МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБДОДАТКУ ПЛАНУВАЛЬНИКА ЗАВДАНЬ ЗАСОБАМИ ANGULAR»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М. Лахуібі
(ініціали та прізвище)
Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(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 СТУДЕНТОВІ

Лахуібі Мураду
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебдодатку планувальника завдань засобами Angular

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(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.	22.01.2024	
2.	Збір вихідних даних.	16.02.2024	
3.	Обробка методичних та теоретичних	11.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	19.04.2024	
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 _____ М. Лахуібі
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебдодатку планувальника завдань засобами Angular»: 60 с., 38 рис., 1 табл., 17 джерел, 2 додатки.

ANGULAR, API, DB, MONGO, MVC, NODE.JS, TASK SCHEDULER SERVICE, TYPESCRIPT, UML.

Об’єкт дослідження – процес розробки вебдодатку планувальника завдань.

Мета роботи – розробити вебдодаток планувальника завдань.

Методи дослідження – моделювання, проєктування, програмний, аналітичний.

У роботі розглянуто проєктування та реалізацію вебдодатку планувальника завдань з використанням Angular. Застосовано UML для візуалізації системи, зокрема діаграми варіантів використання, діяльності, послідовності та розгортання.

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

Таким чином, розроблений вебдодаток планувальника завдань забезпечує зручний інтерфейс для керування завданнями, дозволяючи користувачам легко створювати та налаштовувати завдання. Використання Angular, Node.js та сучасних підходів до розробки програмного забезпечення сприяло створенню ефективної та масштабованої системи.

SUMMARY

Bachelor’s qualifying paper “Development of a Task Scheduler Web Application Using Angular”: 60 pages, 38 figures, 1 table, 17 references, 2 supplements.

ANGULAR, API, DB, MONGO, MVC, NODE.JS, TASK SCHEDULER SERVICE, TYPESCRIPT, UML.

The object of the study is the process of developing a task scheduler web application.

The aim of the study is to develop a task scheduler web application.

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

The paper discusses the design and implementation of a task scheduler web application using Angular. UML was used to visualize the system, including use case diagrams, activities, sequences, and deployments.

The system provides the ability to create, view, edit, and delete tasks. User registration/authentication, task and metadata storage in MongoDB were implemented. Unit and integration testing was performed.

Thus, the developed task scheduler web application provides a user-friendly interface for managing tasks, allowing users to easily create and customize tasks. The use of Angular, Node.js and modern software development approaches contributed to the creation of an efficient and scalable system.

ЗМІСТ

ВСТУП

Розробка вебдодатків для планування завдань є актуальним і важливим завданням у сучасному світі, де ефективне керування часом, організація робочих процесів та продуктивність відіграють ключову роль. Такі додатки дозволяють користувачам створювати, редагувати та відслідковувати виконання завдань, що сприяє підвищенню продуктивності та забезпечує своєчасне виконання робочих обов'язків. Вони також є зручним інструментом для взаємодії між членами команд, допомагаючи координувати спільну роботу та відстежувати прогрес проєктів.

Планувальники завдань надають можливість ефективно організовувати робочий час, розставляти пріоритети та уникати затримок у виконанні завдань. Такі додатки допомагають підтримувати високу продуктивність, уникати перевантажень та забезпечувати ефективне використання робочого часу.

Крім того, вебдодатки для планування завдань сприяють підвищенню дисципліни та відповідальності користувачів. Вони дозволяють контролювати виконання завдань, отримувати нагадування про наближення дедлайнів та забезпечувати своєчасне виконання робіт. Сучасні рішення також дозволяють персоналізувати інтерфейс та функціональні можливості, враховуючи індивідуальні потреби користувачів, роблячи процес планування більш зручним та ефективним.

Виходячи з цього, було створено вебдодаток для планування завдань, який дозволяє швидко та зручно створювати нові завдання, керувати ними та отримувати сповіщення про наближення дедлайнів.

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

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

Мета: розробити вебдодаток планувальника завдань.

Задачі:

– сформувати вимоги до вебдодадку;
– спроєктувати та побудувати архітектуру вебдодадку;
– реалізувати вебдодаток планувальника завдань;
– протестувати роботу вебдодадку.

Об’єкт дослідження: процес розробки планувальника завдань, інструменти для роботи з календарем та повідомленнями.

Предмет дослідження: фреймворк Angular.

Методи дослідження: моделювання, проєктування, програмний, аналітичний.

Перший розділ присвячено збору та аналізуванню вимог до вебдодатку, оглядута порівнянню аналогів, опису системи і предметної області.

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

Третій розділ присвячено реалізації та тестуванню роботи вебдодатку, наведено керівництво користувача, яке описує процес роботи із вебдодатком планувальника завдань.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Система – вебдодаток планувальника завдань, створений засобами Angular.

Angular – це вебфреймворк, розроблений компанією Google, який використовується для створення динамічних односторінкових додатків (Single Page Applications, SPA).

Node.js – це середовище виконання для запуску JavaScript-коду поза веббраузером.

ДВВ – Діаграма варіантів використання.

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 з нею взаємодіяти.

1.1.2 Технічні терміни

NoSQL БД – документо-орієнтована база даних.

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

Framework – це програмна оболонка, яка дозволяє спростити і прискорити вирішення типових завдань.

Компонент – структурна одиниця Angular.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати можливість створювати та керувати завданнями.

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

Мета створення системи – розробка вебдодатку планувальника завдань.

1.2.2 Загальні функціональні можливості системи

Система має надавати гостям такі можливості:

– реєстрація/ вхід в/до системі(и);

Система має надавати авторизованим користувачам такі можливості:

– створення/видалення/перегляд/редагування/зміна статусу завдання;
– перегляд/фільтрування списку завдань;
– надсилання завдання на email;
– отримання сповіщення;
– вихід з системи.

1.3 Нефункціональні вимоги

1.3.1 Інтерфейс користувача

Інтерфейс системи має бути адаптивним та інтуїтивно зрозумілим. Це дозволить відображати його належним чином на різних пристроях та забезпечить зручність використання як для досвідчених користувачів, так і для новачків.

1.3.2 Підтримка браузерів

Важливо забезпечити безперебійну роботу системи в найновіших версіях провідних веббраузерів, серед яких Mozilla Firefox, Google Chrome, Safari, Microsoft Edge та Opera. Це дозволить максимально розширити коло користувачів, які матимуть доступ до даної системи.

1.3.3 Вимоги до продуктивності

Сторінки системи повинні завантажуватися максимально швидко, за час, що не перевищує одну секунду. Водночас, операції створення та отримання завдань не мають призводити до сповільнення загальної продуктивності та швидкодії системи.

1.3.4 Вимоги до безпеки

Доступ до функцій системи повинен надаватися виключно авторизованим користувачам. Це необхідно для захисту даних та запобігання несанкціонованому використанню.

Система має включати захисні механізми, які унеможливлюватимуть введення користувачами даних, що можуть становити загрозу безпеці та використовуватися як експлойти (шкідливий код або команди).

1.4 Опис предметної області

Предметною областю є розробка вебдодатку планувальника завдань. Дана система повинна надавати користувачам можливість створювати завдання та взаємодіяти з ними. Процес взаємодії з системою залежить від ролі користувача.

Для гостя доступні лише функції створення облікового запису та входу. Після цього він, як авторизований користувач, може створювати завдання та керувати ними.

Процес створення завдання відбувається наступним чином. Користувач входить до облікового запису, система відображає інтерфейс планувальника завдань. Для створення нового завдання, користувач натискає на необхідну дату, система відображає форму створення нового завдання, де користувач може ввести дані. Після введення необхідних даних, користувач натискає на кнопку «Додати». Система зберігає дані завдання до БД, відображає повідомлення про успішну операцію та відображає інтерфейс планувальника завдань.

1.5 Опис системи

Створення ефективних вебдодатків для планування завдань є вкрай актуальним у сучасному динамічному світі, де своєчасне виконання завдань, оптимальна організація робочого процесу та залученість користувачів відіграють ключову роль. Такі додатки дозволяють користувачам швидко ставити перед собою цілі, розбивати їх на конкретні завдання, визначати терміни та пріоритети, забезпечуючи тим самим підвищену продуктивність та контроль над виконанням запланованого.

Вебдодатки для планування завдань також є зручним інструментом для організацій, компаній та команд, допомагаючи координувати спільну роботу, розподіляти обов’язки, відстежувати прогрес та забезпечувати своєчасне завершення проєктів. Вони сприяють ефективній комунікації та співпраці, підтримуючи постійний зв’язок між учасниками та інформуючи їх про нові завдання, зміни пріоритетів чи терміни виконання.

Крім того, планувальники завдань можуть підвищувати залученість користувачів та мотивувати їх досягати поставлених цілей через зручні нагадування, відстеження прогресу та можливість візуалізації виконаної роботи.

Сучасні рішення дозволяють персоналізувати інтерфейс, налаштування та функціональність під індивідуальні потреби користувачів, роблячи планування більш ефективним і зручним.

З огляду на ці переваги було створено цей вебдодаток планувальника завдань, який пропонує зручний інструмент для організації та керування завданнями в зрозумілому та інтуїтивному інтерфейсі.

1.6 Огляд і порівняння аналогів

Планування та ефективне керування завданнями стало невід'ємною частиною як особистої продуктивності, так і успіху бізнес-проєктів у сучасному швидкоплинному світі.

На ринку представлено чимало програмних рішень, які полегшують цей процес, від простих todo-списків до повнофункціональних систем управління проєктами.

Розглянемо кілька популярних аналогів вебдодатків для планування завдань та порівняємо їх з нашим власним рішенням.

Нижче наведено стислий опис п'яти конкурентних сервісів, а також таблиця, що демонструє їхні ключові можливості у зіставленні з функціоналом нашого вебдодатку.

Trello – популярний додаток на основі дошок та карток. Він візуалізує робочий процес та дозволяє призначати завдання, встановлювати терміни, додавати вкладення та коментарі. Trello має безкоштовну і преміум-версії [16].

Asana – потужна система для управління проєктами та завданнями з можливістю відстеження часу, створення звітів, інтеграцією з іншими інструментами. Має безкоштовний тарифний план і платні преміум-рівні [2].

Todoist – простий і зручний додаток для планування завдань із можливістю ієрархічної структури завдань, пріоритетів, нагадувань та синхронізації між пристроями. Пропонує безкоштовну та преміум підписку [15].

Microsoft To Do – інтегрований з Microsoft 365 планувальник завдань із можливістями розділити завдання на список, додавати нагадування та вказувати терміни. Доступний окремо та в пакеті Office 365 [9].

Google Tasks – вбудована в G Suite функція для створення та керування списками завдань, яка інтегрується з календарем та поштою Gmail. Безкоштовна для користувачів Google [7].

Порівняння ключових функцій цих систем наведено в таблиці 1.1.

Наш вебдодаток для планування завдань пропонує базовий набір функцій, зосереджений на керуванні простими лінійними списками завдань з можливістю призначати їх іншим користувачам та встановлювати терміни завершення. Однак він поки що позбавлений деяких корисних можливостей, таких як візуалізація завдань, структурування завдань у ієрархії, налаштування пріоритетів, додавання вкладень та нагадувань.

Порівняно з конкурентами, наш додаток виглядає дещо обмеженим у функціоналі. Такі рішення, як Trello, Asana та Todoist, пропонують більш розширені можливості візуалізації, структурування завдань, встановлення пріоритетів, нагадувань та роботи з вкладеннями.

Водночас, пріоритетом нашого додатку є простота використання та безкоштовний базовий доступ без обмежень.

Для підвищення конкурентоспроможності на ринку планувальників завдань нам варто розглянути можливість включення додаткових функцій, як-от канбан-дошки для візуалізації, ієрархічна структура завдань, підтримка пріоритетів та вкладень. Це дозволить охопити ширшу аудиторію користувачів з різними потребами та побажаннями. Також доцільно зберегти безкоштовну необмежену версію як ключову перевагу.

Загалом, поточна версія нашого додатку може задовольнити базові потреби в простому плануванні завдань, проте для досягнення більшої конкурентоспроможності необхідно вдосконалювати та розширювати функціональні можливості.

