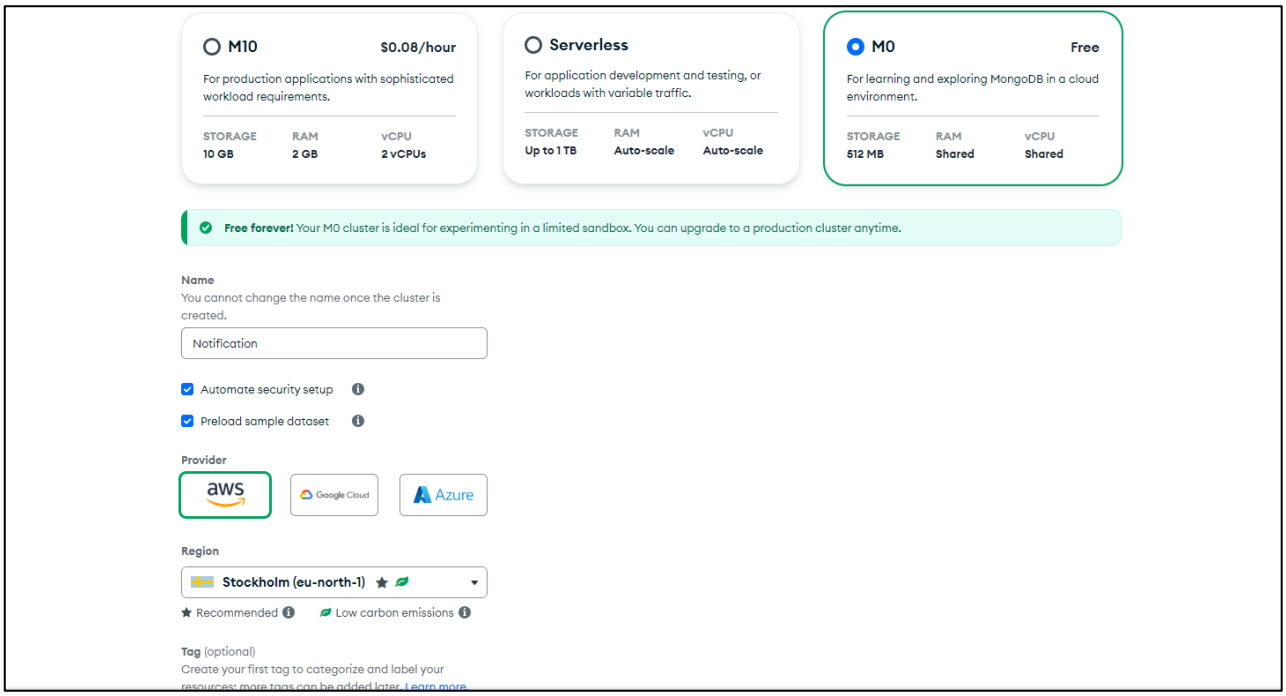

Рисунок 3.3 – Налаштування кластера

Після вибору налаштувань кластеру, необхідно створити користувача бази даних. Для цього потрібно ввести ім’я та пароль, а потім натиснути на кнопку «Create database user» (рис. 3.4).

Для переходу на другий крок, необхідно натиснути на кнопку «Choose a connection method». Сервіс пропонує обрати метод з’єднання, в нас це MongoDB Compass. В інтерфейсі обраного методу обираємо налаштування під себе та отримуємо дані для з’єднання з нашою БД. Їх ми можемо використовувати як в Compass, так і в Node.js (рис. 3.5) [7].

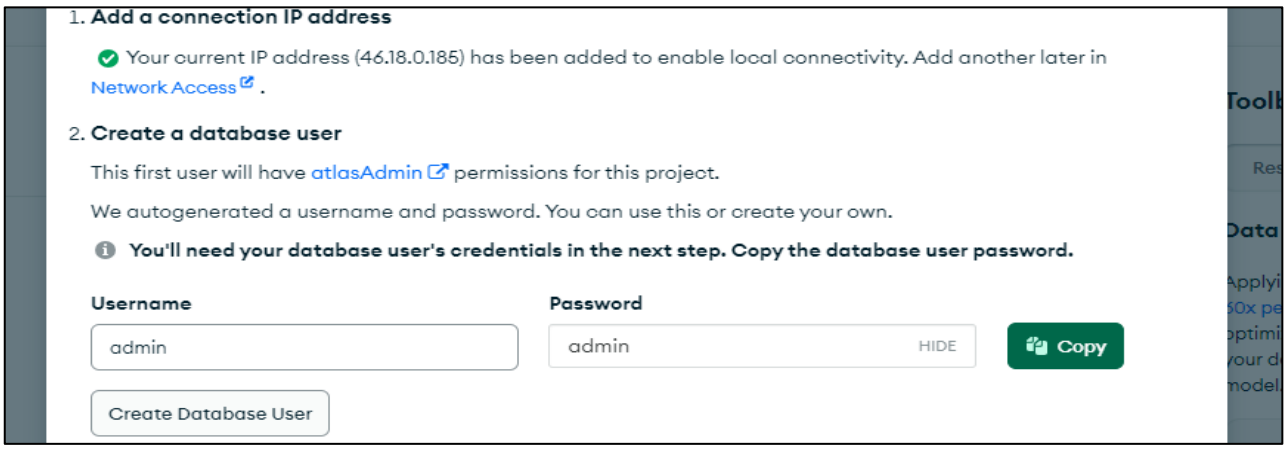

Рисунок 3.4 – Створення користувача БД

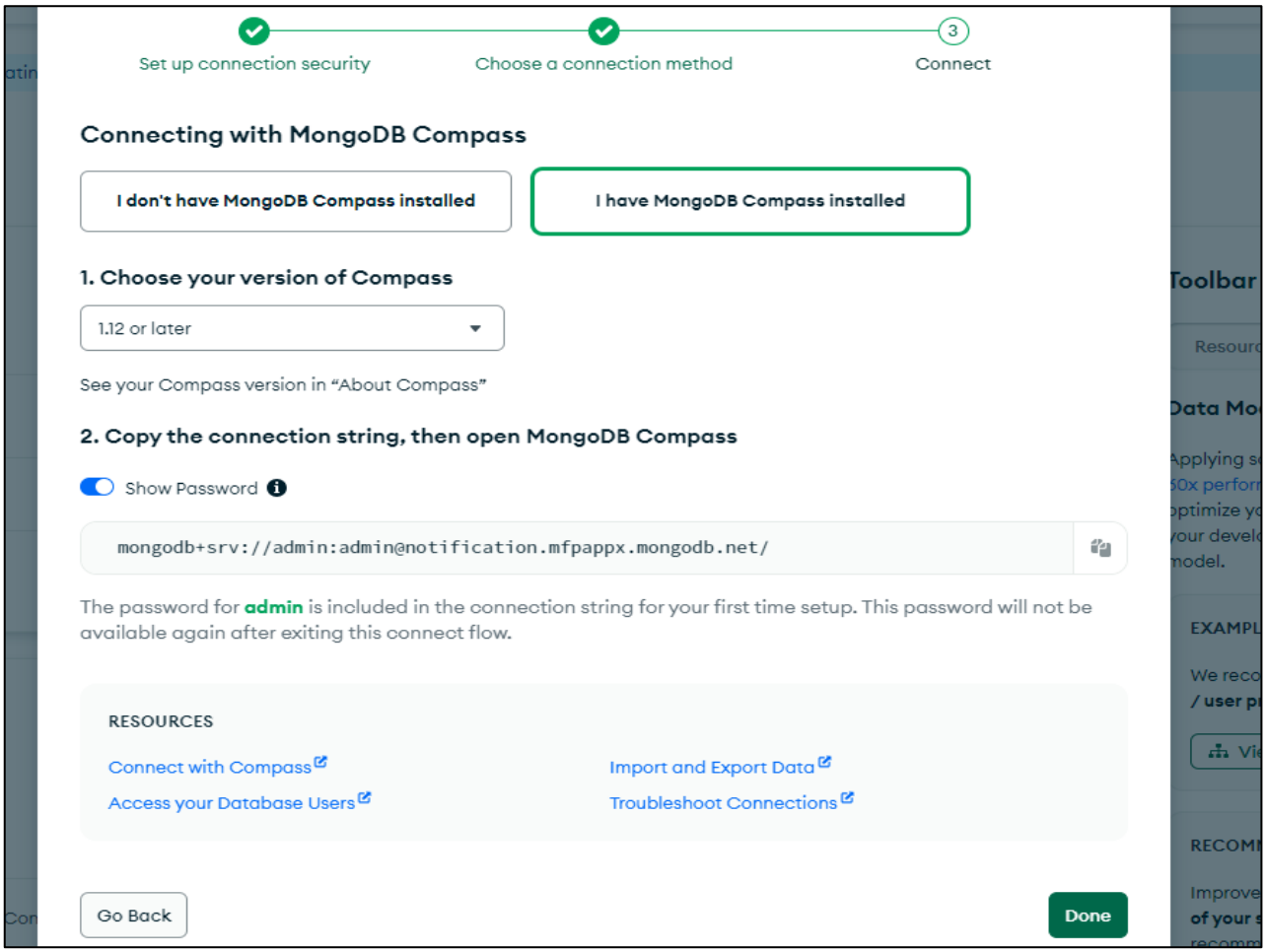

Рисунок 3.5 – Отримання даних для з'єднання з БД

Після створення, користувач може побачити кластер у відповідному розділі dashboard (рис. 3.6) [7].