Таблиця 1.1 – Порівняння аналогів

Функція	**Наш додаток**	**Trello**	**Asana**	**Todoist**	**Microsoft To Do**	**Google Tasks**
Візуалізація завдань	Списки	Канбан-дошки	Канбан-дошки, списки	Списки	Списки	Списки
Структура/ ієрархія завдань	Лінійний список	Гнучкі дошки з картками	Підзадачі	Багаторівнева вкладеність	Лінійний список	Лінійний список
Призначення завдань	Для окремих користувачів	Для окремих учасників	На команди/ окремих людей	Немає	Немає	Немає
Терміни/ дедлайни	Дати завершення	Дати завершення	Дати завершення	Дати завершення	Дати завершення	Немає
Пріоритети	Немає	Високий, середній, низький	Високий, середній, низький	4 рівні пріоритетів	Немає	Немає
Нагадування	Ел. пошта	Немає	Ел. пошта, мобільні сповіщення	Пуш, ел. пошта, дзвінки	Ел. пошта	Немає
Додавання вкладень	Немає	Файли, зображення, посилання	Файли всіх типів	Файли, зображення	Немає	Немає
Інтеграція з іншими сервісами	Висока	Понад 200 інтеграцій	Багато інтеграцій	Бізнес-інструменти	Microsoft 365	G Suite
Безкоштовна	Так	Обмежена	Обмежена	Обмежена	Немає	Так

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

Використання уніфікованої мови моделювання (UML) під час розробки системи є невід'ємною складовою сучасних методологій проєктування програмного забезпечення. UML забезпечує стандартизовану нотацію та термінологію для візуального представлення структури, поведінки та взаємозв'язків між компонентами системи. Її застосування дозволяє досягти низки переваг на різних етапах життєвого циклу розробки.

На ранніх стадіях проєктування UML сприяє візуалізації та документуванню концептуальної моделі системи. Використовуючи такі діаграми, як діаграми класів, діаграми послідовностей чи діаграми станів, розробники можуть провести детальний аналіз вимог, виявити потенційні проблеми та обговорити можливі підходи до реалізації ще до початку безпосереднього кодування [17]. Це дозволяє зменшити ризики та витрати, пов'язані з необхідністю внесення змін на пізніших етапах розробки.

Під час реалізації системи UML забезпечує ефективну комунікацію між членами команди розробників, а також між розробниками та іншими зацікавленими сторонами, такими як замовники чи менеджери проєктів [17]. Використання стандартизованої нотації та термінології полегшує обмін ідеями, обговорення концепцій та узгодження вимог, зменшуючи ризик непорозумінь та помилок.

На етапі супроводу та модифікації системи діаграми UML відіграють роль наочної документації, яка полегшує розуміння архітектури та коду розробниками, що приєднуються до проєкту [17]. Візуалізація структури та поведінки системи допомагає ідентифікувати точки внесення змін, оцінити їх вплив та забезпечити цілісність системи в процесі її еволюції.

Таким чином, використання UML під час розробки системи є важливою практикою, яка забезпечує візуалізацію та документування архітектури, полегшує комунікацію між зацікавленими сторонами, зменшує ризики та витрати на ранніх етапах проєктування, а також сприяє ефективному супроводу та модифікації системи протягом її життєвого циклу.

2.2 Діаграма варіантів використання

Діаграма варіантів використання (use case diagram) є одним з ключових видів діаграм в уніфікованій мові моделювання (UML), що використовується для візуалізації функціональних вимог до системи з точки зору різних акторів, які взаємодіють з нею [17]. Ця діаграма дозволяє зобразити функціональність системи у вигляді набору варіантів використання, що представляють окремі послідовності дій, які виконуються системою для досягнення певної мети [17].

Діаграма варіантів використання допомагає зрозуміти та проаналізувати вимоги до системи, забезпечуючи наочне представлення її функціональності. Вона дозволяє ідентифікувати акторів, які взаємодіють з системою, та визначити, які варіанти використання вони ініціюють або беруть участь [17]. Крім того, діаграма допомагає виявити взаємозв’язки та залежності між різними варіантами використання, що є важливим для розуміння цілісності системи.

Діаграма варіантів використання є корисним інструментом як для комунікації з зацікавленими сторонами, так і для документування вимог до системи. Вона забезпечує зрозумілу та стандартизовану форму представлення функціональних вимог, що полегшує обговорення та узгодження їх з замовниками та іншими зацікавленими сторонами [17]. Крім того, діаграма може слугувати базою для подальшого проєктування та реалізації системи, забезпечуючи вихідну точку для визначення класів, їх взаємодій та поведінки.

Загалом, діаграма варіантів використання є потужним інструментом для візуалізації, аналізу та документування функціональних вимог до системи. Вона

забезпечує зрозумілу та стандартизовану форму представлення функціональності системи з точки зору різних акторів, що сприяє ефективній комунікації та координації зусиль між різними зацікавленими сторонами на ранніх етапах розробки.

На рисунку 2.1 представлена діаграма варіантів використання сервісу сповіщень.

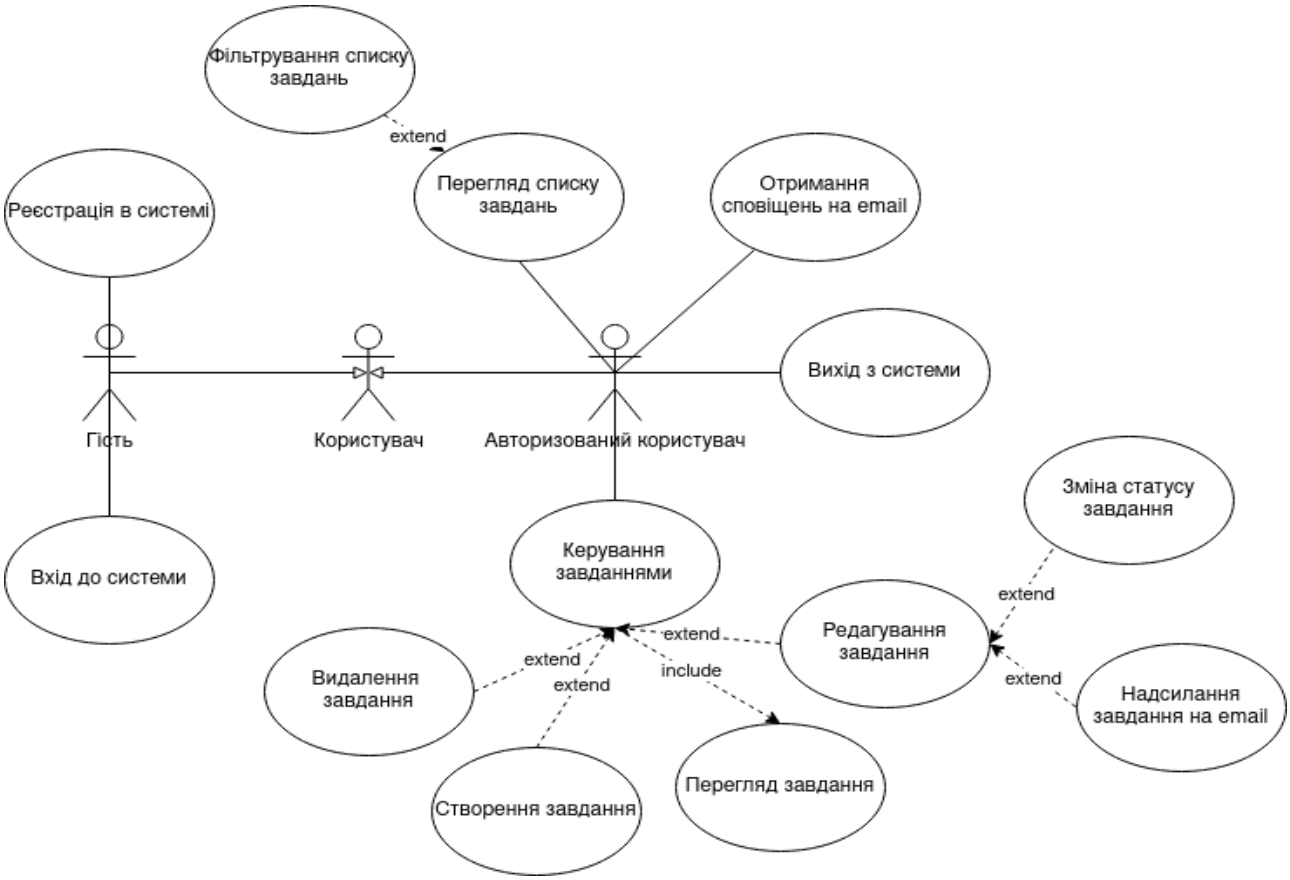

Рисунок 2.1 – Загальна ДВВ

На діаграмі є три актори: «Гість», «Авторизований користувач» та «Користувач». «Гість» та «Авторизований користувач» можуть взаємодіяти з системою, виконуючи різні дії. Актор «Користувач» є узагальненням, який об’єднує можливості інших двох акторів.

Виділено 1 основний варіант використання – «Створення завдання». Спочатку користувач авторизується в системі. Після успішного входу система показує йому інтерфейс планувальника завдань. Щоб створити нове завдання,

користувач клікає на потрібну дату в календарі, і система відкриває форму для введення деталей нового завдання. Після заповнення необхідних полів користувач натискає кнопку «Додати». Тоді система зберігає інформацію про нове завдання в базі даних, повідомляє користувачеві про успішне збереження і повертається до перегляду планувальника завдань з новим завданням.

Варіанти використання – це інструмент для визначення та опису функціональних можливостей програмної системи з точки зору її користувачів або зовнішніх сутностей (акторів). Кожен варіант використання описує певний сценарій або послідовність дій, які актор може виконати при взаємодії з системою для досягнення конкретної мети. Тобто, варіанти використання показують різні способи використання функціоналу системи її користувачами для вирішення їхніх завдань.

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

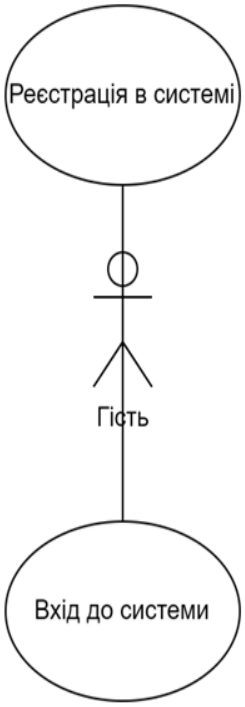

Рисунок 2.2 – ДВВ «Гість»

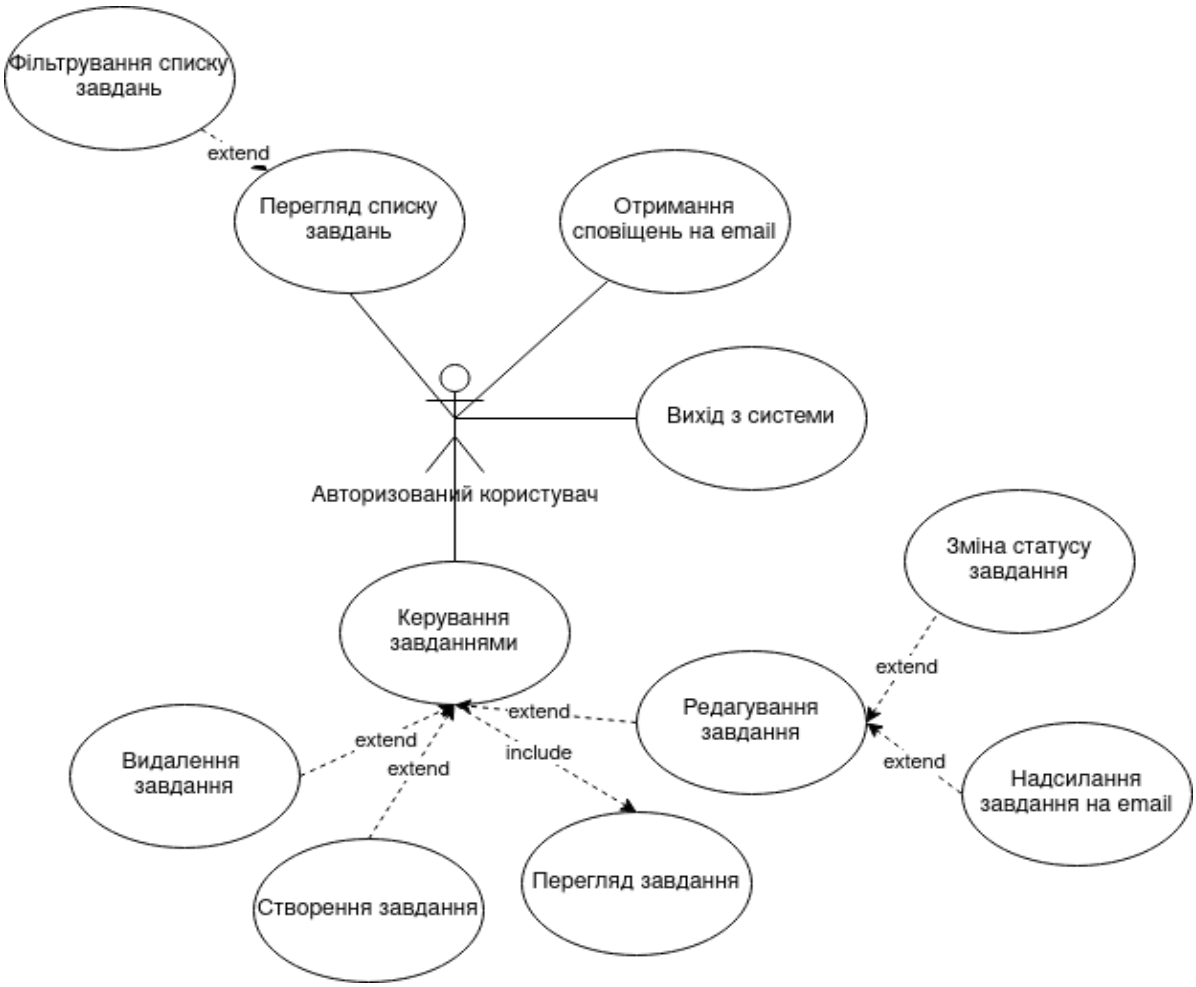

Рисунок 2.3 – ДВВ «Авторизований користувач»

Головна мета варіантів використання – змоделювати та задокументувати вимоги до поведінки та функціональності системи з точки зору акторів. Вони визначають конкретні сценарії використання різних функцій, послідовність кроків для досягнення певного результату. Описана в варіанті використання послідовність дій показує, як актор взаємодіє з системою для виконання завдання чи задоволення потреби.

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 сторінку входу до системи та обирає пункт «Зареєструватися» під написом «Не маєте облікового запису?». 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 сторінку входу до системи. Система відображає форму входу до системи. Після введення даних (електронна пошта та пароль), користувач натискає на кнопку «Вхід», система відображає особистий планувальник завдань 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» в правому верхньому кутку інтерфейса. Система видаляє дані про сесію користувача та переадресовує на сторінку входу.

Прецедент «Керування завданнями».

Призначення: даний варіант використання надає можливість користувачу керувати завданнями в планувальнику.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач авторизується в системі. Система відображає інтерфейс планувальника завдань.

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

Виняткова ситуація 1: сесія користувача вичерпана – система відображає сторінку входу до системи.

Прецедент «Створення завдання».

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

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на дату на котру потрібно запланувати завдання. Система відображає форму створення нового завдання. Користувач може ввести наступні дані: назву, опис, початкову та кінцеву дату завдання. Після введення даних, користувач натискає на кнопку «Додати». Система зберігає дані завдання в БД, ініціалізуючи деякі властивості (чи потребує сповіщення, статус) за замовчуванням, повідомляє користувача про успіх операції та відображає сторінку планувальника.

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

Виняткова ситуація 2: сесія користувача вичерпана – система відображає сторінку входу до системи.

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

Прецедент «Редагування завдання».

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

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на створене завдання. Система відображає форму редагування завдання. Користувач може змінити основні дані, а також змінити статус. Після редагування даних, користувач натискає на кнопку «Оновити». Система оновлює дані завдання в БД, повідомляє користувача про успіх операції та відображає сторінку планувальника.

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

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

Виняткова ситуація 2: сесія користувача вичерпана – система відображає сторінку входу до системи.

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

Прецедент «Видалення завдання».

Призначення: даний варіант використання надає можливість користувачу видалити завдання.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на створене завдання. Система відображає форму редагування завдання. Користувач натискає на кнопку «Видалити». Система видаляє дані завдання з БД, повідомляє користувача про успіх операції та відображає сторінку планувальника.

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

Виняткова ситуація 1: сесія користувача вичерпана – система відображає сторінку входу до системи.

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

Прецедент «Перегляд завдання».

Призначення: даний варіант використання надає можливість користувачу переглядати завдання.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач переходить на сторінку планувальника. Система відображає створені завдання, розділяючи їх за кольоровою палітрою згідно статусу.

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

Виняткова ситуація 1: сесія користувача вичерпана – система відображає сторінку входу до системи.

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

Прецедент «Зміна статусу завдання».

Призначення: даний варіант використання надає можливість користувачу змінювати статус завдання.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на створене завдання. Система відображає форму редагування завдання. Користувач може змінити змінити статус. Після зміни, користувач натискає на кнопку «Оновити». Система оновлює статус завдання в БД, повідомляє користувача про успіх операції та відображає сторінку планувальника, змінюючи колір завдання згідно статусу.

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

Виняткова ситуація 1: сесія користувача вичерпана – система відображає сторінку входу до системи.

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

Прецедент «Надсилання завдання на email».

Призначення: даний варіант використання надає можливість користувачу надсилати створені завдання на обраний email.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на створене завдання. Система відображає форму редагування завдання. Користувач натискає на кнопку «Поділитися». Система відображає форму, де користувач може ввести email. Після введення користувач натискає на кнопку «Поділитися». Система надсилає дані обраного завдання на вказаний email та відображає користувачу повідомлення про успіх операції.

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

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

Виняткова ситуація 2: сесія користувача вичерпана – система відображає сторінку входу до системи.

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

Прецедент «Отримання сповіщень на email».

Призначення: даний варіант використання надає можливість користувачу отримувати сповіщення про завдання, які скоро закінчаться, на вказаний при реєстрації email.

Основний потік подій: даний варіант використання починає виконуватися, коли добігає кінцева дата завдання. Система надсилає сповіщення, яке включає в собі дані такого завдання, на вказаний при реєстрації email, а також помічає завдання в БД, як таке, для якого надіслане сповіщення.

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

Виняткова ситуація 1: сесія користувача вичерпана – система відображає сторінку входу до системи.

Виняткова ситуація 2: помилка сервера – система не надсилає сповіщення, дані помилки можна подивитися в логах системи.

Прецедент «Перегляд списку завдань».

Призначення: даний варіант використання надає можливість користувачу переглянути список створених завдань.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на кнопку «Переглянути список завдань». Система відображає список створених завдань та фільтри.

Альтернативний потік подій: жодного завдання не було створено – система відображає фільтри і напис «Завдань немає».

Виняткова ситуація 1: сесія користувача вичерпана – система відображає сторінку входу до системи.

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

Прецедент «Фільтрування списку завдань».

Призначення: даний варіант використання надає можливість користувачу фільтрувати завдання в списку завдань.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вибирає необхідні фільтри і натискає на кнопку «Фільтрувати». Система відображає список завдань, згідно обраним фільтрам.

Передумова: перед початком виконання даного варіанта використання користувач повинен натиснути на кнопку «Переглянути список завдань».

Виняткова ситуація 1: сесія користувача вичерпана – система відображає сторінку входу до системи.

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

2.3 Діаграма діяльності

Діаграма діяльності (activity diagram) є одним із видів поведінкових діаграм в уніфікованій мові моделювання (UML), яка використовується для візуалізації та документування потоків управління в межах певної операції або процесу [17]. Ця діаграма дозволяє зобразити послідовність кроків, що виконуються в рамках певної діяльності, а також альтернативні шляхи та розгалуження, які можуть виникати в процесі її реалізації.

Діаграма діяльності є особливо корисною для моделювання бізнес-процесів, робочих потоків та алгоритмів, оскільки вона забезпечує наочне представлення їх структури та логіки [17]. Вона дозволяє візуалізувати точки прийняття рішень, паралельні потоки, цикли та винятки, що допомагає краще зрозуміти складну поведінку системи.

Крім того, діаграма діяльності може бути використана для моделювання взаємодії між окремими компонентами системи або між системою та її акторами [17]. Вона дозволяє зобразити обмін даними, синхронізацію та координацію між різними елементами, що є особливо корисним для аналізу та проєктування складних розподілених систем.

Діаграма діяльності також відіграє важливу роль у документуванні та комунікації вимог до системи. Вона забезпечує зрозумілу та стандартизовану форму представлення процесів та робочих потоків, що полегшує обговорення та узгодження їх з зацікавленими сторонами [17]. Крім того, діаграма може слугувати основою для подальшого проєктування та реалізації системи, забезпечуючи базу для визначення компонентів, їх взаємодій та поведінки.

Загалом, діаграма діяльності є потужним інструментом для візуалізації, аналізу та документування складних процесів, робочих потоків та алгоритмів у рамках розробки програмних систем. Вона забезпечує зрозумілу та стандартизовану форму представлення поведінки системи, сприяючи ефективній комунікації та координації зусиль між різними зацікавленими сторонами на різних етапах розробки.

На рисунках 2.4 – 2.6 наведено діаграму діяльності прецеденту «Створити завдання».

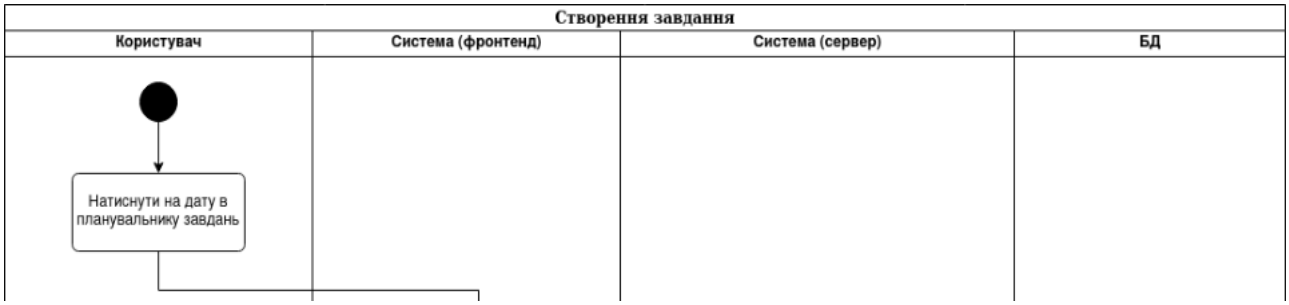

Рисунок 2.4 – Діаграма діяльності (1 частина)

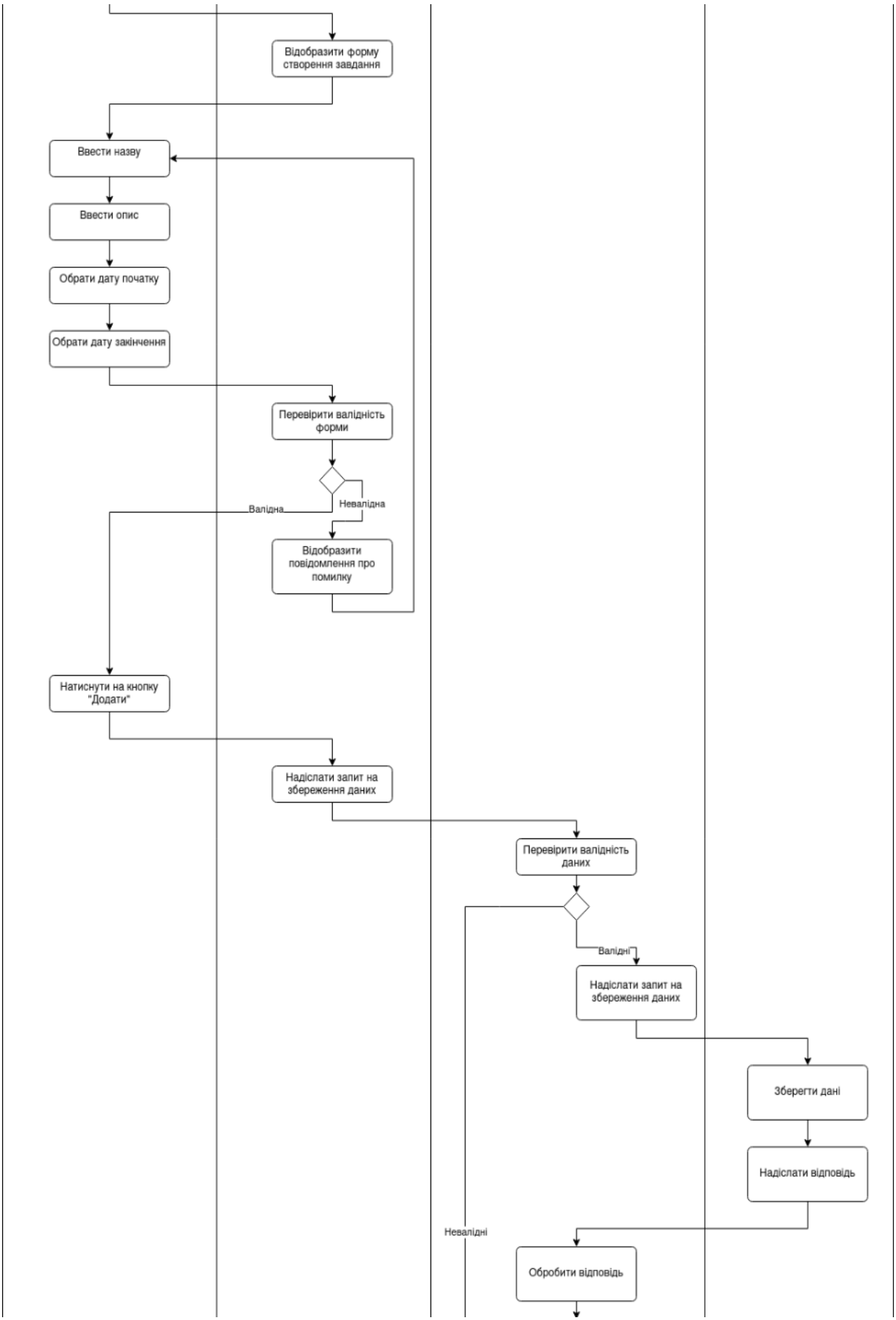

Рисунок 2.5 – Діаграма діяльності (2 частина)