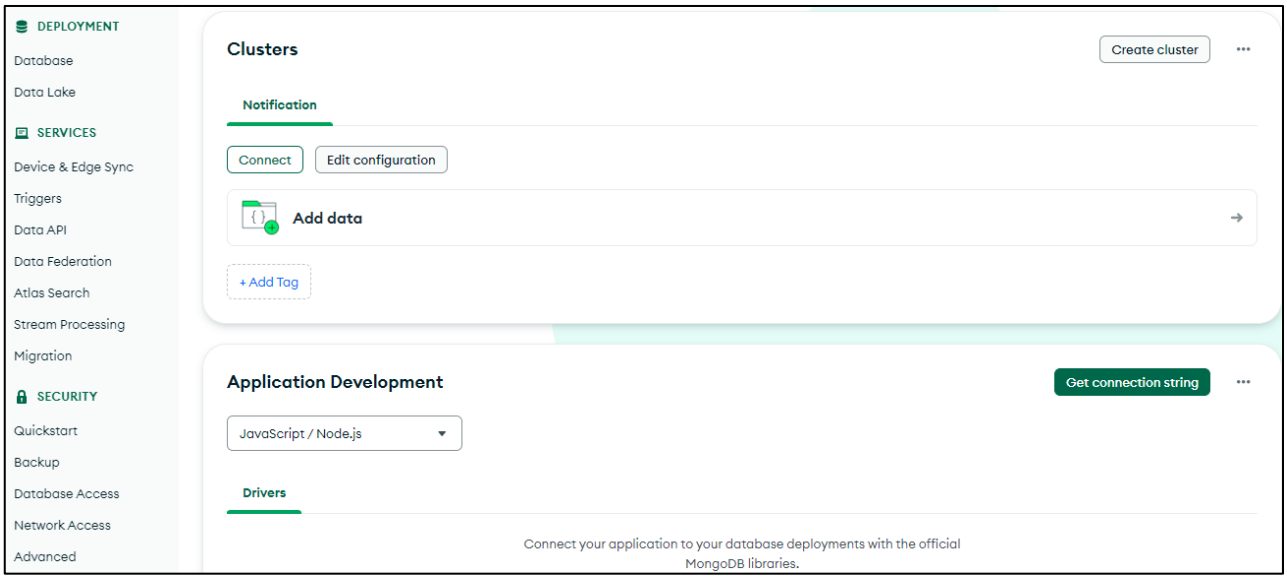

Рисунок 3.6 – Перегляд кластеру

3.4 Тестування проєкту

Unit тестування в Angular є важливою практикою для забезпечення якості та коректної роботи додатку. Angular надає вбудовані інструменти для написання юніт-тестів, такі як Jasmine та Karma [1].

Angular використовує Jasmine як бібліотеку для написання тестів і Karma як середовище виконання для запуску тестів. За угодою, Angular створює файл з тестами для кожного компонента, сервісу або іншого класу в тій самій директорії, що й сам компонент, з розширенням .spec.ts. Конфігурація для налаштування тестового середовища міститься у файлі karma.conf.js та test.ts [1].

Angular надає декілька утиліт для полегшення тестування, такі як TestBed для тестування компонентів, HttpClientTestingModule для тестування HTTP-запитів і RouterTestingModule для тестування маршрутизації. Для тестування компонентів використовуються методи TestBed.configureTestingModule та TestBed.createComponent. Вони дозволяють створити та налаштувати тестове середовище для компонента. Для тестування сервісів можна просто імпортувати та створити екземпляр сервісу і викликати його методи [2, 3, 5].

Jasmine надає функціональність для створення моків та шпигунів, які дозволяють імітувати залежності та перевіряти взаємодію з ними. Тести можна запускати з командного рядка за допомогою команди ng test або через інтерфейс інструментів розробки, таких як VSCode або WebStorm.

Написання юніт-тестів для Angular додатків є важливою практикою для забезпечення стабільності, підтримуваності та якості коду. Це допомагає виявити та запобігти помилкам на ранніх етапах розробки і полегшує внесення змін та рефакторинг в майбутньому.

На рисунку 3.7 наведено приклад такого тестування для компонента Notification.

Для перевірки роботи тестів в Angular, необхідно запустити консольну команду «ng test» [1]. Результати тестування відкриються у вікні браузера. Орієнтуючись на них, можна зрозуміти, які проріхи є в системі (рис. 3.8). Детально ознайомитися з повним кодом теста можна в додатку Б.

```
  it('should save new notification', () => {
    const mockNotification = { title: 'New Notification', description: 'New Notification', notifyDate: '2023-05-
16T12:00:00Z', category: '2', notifyMethods: ['dashboard'] };
    component.currentNotification = mockNotification;

    notificationServiceMock.createNotification.and.returnValue(of(null));

    component.saveNotification();

    expect(notificationServiceMock.createNotification).toHaveBeenCalledWith(mockNotification);
    expect(notificationServiceMock.getNotificationsHistory).toHaveBeenCalled();
    expect(component.currentNotification).toEqual({ title: '', description: '', notifyDate: '', category: '', notifyMethods: []
});
  });
```

Рисунок 3.7 – Unit-тестування NotificationsComponent

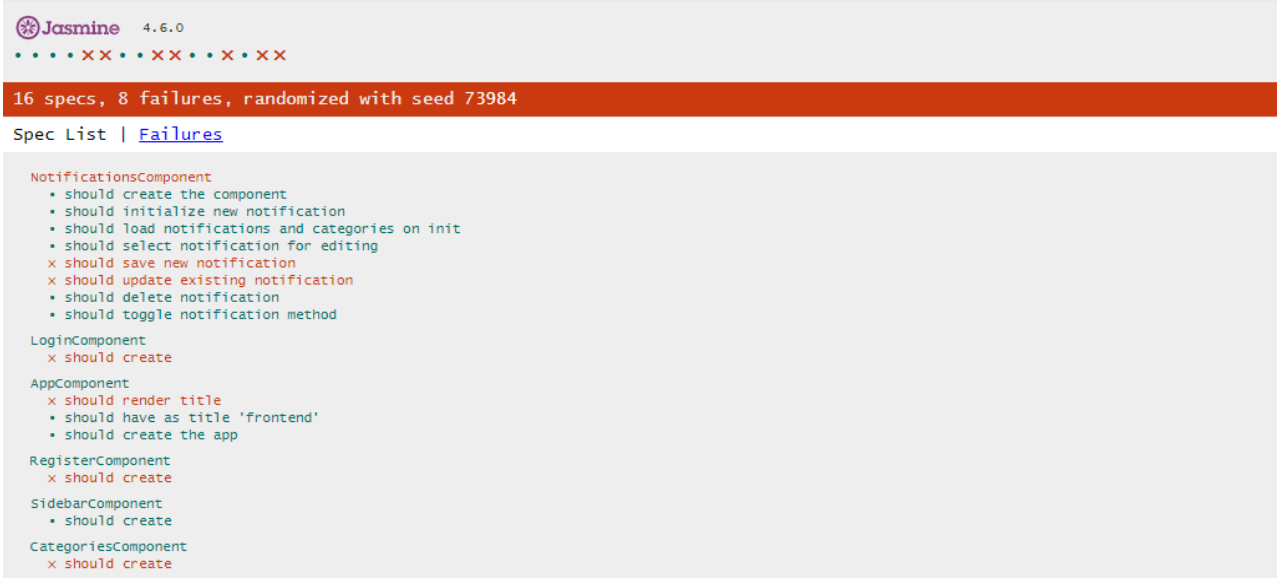

Рисунок 3.8 – Результат тестування NotificationsComponent

3.5 Керівництво користувача

3.5.1 Рівень підготовки користувача

Щоб ефективно користуватися сервісом, необхідно мати базові комп’ютерні навички та досвід роботи з браузерами. Ретельне вивчення

інструкцій користувача допоможе максимально розкрити всі можливості та функціональність сервісу. Ознайомлення з керівництвом забезпечує глибше розуміння принципів роботи сервісу та полегшить освоєння його функцій.