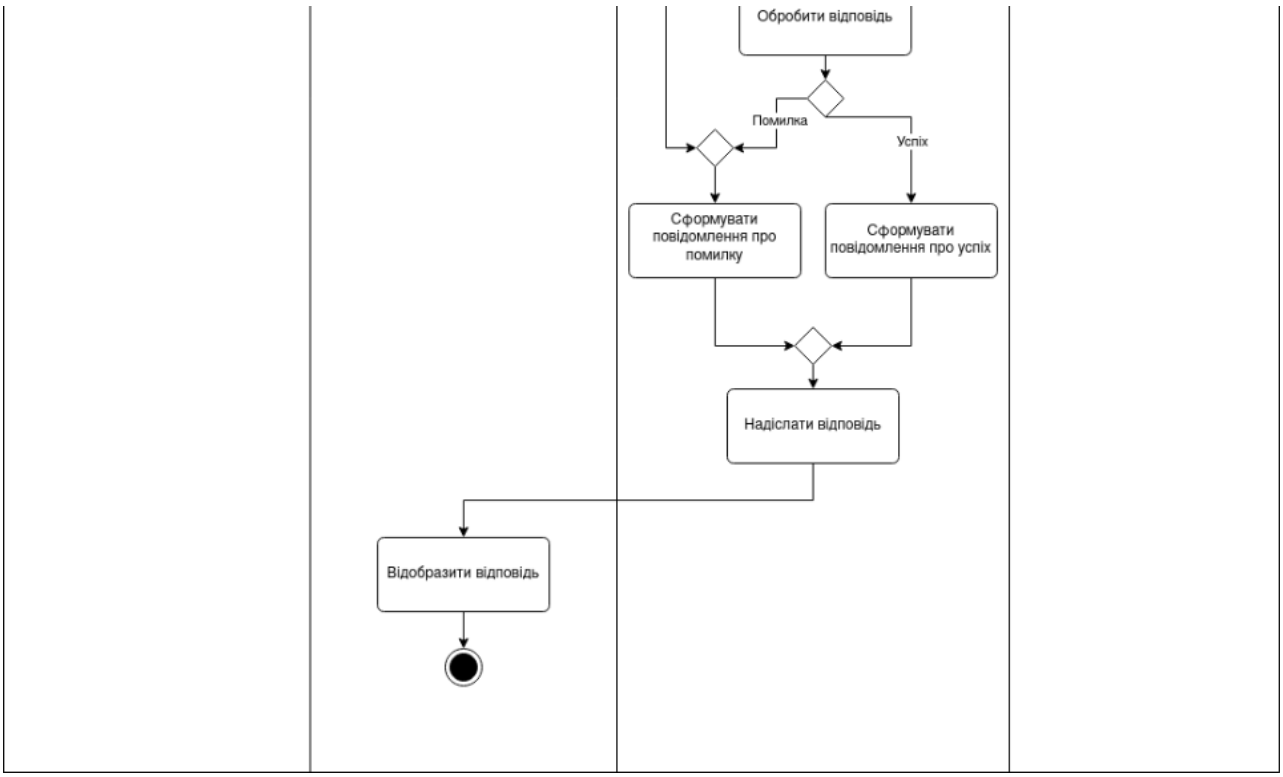

Рисунок 2.6 – Діаграма діяльності (3 частина)

2.4 Діаграма послідовності

Діаграма послідовності (sequence diagram) є одним із видів діаграм взаємодії в уніфікованій мові моделювання (UML), яка використовується для візуалізації та документування послідовності взаємодій між об’єктами системи в межах певного сценарію використання або операції [17]. Ця діаграма дозволяє зобразити порядок обміну повідомленнями між об’єктами з плином часу, що допомагає зрозуміти динамічну поведінку системи.

Діаграма послідовності є особливо корисною для моделювання динамічних аспектів системи, а саме: взаємодії між її компонентами, потоку управління та обміну даними [17]. Вона дозволяє візуалізувати, як різні об’єкти взаємодіють один з одним для виконання певної функціональності, що допомагає краще зрозуміти логіку роботи системи.

Крім того, діаграма послідовності може бути використана для моделювання різних сценаріїв використання системи, включаючи як типові, так

і виняткові ситуації [17]. Це дозволяє проаналізувати та документувати різні можливі шляхи взаємодії між об’єктами, що є особливо важливим для забезпечення повноти та коректності розробки системи.

На рисунку 2.7 описана діаграма послідовності прецеденту «Створити завдання».

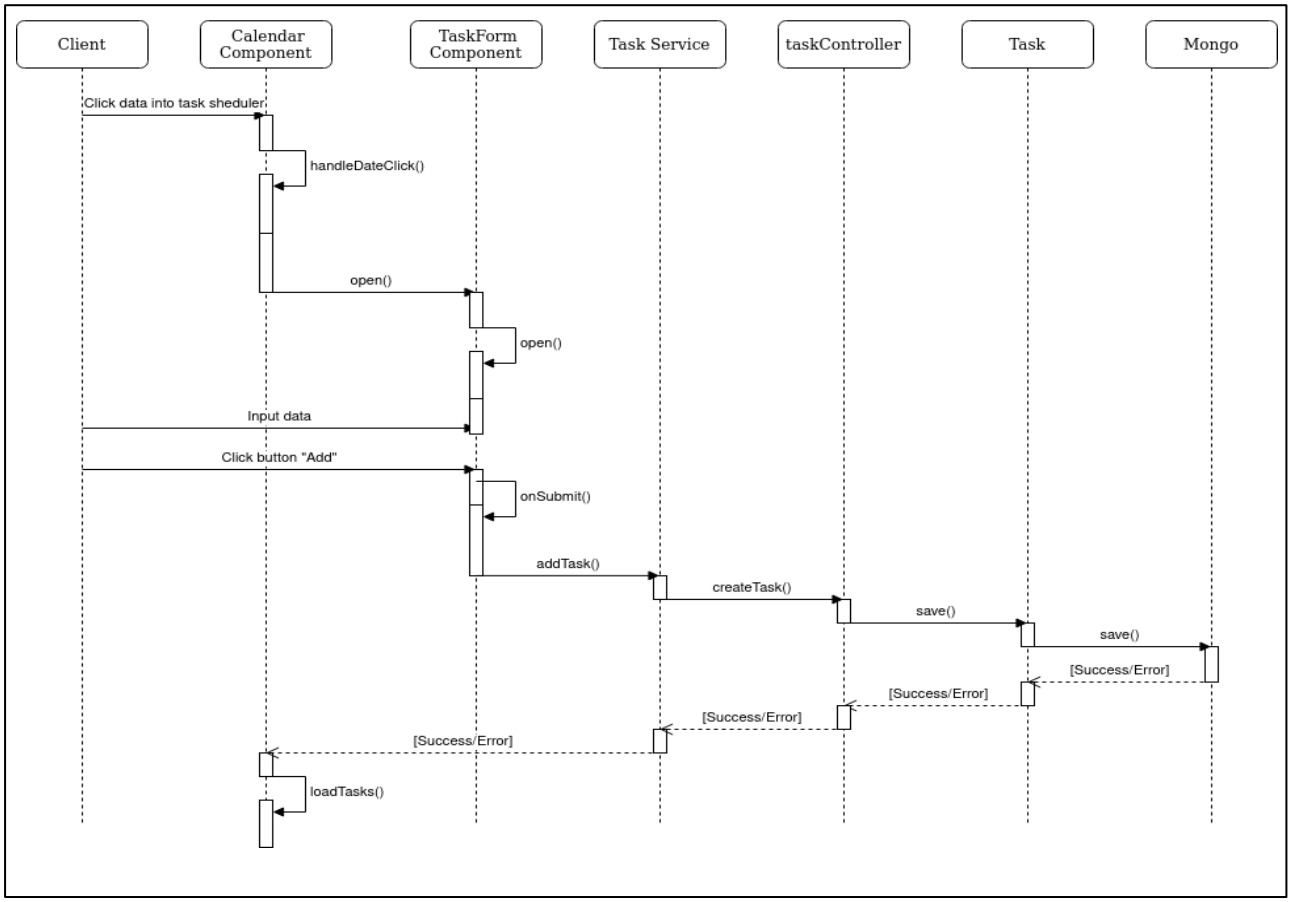

Рисунок 2.7 – ДП «Створити завдання»

Загалом, діаграма послідовності є потужним інструментом для візуалізації, аналізу та документування динамічної поведінки системи, а також взаємодії між її компонентами. Вона забезпечує зрозумілу та стандартизовану форму представлення сценаріїв використання системи, сприяючи ефективній комунікації та координації зусиль між різними зацікавленими сторонами на різних етапах розробки.

2.5 Діаграма розгортання

Діаграма розгортання (deployment diagram) є одним із видів структурних діаграм в уніфікованій мові моделювання (UML), яка використовується для візуалізації та документування фізичної архітектури системи, включаючи її компоненти та артефакти, а також розміщення їх на різних вузлах та обчислювальних ресурсах [17].

Діаграма розгортання дозволяє зобразити топологію системи, показуючи, як різні програмні компоненти розподіляються та розгортаються на фізичних пристроях, таких як сервери, клієнтські машини, мобільні пристрої або вбудовані системи [17]. Це допомагає зрозуміти фізичну структуру системи та взаємозв’язки між її компонентами на різних рівнях розгортання.

Діаграма розгортання відіграє важливу роль у документуванні та комунікації архітектури системи. Вона забезпечує зрозумілу та стандартизовану форму представлення фізичної структури системи, що полегшує обговорення та узгодження її з зацікавленими сторонами, такими як адміністратори системи, оператори центрів обробки даних та команди підтримки [17].

На рисунку 2.8 наведено діаграму розгортання системи.

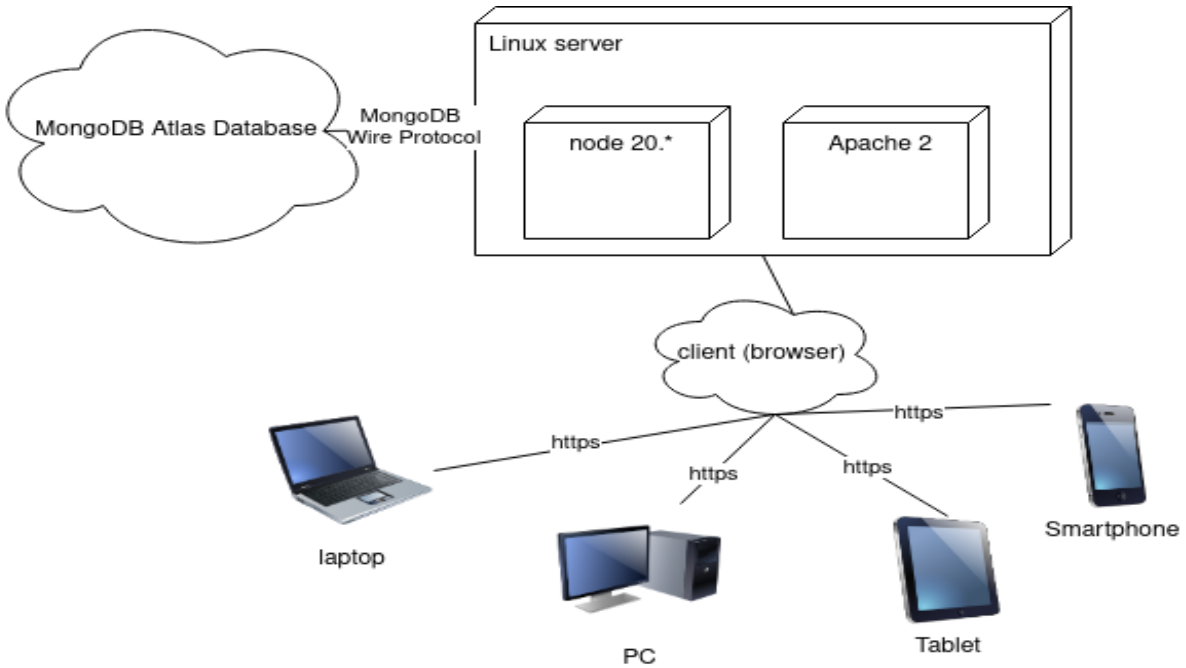

Рисунок 2.8 – Діаграма розгортання

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Для реалізації було використано вебфреймворк Angular 16 та середовище Node.js.

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

FullCalendar – це бібліотека JavaScript для створення інтерактивних та налаштовуваних календарів [6].

MongoDB Atlas Database – хмарний сервіс, який дозволяє розгортати, керувати та масштабувати розподілені сховища даних MongoDB [10].

3.2 Основні класи системи

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

Головними класами системи зі сторони фронтенду є:

– CalendarComponent – відповідає за відображення інтерфейсу планувальника та обробку дій, які пов’язані з датами або івентами;
– TaskFormComponent – відповідає за дії з завданнями (відображення форм та функціональних кнопок);
– TaskService – відповідає за надсилання HTTP запитів на сервер, що стосуються колекції task.

Головними класами системи зі сторони серверу є:

– controllers/taskControllers – відповідає за CRUD операції щодо колекції tasks;

– models/Task – відповідає за взаємодію з колекцією tasks [11].

Нижче приведено код моделі Task (рис. 3.1) [8, 10, 11, 14]. Детально ознайомитися з кодом серверної частини можна в додатку А.

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

const taskSchema : Schema<any, Model<...>, {...}, {...}, {...}, DefaultSchemaOptions,   = new mongoose.Schema( definition: {
  title: { type: String, required: true },
  description: { type: String },
  startTime: { type: Date, required: true },
  endTime: { type: Date, required: true },
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  userEmail: { type: String },
  notified: { type: Boolean, default: false },
  status: {
    type: String,
    required: true,
    enum: ['new', 'in_progress', 'completed', 'overdue'],
    default: 'new'
  }
});

module.exports = mongoose.model( name: 'Task', taskSchema);
```

Рисунок 3.1 – Модель Task

3.3 Налаштування MongoDB Atlas Database

Система використовує MongoDB Atlas Database для збереження інформації про користувачів, категорії та сповіщення. Це рішення покращує продуктивність системи і гарантує безпеку даних.

Щоб створити базу даних на MongoDB Atlas Database, потрібно або увійти до особистого облікового запису на cloud.mongodb.com, або зареєструвати новий обліковий запис, якщо він відсутній. Після входу, необхідно натиснути на кнопку «Create cluster» (рис. 3.2) [10].

Рисунок 3.2 – Кнопка «Create cluster»

Наступним кроком є вибір параметрів кластера, від яких залежатимуть його ціна та продуктивність. Сервіс пропонує використовувати заздалегідь підготовлені шаблони налаштувань для зручності вибору оптимальної конфігурації. На цьому етапі необхідно вказати назву кластера, вибрати провайдера та регіон його розміщення (рис. 3.3) [10].

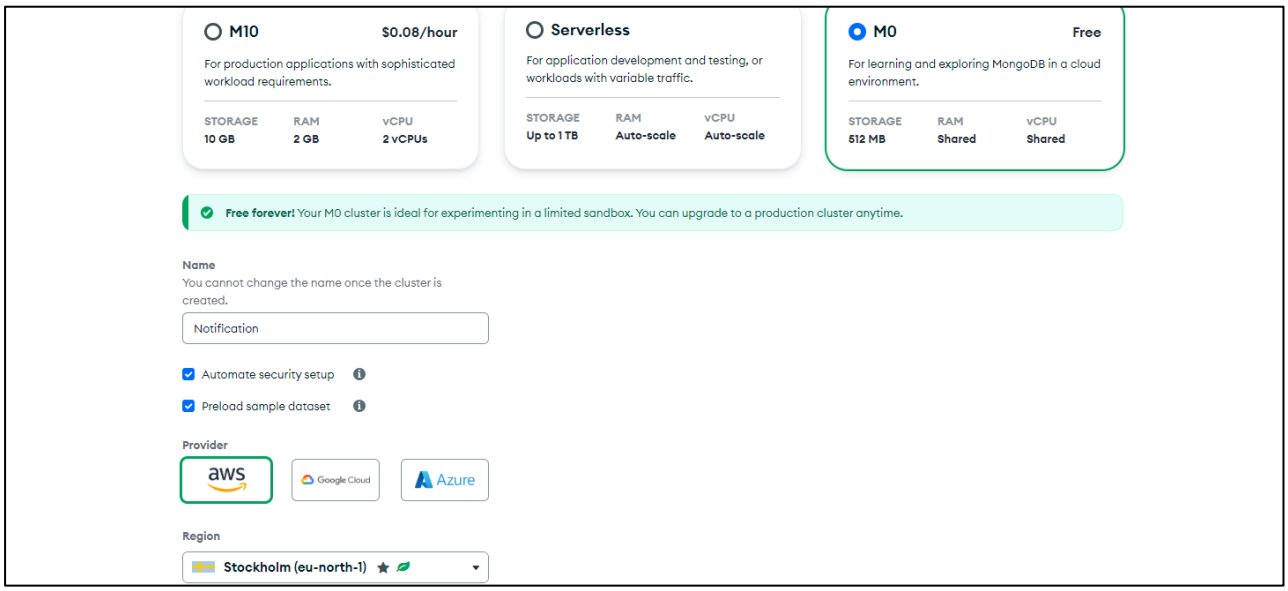

Рисунок 3.3 – Налаштування кластера

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

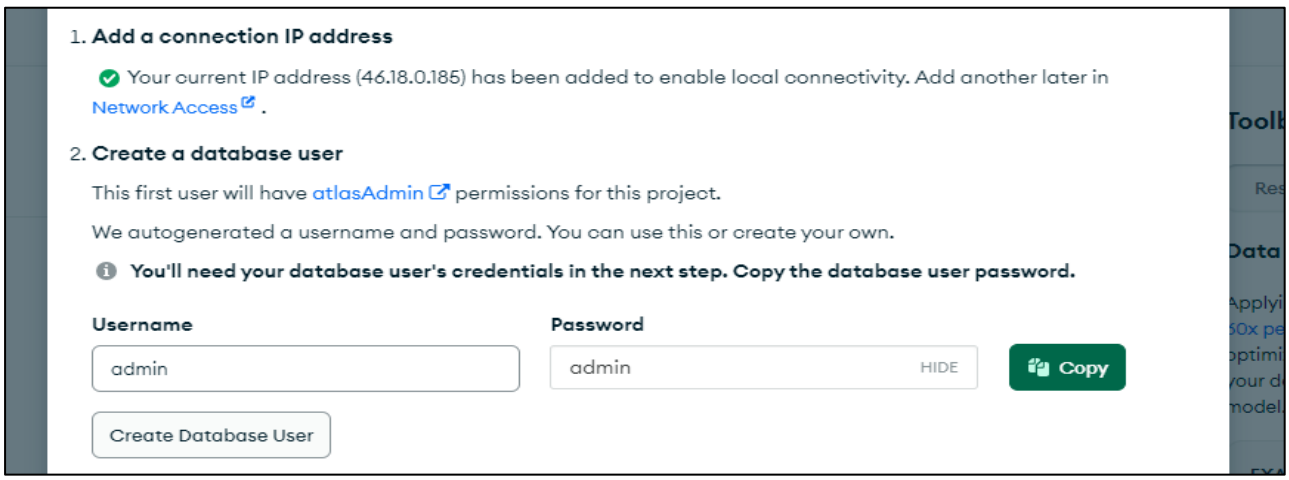

Рисунок 3.4 – Створення користувача БД

Для переходу до наступного кроку необхідно натиснути на кнопку «Choose a connection method». Сервіс пропонує вибрати спосіб підключення, в нашому випадку це MongoDB Compass. В інтерфейсі обраного способу можна налаштувати параметри під свої потреби та отримати дані для підключення до нашої бази даних. Ці дані можна використовувати як в Compass, так і в Node.js (рис. 3.5) [8, 10, 11, 14].

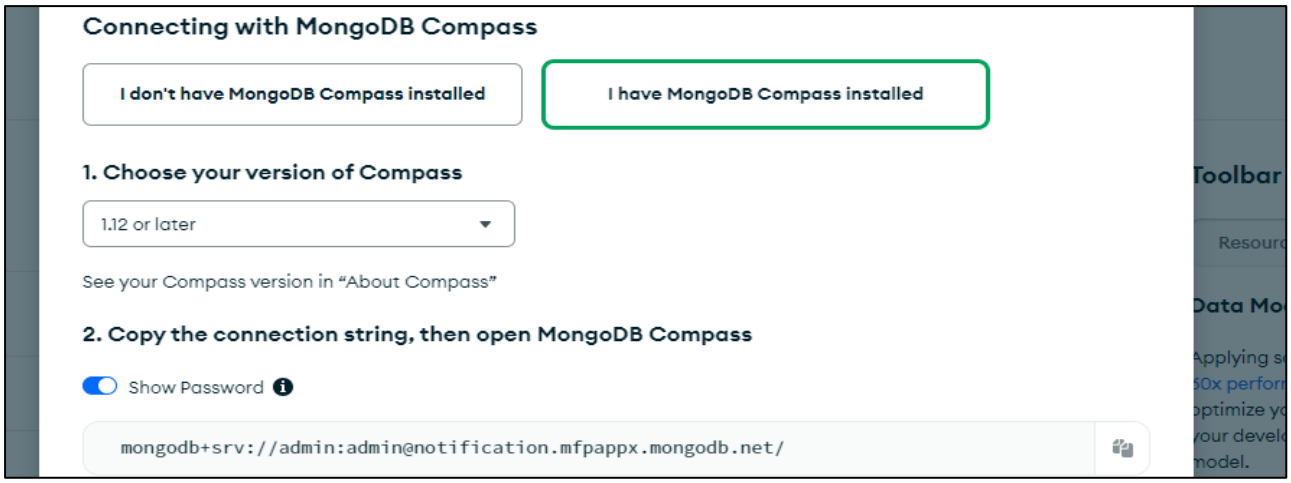

Рисунок 3.5 – Отримання даних для з'єднання з БД

Після створення кластеру, користувач може підключитися до нього в MongoDB Compass (рис. 3.6) [10].

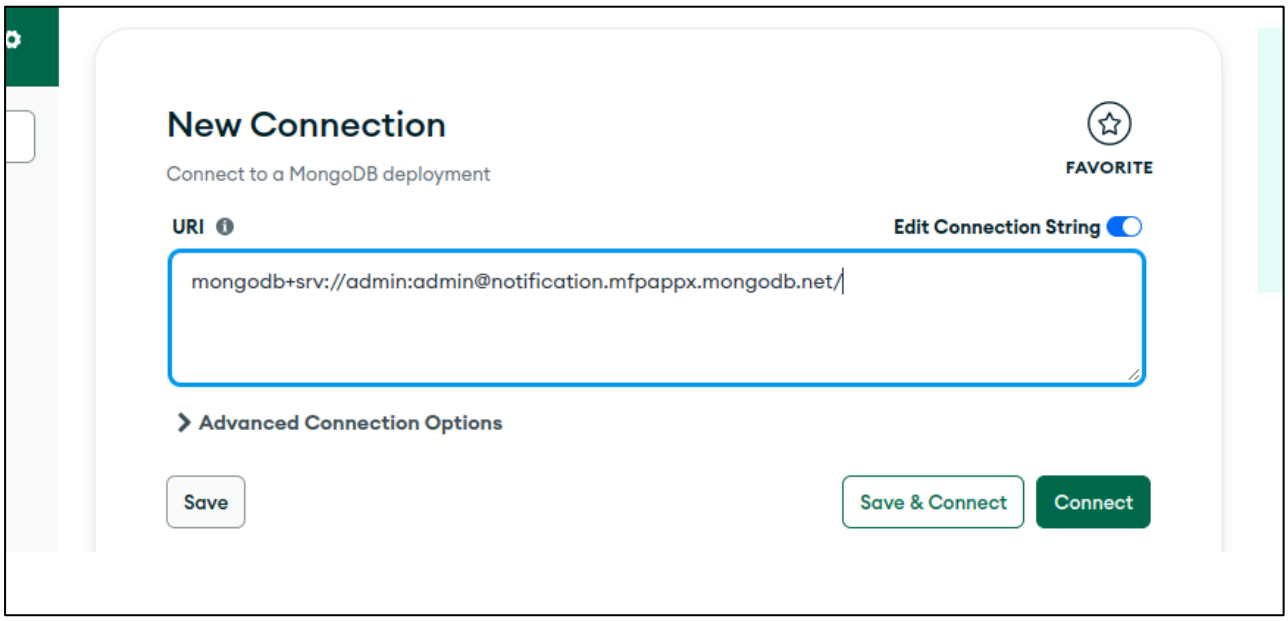

Рисунок 3.6 – Перегляд кластеру

3.4 Тестування проєкту

Unit тестування відіграє ключову роль у розробці додатків на Angular фреймворку. Цей процес забезпечує якість коду, полегшує рефакторинг і зменшує час на налагодження. Angular надає вбудовані інструменти та бібліотеки, такі як Jasmine та Angular TestBed, для написання та виконання unit тестів [1, 12, 13].

Процес написання unit тестів в Angular зазвичай розпочинається з налаштування тестового середовища. Для цього створюється окремий файл специфікації тестів, де імпортуються необхідні модулі та класи.

Наступним кроком є налаштування TestBed, що дозволяє імпортувати необхідні модулі, компоненти, провайдери та залежності для тестування компоненту.

Після налаштування тестового середовища створюється екземпляр ComponentFixture, який надає методи та властивості для взаємодії з компонентом під час тестування. Тестові специфікації визначаються за допомогою функцій describe та it з бібліотеки Jasmine [12]. Всередині тестових випадків використовуються очікування (expectations) для перевірки поведінки компоненту.

Виконання unit тестів у Angular проєкті здійснюється за допомогою команди ng test або npm run test. Результати тестів відображаються у терміналі або у спеціальному вікні браузера. Angular CLI також надає інструменти для генерації тестових файлів під час створення нових компонентів [3, 13].

Unit тестування є важливою практикою для забезпечення якості коду та полегшення процесу рефакторингу та налагодження в Angular додатках. Воно допомагає виявляти та усувати помилки на ранніх стадіях розробки, сприяючи створенню стабільного та надійного програмного забезпечення.

На рисунку 3.7 наведено приклад такого тестування для компонента TaskForm.