3.5.2 Реєстрація в системі

Перед початком використання сервісу, користувачу необхідно зареєструватися в системі. Для цього, на сторінці входу до системи, потрібно натиснути на кнопку «Зареєструватися», що знаходиться під написом «Не маєте облікового запису?» (рис. 3.9).

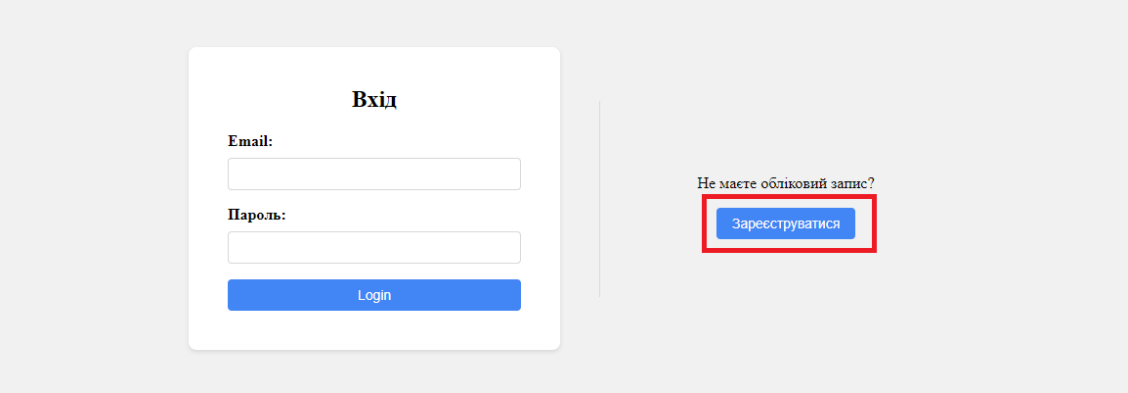

Рисунок 3.9 – Кнопка «Зареєструватися»

Система відображає форму реєстрації. Користувач вводить email та пароль, і натискає на кнопку «Зареєструватися» (рис. 3.10).

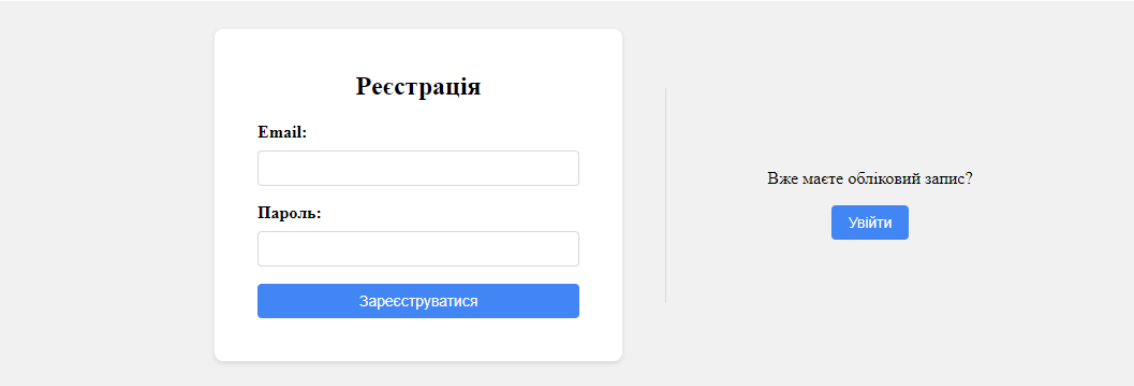

Рисунок 3.10 – Форма реєстрації

Якщо користувача, з вказаним email, вже зареєстровано, то система відображає повідомлення про помилку з текстом «Користувач з таким email вже існує» (рис. 3.11). Користувач може натиснути на кнопку «Увійти», щоб перейти до форми входу до системи.

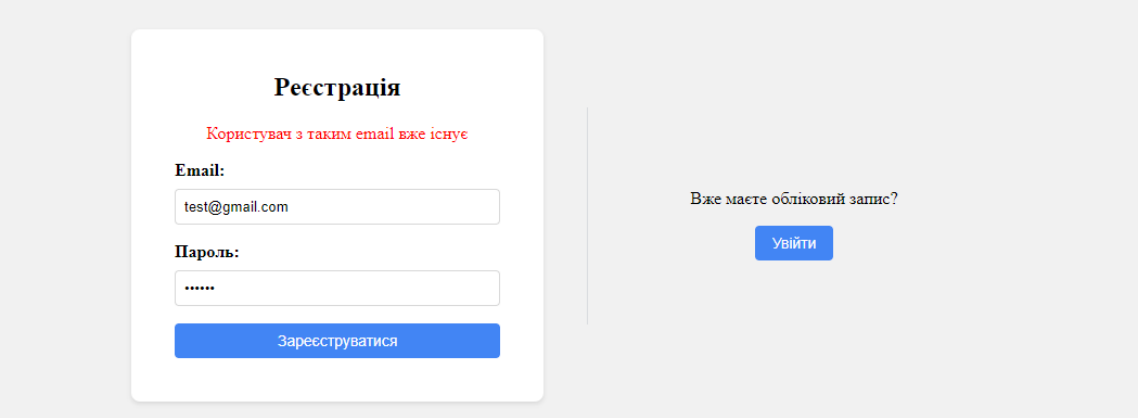

Рисунок 3.11 – Помилка при реєстрації

Якщо реєстрація успішна, то система відображає форму входу до системи.

3.5.3 Вхід до системи

Після того, як користувач зареєструвався або якщо вже є обліковий запис, користувачу необхідно перейти на сторінку входу до системи. Зробити це можна перейшовши за адресою сервіса. Система відображає форму входу до системи (рис. 3.12).

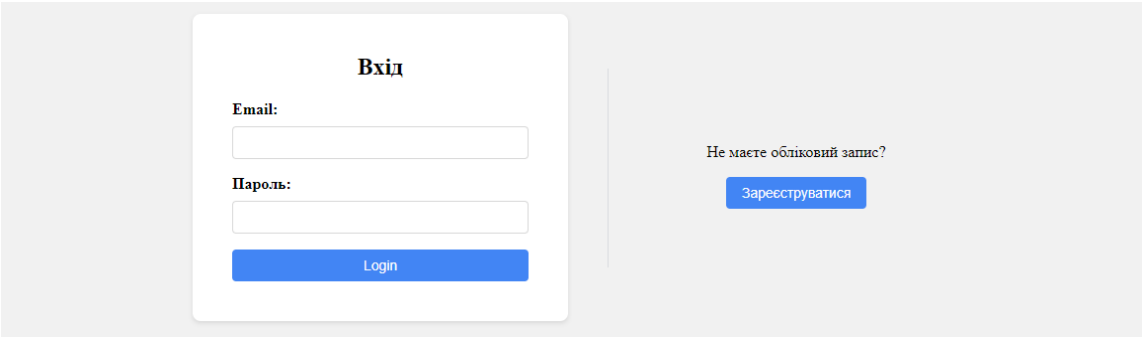

Рисунок 3.12 – Форма входу

Користувач вводить облікові дані і натискає на кнопку «Вхід». Система відображає особистий кабінет (рис. 3.13).

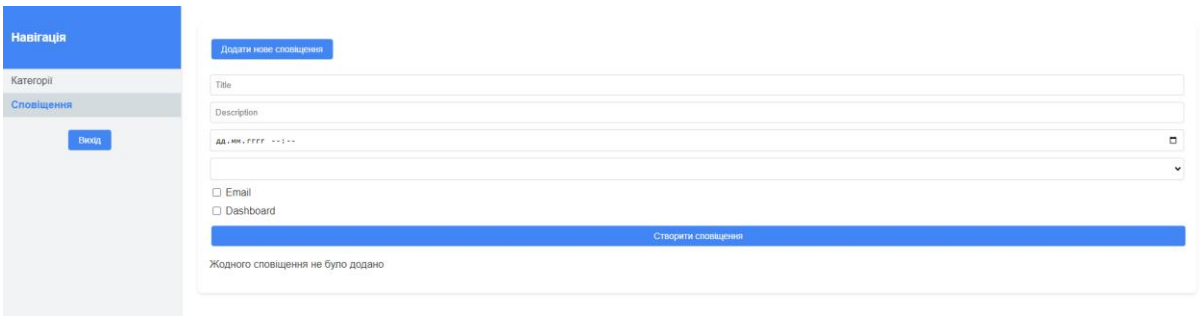

Рисунок 3.13 – Особистий кабінет

Якщо користувача з вказаними email і паролем не існує або введено неправильні дані, то система сповіщає про це. Для кожного з типу помилок відображається відповідне повідомлення (рис. 3.14 – 3.15).

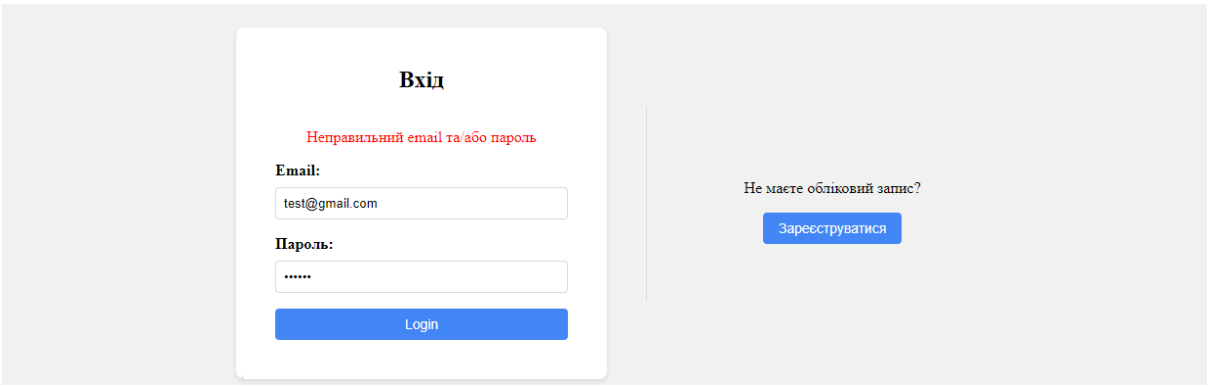

Рисунок 3.14 – Неправильний email та/або пароль

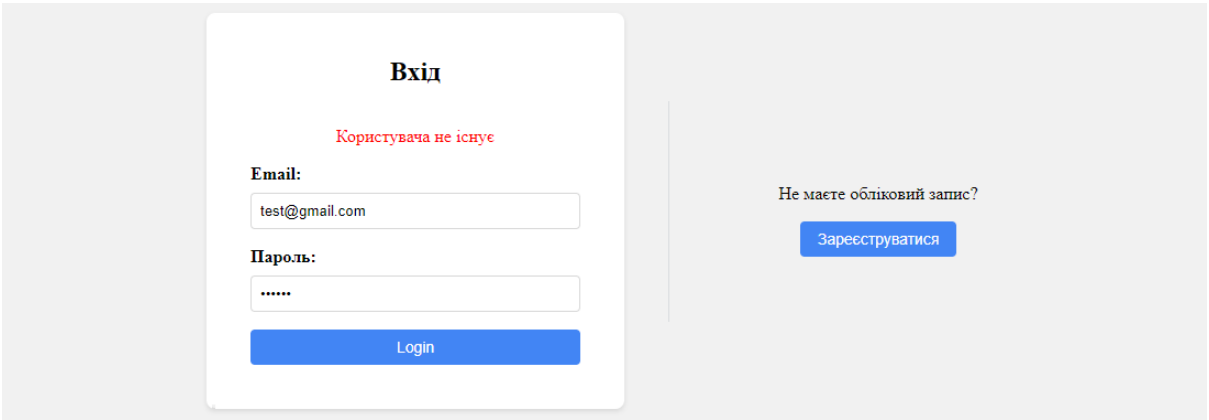

Рисунок 3.15 – Користувача не існує

Помилка може виникнути навіть у тому випадку, коли користувач ввів валідні дані. Це може бути помилка із-за проблем з сервером.

3.5.4 Створення категорії сповіщення

Перед створенням сповіщень, необхідно спочатку створити категорії, до яких вони будуть належати. Для цього користувачу необхідно натиснути на пункт лівобічного меню «Категорії». Система відображає інтерфейс керування категоріями. Якщо жодної категорії не було додано, то відображається повідомлення «Жодної категорії не було додано». Користувач натискає на кнопку «Додати нову категорію». Система відображає форму створення нової категорії. Користувач вводить назву на опис категорії (рис. 3.16).

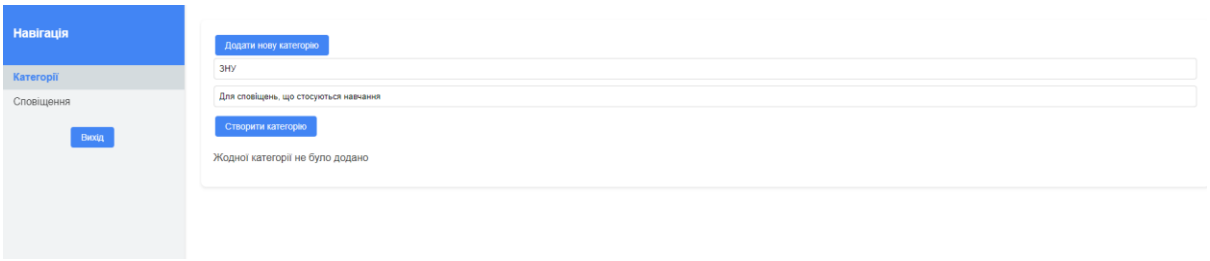

Рисунок 3.16 – Створення нової категорії

Після введення даних, користувач натискає на кнопку «Створити категорію». Система зберігає дані та оновлює список створених категорій (рис. 3.17).

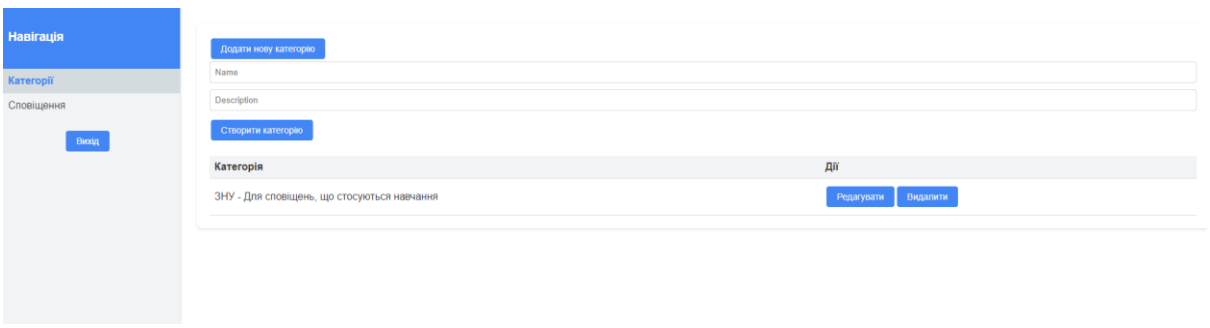

Рисунок 3.17 – Створена категорія

Подібним чином можна створювати інші категорії, які будуть відповідати за конкретні цілі (рис. 3.18).

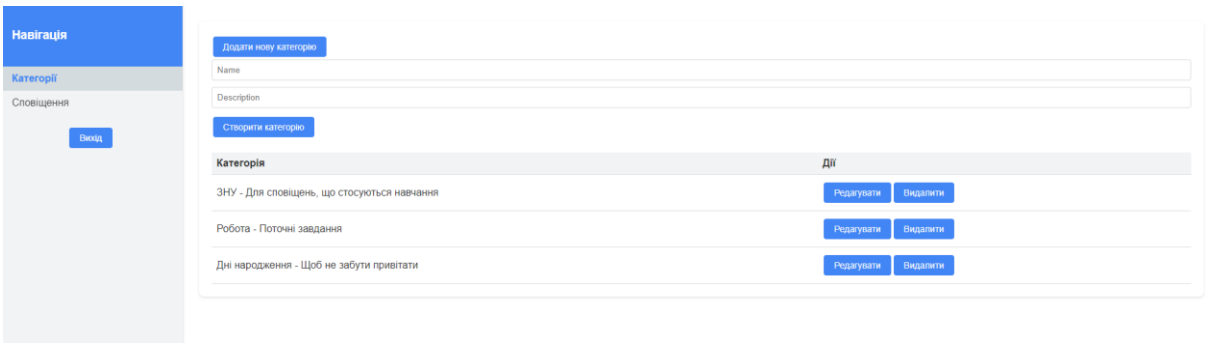

Рисунок 3.18 – Додані категорії

3.5.5 Редагування/видалення категорії

Дані кожної створеної категорії можна змінити. Для цього користувачу необхідно натиснути на кнопку «Редагувати» у рядку відповідної категорії. Система відображає форму створення нової категорії та підтягує в неї дані обраної, також змінюється напис кнопки (рис. 3.19).