```
it( expectation: 'should create', assertion: () => {
  expect(component).toBeTruthy();
});

it( expectation: 'should initialize the form on ngOnInit with empty task', assertion: () => {
  component.ngOnInit();
  expect(component.taskForm).toBeDefined();
  expect(component.taskForm.get('title')?.value).toEqual( expected: '');
});

describe( description: 'onSubmit', specDefinitions: () => {
  it( expectation: 'should call updateTask if task exists', assertion: () => {
    const task: Task = { _id: '123', title: 'Test Task', description: 'Do something', startTime: new Date()
    component.task = task;
    component.ngOnInit();
    taskService.updateTask.and.returnValue(of(task));
    component.onSubmit();
    expect(taskService.updateTask).toHaveBeenCalled();
  });
```

Рисунок 3.7 – Unit-тестування TaskFormComponent

Для запуску тестування в Angular додатку використовується команда «ng test»у терміналі [1, 5]. Після запуску відкривається вікно браузера, в якому відображаються результати виконання тестів. Аналізуючи отримані результати, розробники можуть визначити наявні недоліки чи проблемні місця в системі та вжити необхідних заходів для їх усунення (рис. 3.8). Детально ознайомитися з повним кодом теста можна в додатку Б.

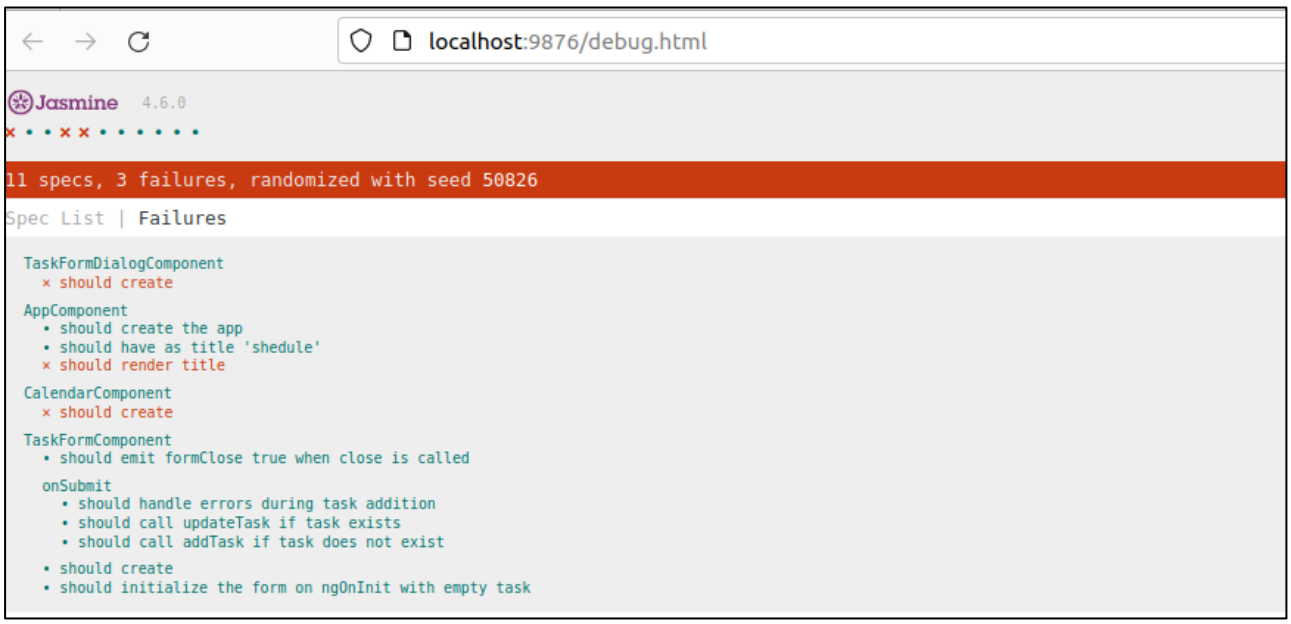

Рисунок 3.8 – Результат тестування TaskFormComponent

3.5 Керівництво користувача

3.5.1 Рівень підготовки користувача

Для повноцінного використання даного сервісу користувачеві необхідні базові навички роботи з комп’ютером та веббраузерами. Ретельне вивчення інструкцій користувача дозволяє максимально розкрити весь потенціал сервісу та ознайомитися з його функціональними можливостями. Детальне ознайомлення з керівництвом користувача забезпечує глибше розуміння принципів роботи сервісу, що значно полегшить опанування його функцій та зручне користування.