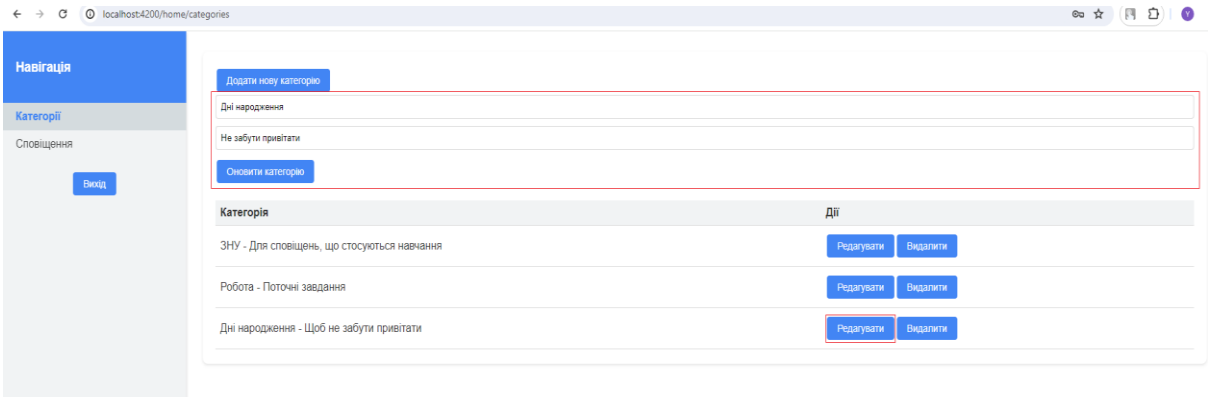

Рисунок 3.19 – Редагування категорії «Дні народження»

Після редагування даних, користувач натискає на кнопку «Оновити категорію». Система оновлює дані та відображає категорії користувача (рис. 3.20).

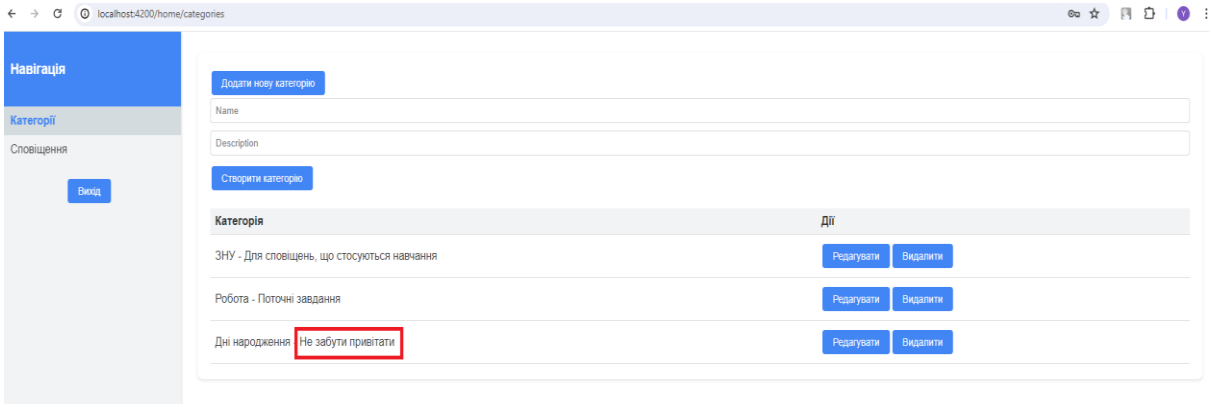

Рисунок 3.20 – Оновлені дані

Для видалення категорії, користувачу необхідно натиснути на кнопку «Видалити» в рядку відповідної категорії. Система видаляє дані категорії та відображає список створених категорій користувача.

3.5.6 Створення сповіщення

Для переходу до інтерфейсу керування сповіщеннями користувачу необхідно натиснути на пункт меню «Сповіщення». Система відображає інтерфейс керування сповіщеннями. Якщо жодного сповіщення не було додано, то відображається повідомлення «Жодного сповіщення не було додано» (рис. 3.21).

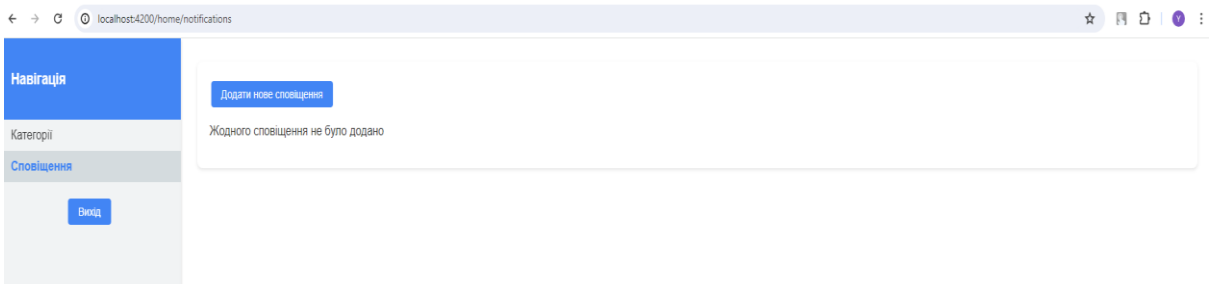

Рисунок 3.21 – Сторінка керування сповіщеннями

Для створення сповіщення користувачу необхідно натиснути на кнопку «Додати нове сповіщення». Система відображає форму створення нового

сповіщення. Користувач вводить назву та опис сповіщення, обирає дату, час, категорію і спосіб сповіщення (рис. 3.22).

Після введення даних, користувач натискає на кнопку «Створити сповіщення». Система зберігає дані та оновлює список створених сповіщень (рис. 3.23).

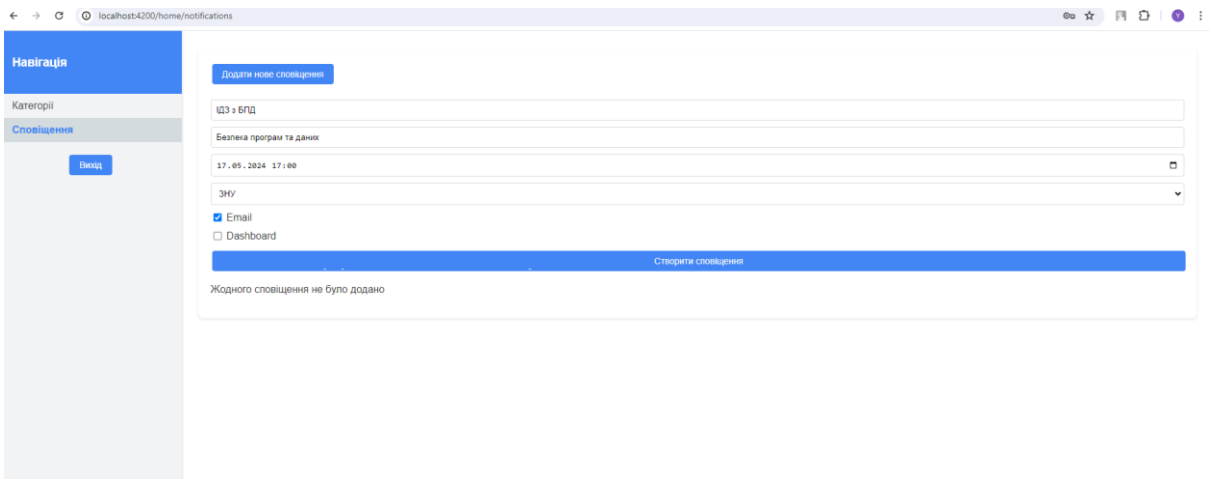

Рисунок 3.22 – Створення нового сповіщення

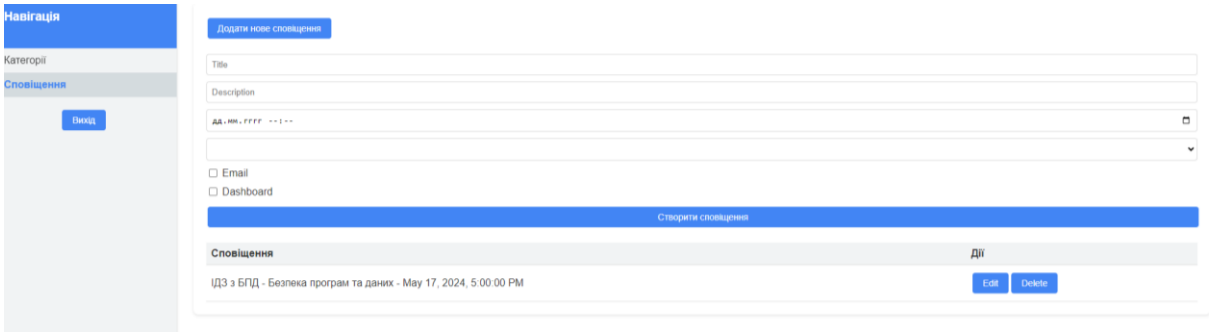

Рисунок 3.23 – Створене сповіщення

Подібним чином можна створювати інші сповіщення, обираючи необхідні категорії, дату та час.

3.5.7 Редагування/видалення сповіщення

За аналогієї з категоріями, дані кожного створенного сповіщення можна змінити. Для цього користувачу необхідно натиснути на кнопку «Редагувати» в

рядку відповідного сповіщення. Система відображає форму створення нового сповіщення та підтягує в неї дані обраного, також змінюється напис кнопки (рис. 3.24).

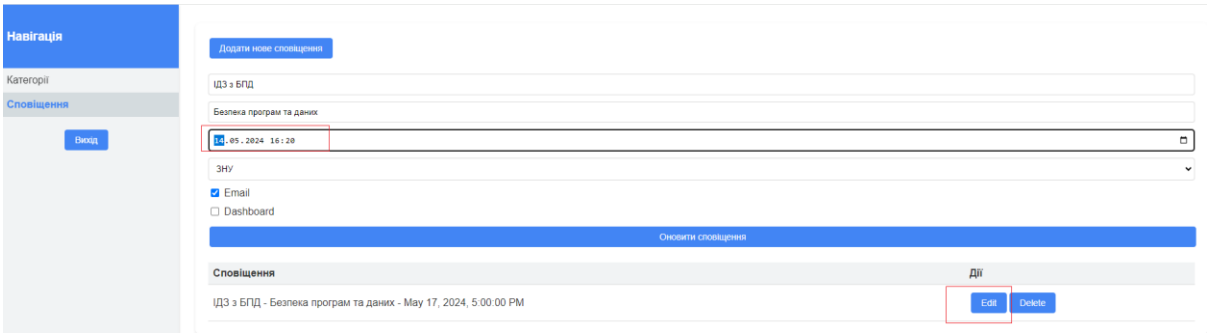

Рисунок 3.24 – Редагування сповіщення

Після редагування даних, користувач натискає на кнопку «Оновити сповіщення». Система оновлює дані та відображає сповіщення користувача (рис. 3.25).

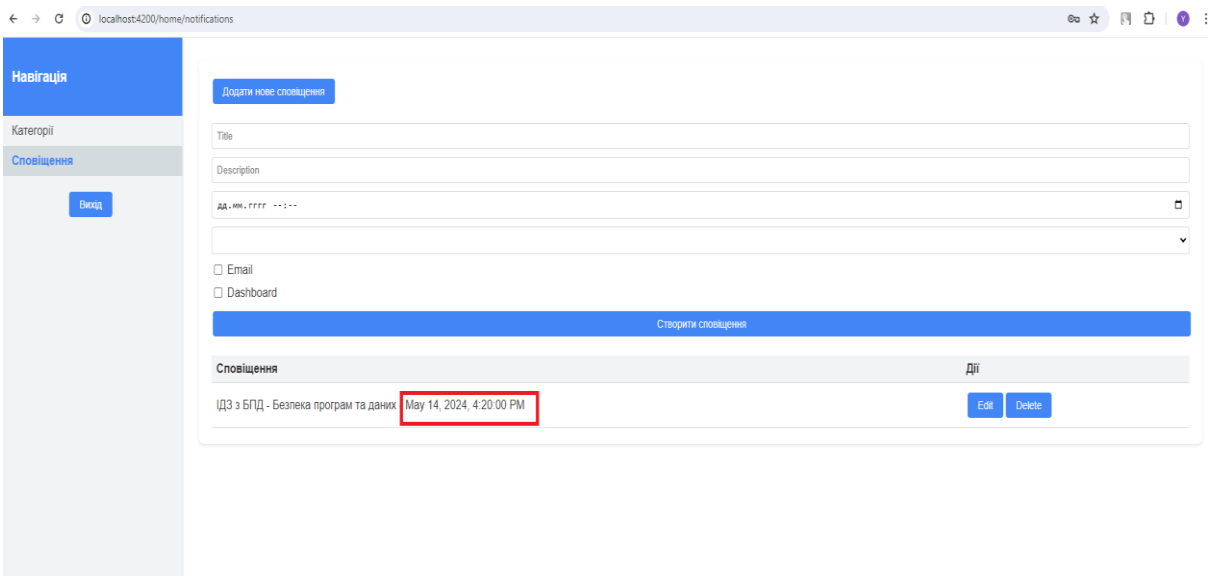

Рисунок 3.25 – Оновлені дані

Для видалення сповіщення, користувачу необхідно натиснути на кнопку «Видалити» у рядку відповідного сповіщення. Система видаляє дані сповіщення та відображає список створених сповіщень користувача (рис. 3.26 – 3.27).

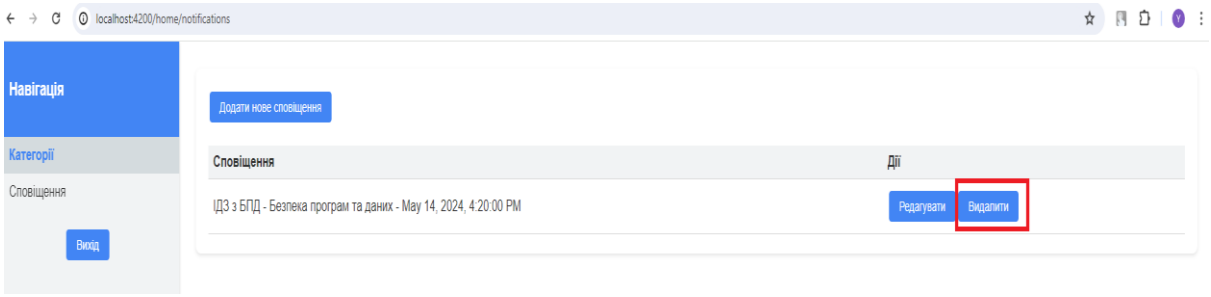

Рисунок 3.26 – Кнопка «Видалити»

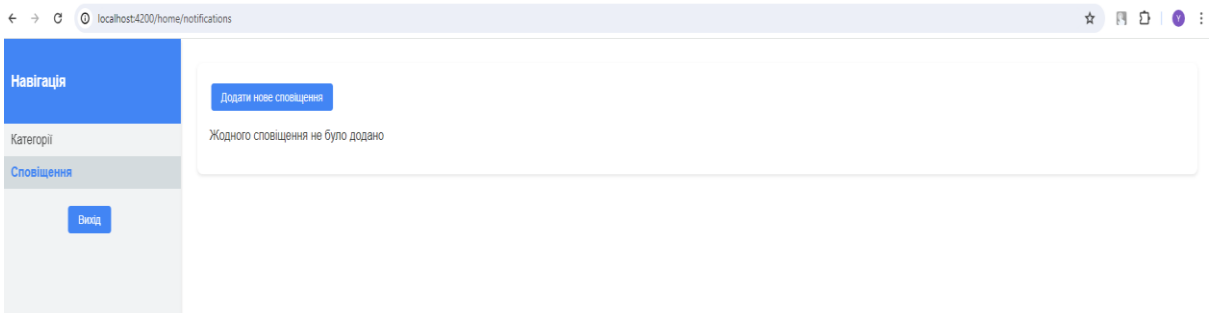

Рисунок 3.27 – Результат видалення

3.5.8 Отримання сповіщення

Система кожну секунду прослуховує наявні сповіщення. Якщо якесь сповіщення підходить за датою та часом, а також знаходиться в статусі «pending», то система надсилає повідомлення згідно обраного методу. Це може бути або повідомлення на пошту, яку зазначив користувач при реєстрації, або повідомлення в особистому кабінеті (рис. 3.28).