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 посилання «Зареєструватися», що знаходиться біля напису «Не маєте облікового запису?» (рис. 3.9).

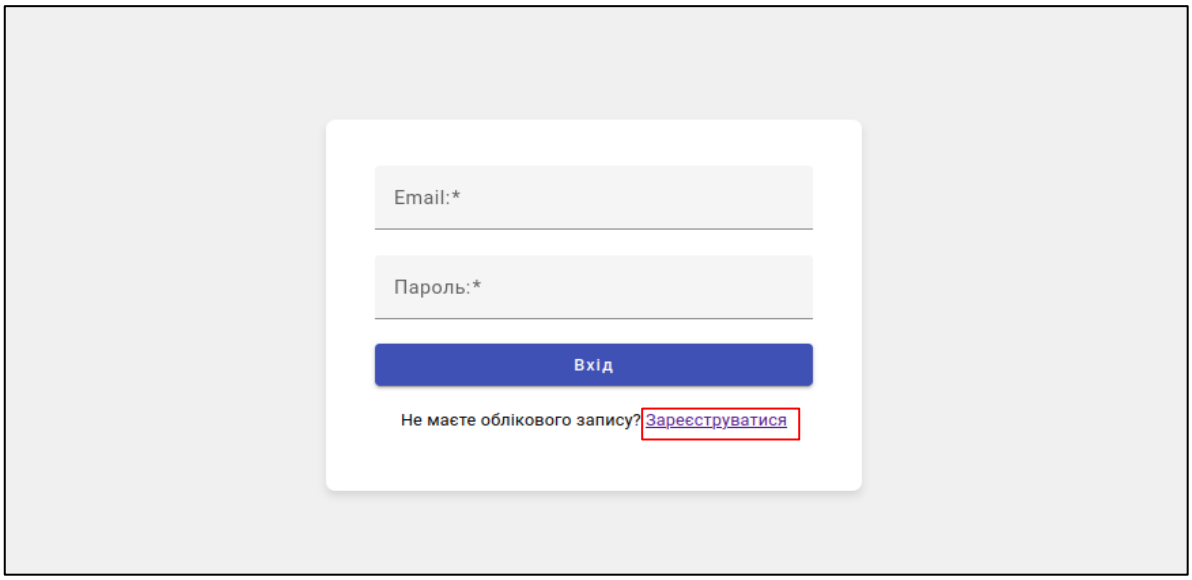

Рисунок 3.9 – Посилання «Зареєструватися»

Користувачеві відображається форма реєстрації, в якій необхідно ввести свою електронну адресу та пароль. Після введення даних користувач має натиснути кнопку Реєстрація» (рис. 3.10). Якщо в системі вже зареєстрований користувач з таким електронним поштовим акаунтом, на екрані з’явиться повідомлення про помилку з текстом «Користувач вже існує» (рис. 3.11). У такому випадку користувач має можливість перейти до форми входу до системи, натиснувши кнопку «Вхід».

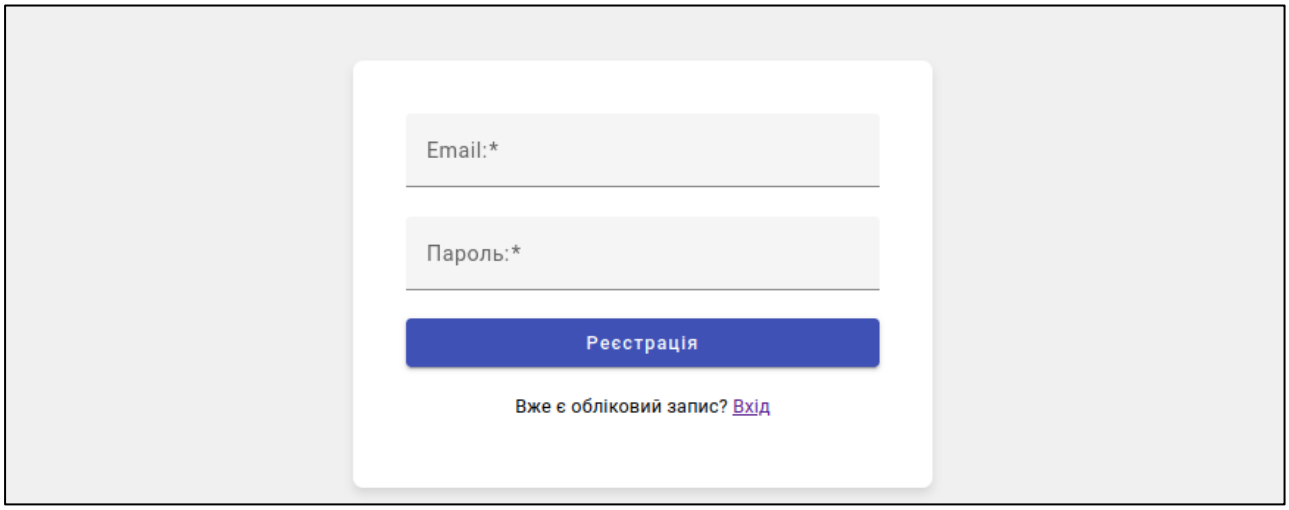

Рисунок 3.10 – Форма реєстрації

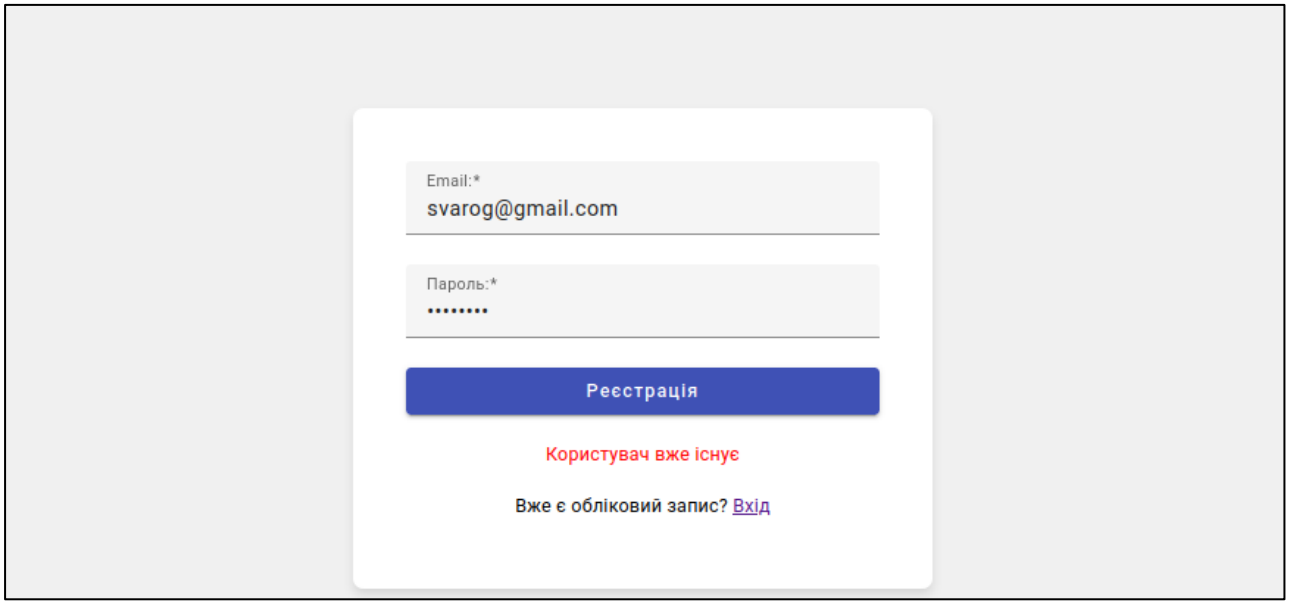

Рисунок 3.11 – Помилка при реєстрації

Після реєстрації, система відображає форму входу до системи.

3.5.3 Вхід до системи

Після успішної реєстрації або у випадку, коли користувач вже має обліковий запис, йому необхідно перейти на сторінку входу до системи. Це можна зробити, відвідавши адресу вебсервісу. Система відображає форму для входу до облікового запису (рис. 3.12).

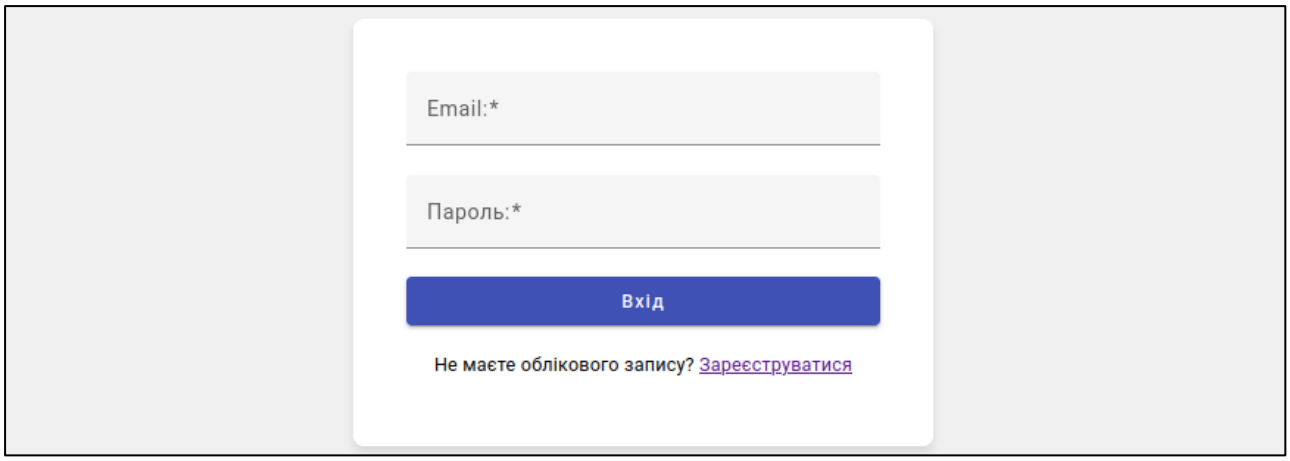

Рисунок 3.12 – Форма входу

Користувачеві потрібно ввести свої облікові дані (електронну адресу та пароль) і натиснути кнопку «Вхід». Після успішної аутентифікації система надасть доступ до планувальника завдань користувача (рис. 3.13).

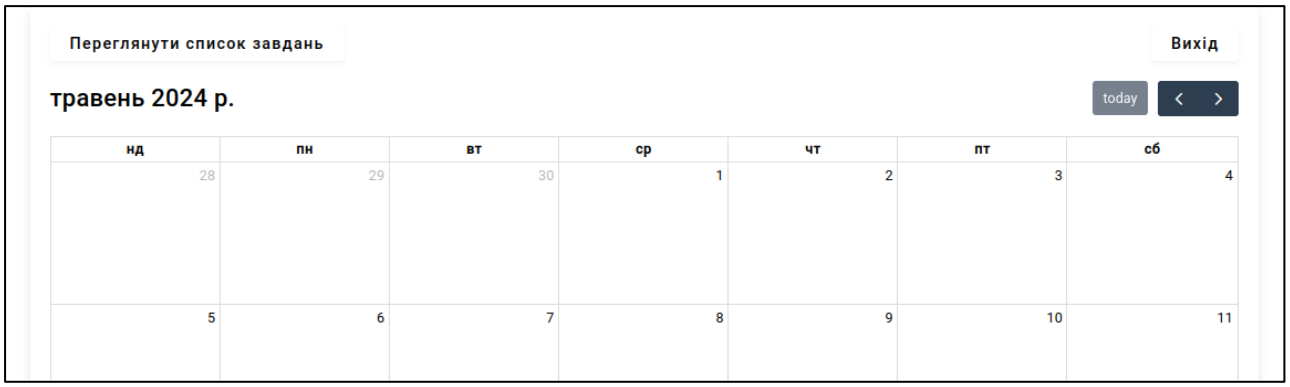

Рисунок 3.13 – Особистий кабінет

Якщо введені дані не відповідають жодному існуючому облікового запису або містять помилки, система повідомить про це користувача.

Для кожного типу помилки відображатиметься відповідне повідомлення з поясненням (рис. 3.14 – 3.15).

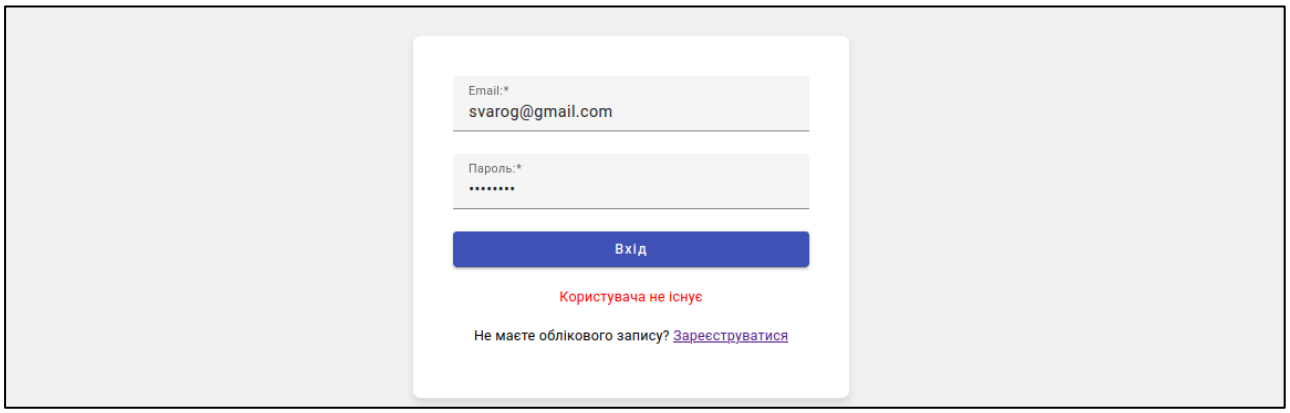

Рисунок 3.14 – Користувача не існує

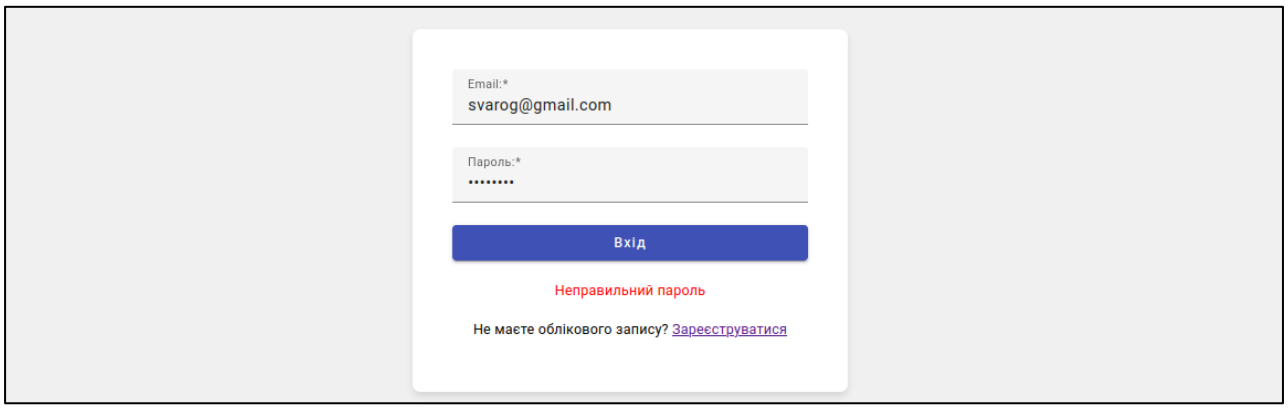