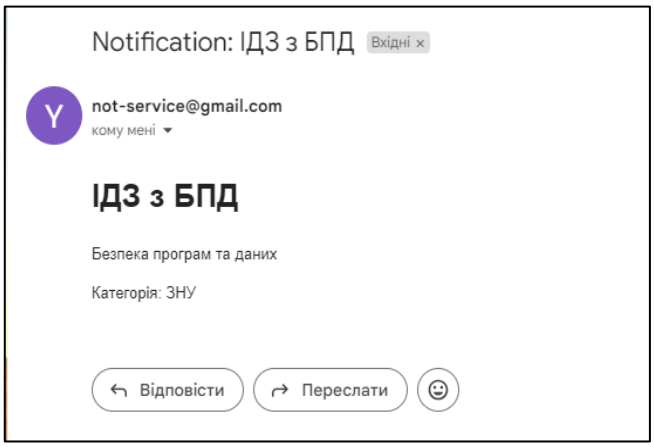

Рисунок 3.28 – Сповіщення на пошту

Система надсилає ті дані, які користувач вказував при створенні сповіщення, окрім дати, часу та методу. Ці дані призначені для системи, щоб розуміти, куди та коли користувачу необхідно надіслати сповіщення. Якщо користувачу необхідна додаткова інформація, то він її повинен вказати в розділі «Опис сповіщення», а також може створити тематичні категорії, які будуть надавати більше інформації.

Після відправки сповіщення, система переводить його в статус «send» (рис. 3.29).

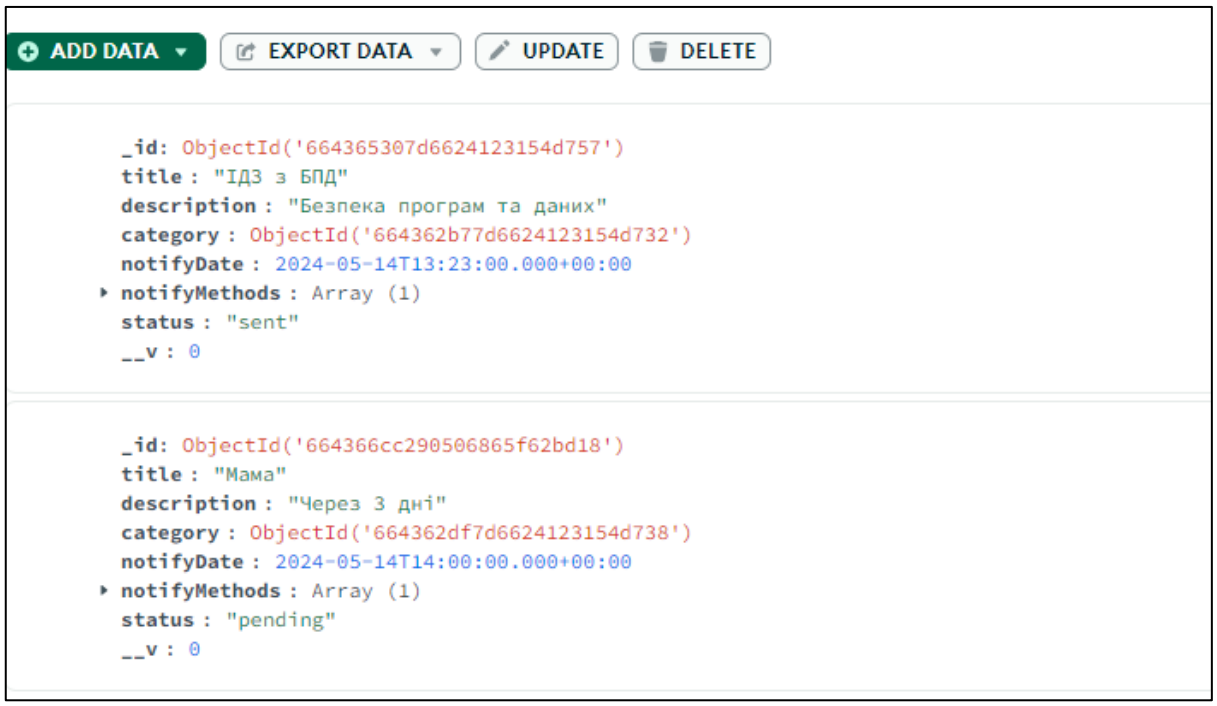

Рисунок 3.29 – Дані сповіщень в БД

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було успішно розроблено сервіс сповіщень, який реалізовано засобами Angular та Node.js. Проведений аналіз предметної області, проєктування системи та її реалізація дозволили досягти основної мети роботи – створення функціонального, надійного та зручного інструменту для сповіщень.

У відповідності з метою кваліфікаційної роботи було розроблено сервіс сповіщень з використаннями наступних технологій:

- MongoDB для зберігання даних;
- Node.js для автризації та відправки запитів до БД;
- Angular для реалізації клієнтської частини, а також відправки запитів до серверу.

У відповідності з поставленими задачами були виконані наступні етапи створення сервісу:

- сформовані вимоги до сервісу (визначено як функціональні, так і нефункціональні вимоги до сервісу сповіщень, що включають інтерфейс користувача, продуктивність, безпеку та підтримку різних браузерів);
- спроєктована та побудована структура сервісу (створено діаграми варіантів використання, діяльності, послідовності та розгортання; детально описані прецеденти системи);
- реалізовано сервіс сповіщень (наведена інструкція по налаштуванню MongoDB Atlas та надано керівництво користувача);
- протестована робота сервісу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Angular Developer Documentation. URL: https://angular.io/docs/ (дата звернення: 10.03.2024).

2. Bampakos A. Angular Projects: Build modern web apps by exploring Angular 12 with 10 different projects and cutting-edge technologies. Birmingham : Packt Publishing, 2021. 344 p.

3. Bampakos A., Deeleman P. Learning Angular: A no-nonsense guide to building web applications with Angular 15. Birmingham : Packt Publishing, 2023. 446 p.

4. Chebbi L. Reactive Patterns with RxJS for Angular: A practical guide to managing your Angular application’s data reactively and efficiently using RxJS 7. Birmingham : Packt Publishing, 2022. 224 p.

5. Fain Y., Moiseev A. Angular Development with TypeScript. New York : Manning Publications, 2019. 560 p.

6. Li D. Introduction to Backend Development with Node.js and Express: Building Robust Web Applications with Node.js and Express Framework. Vancouver : Kindle Edition, 2023. 94 p.

7. MongoDB Documentation. URL: https://www.mongodb.com/docs/ (дата звернення: 14.03.2024).

8. Node.js Developer Documentation. URL: https://nodejs.org/en/docs (дата звернення: 12.03.2024).

9. Rappl F. Modern Frontend Development with Node.js: A compendium for modern JavaScript web development within the Node.js ecosystem. Birmingham : Packt Publishing, 2022. 208 p.

10. Unhelkar B. Software Engineering with UML. Boca Raton : Auerbach Publications, 2020. 426 p.

ДОДАТОК А

Серверна частина

```
const express = require('express');
const Notification = require('../models/Notification');
const authenticateToken = require('../middleware/authenticateToken');
const router = express.Router();

router.post('/', async (req, res) => {
  try {
    const { title, description, category, notifyDate, notifyMethods } = req.body;
    const newNotification = new Notification({
      title,
      description,
      category,
      notifyDate,
      notifyMethods
    });
    await newNotification.save();
    res.status(201).send(newNotification);
  } catch (error) {
    res.status(400).send(error.message);
  }
});



router.put('/:id', async (req, res) => {
  try {
    const updateData = req.body;
    const notification = await Notification.findOneAndUpdate({ _id: req.params.id }, updateData, { new: true
});
    if (!notification) {
      return res.status(404).send('Notification not found or not authorized');
    }
    res.send(notification);
  } catch (error) {
```

```
    res.status(400).send(error.message);
  }
});

router.delete('/:id', async (req, res) => {
  try {
    const notification = await Notification.findOneAndDelete({ _id: req.params.id });
    if (!notification) {
      return res.status(404).send('Notification not found or not authorized');
    }
    res.status(204).send();
  } catch (error) {
    res.status(500).send(error.message);
  }
});


router.get('/history', async (req, res) => {
  try {
    const notifications = await Notification.find({}).sort({ notifyDate: -1 });
    res.send(notifications);
  } catch (error) {
    res.status(500).send(error.message);
  }
});

module.exports = router;

const mongoose = require('mongoose');

const notificationSchema = new mongoose.Schema({
  title: { type: String, required: true },
  description: { type: String },
  category: { type: mongoose.Schema.Types.ObjectId, ref: 'Category' },
  notifyDate: { type: Date, required: true },
  notifyMethods: { type: [String], enum: ['email', 'dashboard'], required: true },
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: false },
  status: { type: String, enum: ['pending', 'sent'], default: 'pending' }
```

```
});

module.exports = mongoose.model('Notification', notificationSchema);

const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const nodemailer = require('nodemailer');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
const schedule = require('node-schedule');
const socketIo = require('socket.io');
const http = require('http');
const cors = require('cors');

const userRoutes = require('./routes/users');
const notificationRoutes = require('./routes/notifications');
const categoryRoutes = require('./routes/categories');

//const User = require('./models/User'); // Модель користувача
const Notification = require('./models/Notification'); // Модель сповіщення
const Category = require('./models/Category');

const app = express();
const server = http.createServer(app);
const io = socketIo(server); // WebSocket зв'язок

app.use(cors());
app.use(bodyParser.json());

app.use('/api/auth', userRoutes);
app.use('/api/notifications', notificationRoutes);
app.use('/api/categories', categoryRoutes);

// Налаштування SMTP для nodemailer
const transporter = nodemailer.createTransport({
  host: 'smtp.gmail.com',
  port: 587,
```

```
  secure: false,
  auth: {
    user: 'krivoy.znu@gmail.com',
    pass: 'jfqw eoya fscv zipt'
  }
});

// Підключення до MongoDB
mongoose.connect('mongodb+srv://admin:maladMin_235@notification.6oye16z.mongodb.net/', {
useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.error('MongoDB connection error:', err));

// Відправлення електронної пошти
function sendEmail(to, subject, html) {
    transporter.sendMail({ from: 'krivoy.znu@gmail.com', to, subject, html }, (err, info) => {
    if (err) console.error('Error sending email:', err);
    else console.log('Email sent:', info.response);
  });
}

// WebSocket обробка
io.on('connection', socket => {
  console.log('New client connected');

  socket.on('disconnect', () => {
    console.log('Client disconnected');
  });
});

app.get('/test-notifications', async (req, res) => {
    const hashedPassword = await bcrypt.hash('123', 10);
    const check = await bcrypt.compare('123', hashedPassword);
    console.log('check ', check);
});

// Функція для запланування відправлення сповіщень
function scheduleNotifications() {
```

```
  Notification.find({ status: 'pending', notifyDate: { $lte: new Date() } })
    .then(notifications => {
      notifications.forEach(notification => {
      if (notification.notifyMethods.includes('email')) {
         sendEmail('krivoyyaroslav21@gmail.com', `Notification: ${notification.title}`,
`<h1>${notification.title}</h1><p>${notification.description}</p><p>Category:
${notification.category}</p>`);
      }

      if (notification.notifyMethods.includes('dashboard')) {
        io.emit('notification', { message: `You have a new notification: ${notification.title} -
${notification.description}` });
      }

      Notification.findByIdAndUpdate(notification._id, { status: 'sent' }).exec();
     });
    })
    .catch(err => console.error('Error scheduling notifications:', err));
}

schedule.scheduleJob('* * * * *', scheduleNotifications);

const PORT = process.env.PORT || 3000;
server.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

ДОДАТОК Б

Тестування NotificationsComponent

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { NotificationsComponent } from './notifications.component';
import { NotificationService } from '../../services/notification.service';
import { CategoryService } from '../../services/category.service';
import { of } from 'rxjs';

describe('NotificationsComponent', () => {
  let component: NotificationsComponent;
  let fixture: ComponentFixture<NotificationsComponent>;
  let notificationServiceMock: jasmine.SpyObj<NotificationService>;
  let categoryServiceMock: jasmine.SpyObj<CategoryService>;

  beforeEach(() => {
    const notificationServiceSpy = jasmine.createSpyObj('NotificationService', ['getNotificationsHistory',
'updateNotification', 'createNotification', 'deleteNotification']);
    const categoryServiceSpy = jasmine.createSpyObj('CategoryService', ['getCategories']);

    TestBed.configureTestingModule({
      declarations: [NotificationsComponent],
      providers: [
        { provide: NotificationService, useValue: notificationServiceSpy },
        { provide: CategoryService, useValue: categoryServiceSpy }
      ]
    });

    notificationServiceMock = TestBed.inject(NotificationService) as jasmine.SpyObj<NotificationService>;
    categoryServiceMock = TestBed.inject(CategoryService) as jasmine.SpyObj<CategoryService>;

    fixture = TestBed.createComponent(NotificationsComponent);
    component = fixture.componentInstance;
  });

  it('should create the component', () => {
```

```
    expect(component).toBeTruthy();
  });

  it('should load notifications and categories on init', () => {
    const mockNotifications = [{ title: 'Notification 1' }, { title: 'Notification 2' }];
    const mockCategories = [{ name: 'Category 1' }, { name: 'Category 2' }];

    notificationServiceMock.getNotificationsHistory.and.returnValue(of(mockNotifications));
    categoryServiceMock.getCategories.and.returnValue(of(mockCategories));

    component.ngOnInit();

    expect(component.notifications).toEqual(mockNotifications);
    expect(component.categories).toEqual(mockCategories);
  });

  it('should initialize new notification', () => {
    component.initNewNotification();

    expect(component.currentNotification).toEqual({ title: '', description: '', notifyDate: '', category: '',
notifyMethods: [] });
    expect(component.editMode).toBeFalsy();
  });

  it('should select notification for editing', () => {
    const mockNotification = { _id: '1', title: 'Test', description: 'Test Notification', notifyDate: '2023-05-
15T10:00:00Z', category: '1', notifyMethods: ['email'] };

    component.selectNotification(mockNotification);

    expect(component.currentNotification).toEqual({ ...mockNotification, notifyDate: '2023-05-15T10:00' });
    expect(component.editMode).toBeTruthy();
  });

  it('should save new notification', () => {
    const mockNotification = { title: 'New Notification', description: 'New Notification', notifyDate: '2023-05-
16T12:00:00Z', category: '2', notifyMethods: ['dashboard'] };
    component.currentNotification = mockNotification;
```

```
    notificationServiceMock.createNotification.and.returnValue(of(null));

    component.saveNotification();

    expect(notificationServiceMock.createNotification).toHaveBeenCalledWith(mockNotification);
    expect(notificationServiceMock.getNotificationsHistory).toHaveBeenCalled();
    expect(component.currentNotification).toEqual({ title: '', description: '', notifyDate: '', category: '',
notifyMethods: [] });
  });

  it('should update existing notification', () => {
    const mockNotification = { _id: '1', title: 'Updated Notification', description: 'Updated Notification',
notifyDate: '2023-05-17T14:00:00Z', category: '1', notifyMethods: ['email', 'dashboard'] };
    component.currentNotification = mockNotification;
    component.editMode = true;

    notificationServiceMock.updateNotification.and.returnValue(of(null));

    component.saveNotification();

    expect(notificationServiceMock.updateNotification).toHaveBeenCalledWith('1', mockNotification);
    expect(notificationServiceMock.getNotificationsHistory).toHaveBeenCalled();
    expect(component.currentNotification).toEqual({ title: '', description: '', notifyDate: '', category: '',
notifyMethods: [] });
  });

  it('should delete notification', () => {
    const notificationId = '2';

    notificationServiceMock.deleteNotification.and.returnValue(of(null));

    component.deleteNotification(notificationId);

    expect(notificationServiceMock.deleteNotification).toHaveBeenCalledWith(notificationId);
    expect(notificationServiceMock.getNotificationsHistory).toHaveBeenCalled();
  });

  it('should toggle notification method', () => {
```

```
    component.currentNotification.notifyMethods = ['email'];

    component.toggleNotificationMethod('email');
    expect(component.currentNotification.notifyMethods).toEqual([]);

    component.toggleNotificationMethod('dashboard');
    expect(component.currentNotification.notifyMethods).toEqual(['dashboard']);
  });
});
```