Рисунок 3.15 – Неправильний пароль

Навіть у випадку введення коректних облікових даних може виникнути помилка через проблеми з сервером або тимчасову недоступність системи. У такій ситуації користувачеві також буде показано відповідне повідомлення про помилку.

3.5.4 Створення завдання

Для створення завдання необхідно натиснути на комірку дати в планувальнику завдань.

Система відображає форму створення нового завдання для обраної дати (рис. 3.16).

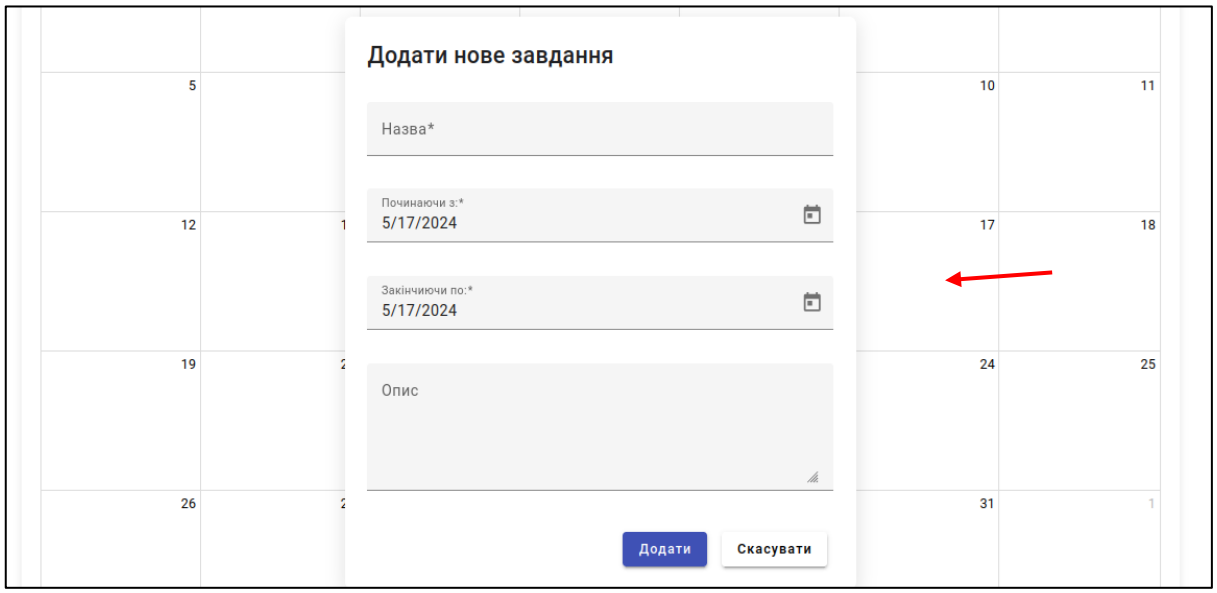

Рисунок 3.16 – Форма створення завдання

Наступним кроком потрібно заповнити поля форми та натиснути на кнопку «Додати» (рис. 3.17).

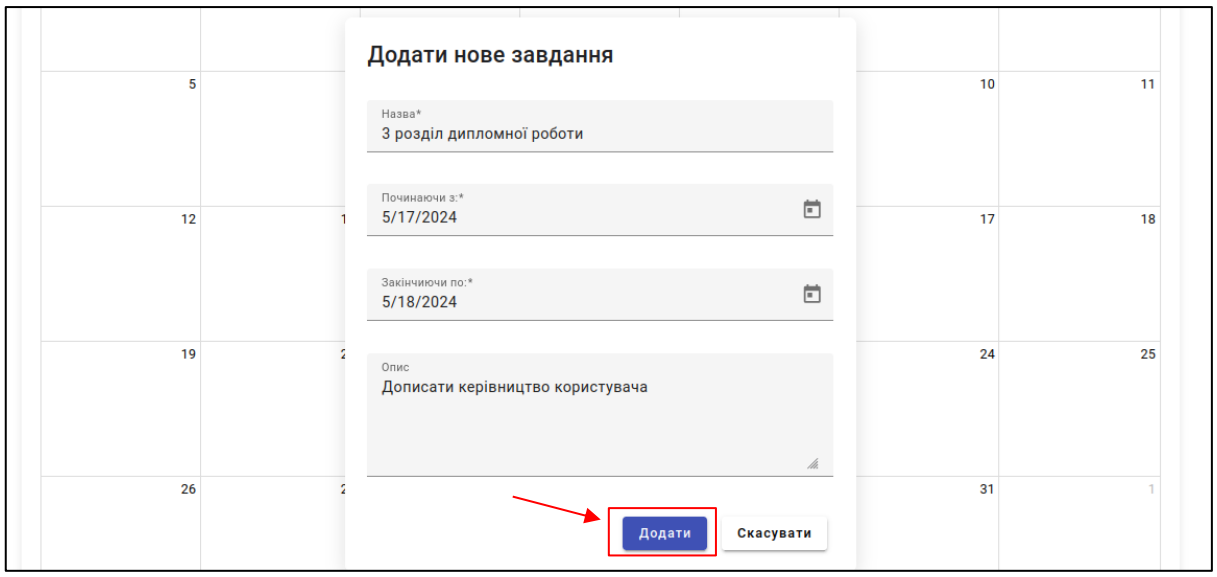

Рисунок 3.17 – Додати нове завдання

Після збереження даних про нове завдання, система відобразить його в планувальнику (рис. 3.18).

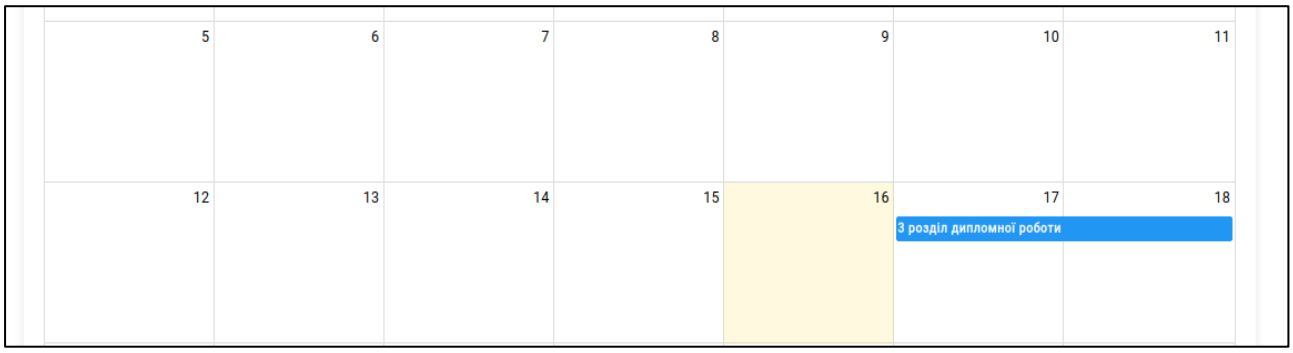

Рисунок 3.18 – Створене завдання

Колір завдання залежить від його статусу. Синім кольором позначаються нові завдання, помаранчевим – в процесі, зеленим – завершені, червоним – протерміновані.

3.5.5 Редагування завдання

Для редагування даних завдання необхідно натиснути на нього в планувальнику завдань. Система відображає форму, в яку підтягуються дані обраного завдання (рис. 3.19).

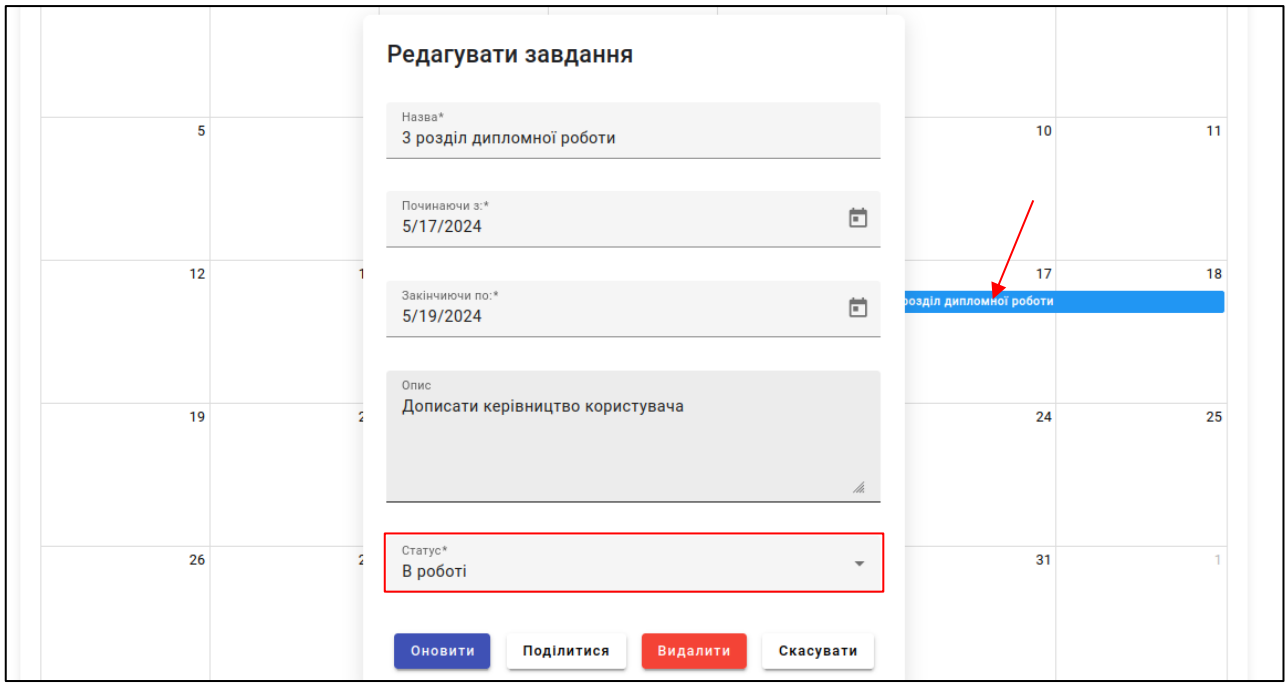

Рисунок 3.19 – Форма редагування завдання

Наступним кроком, після редагування даних, потрібно натиснути на кнопку «Оновити» (рис. 3. 20).

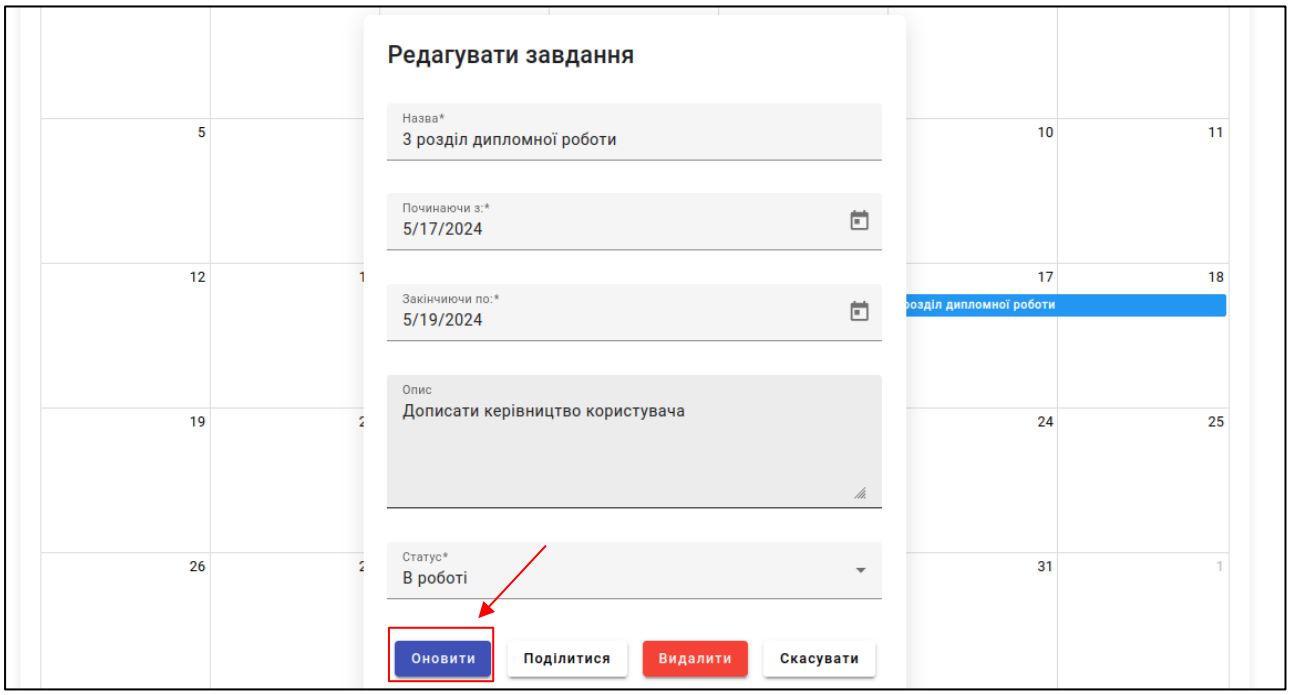

Рисунок 3.20 – Кнопка «Оновити»

Система відображає повідомлення про успішну операцію та оновлює завдання в планувальнику завдань (рис. 3.21).

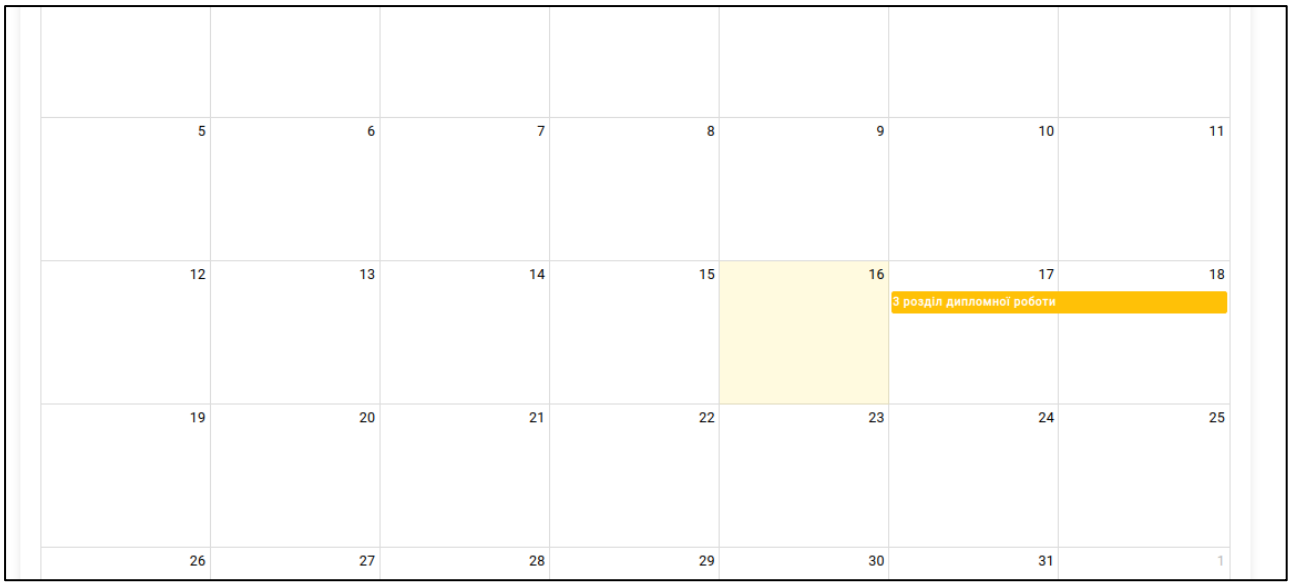

Рисунок 3.21 – Оновлене завдання

3.5.6 Видалення завдання

Для видалення завдання, необхідно натиснути на нього в планувальнику завдань. Система відображає форму редагування завдання (рис. 3.22).

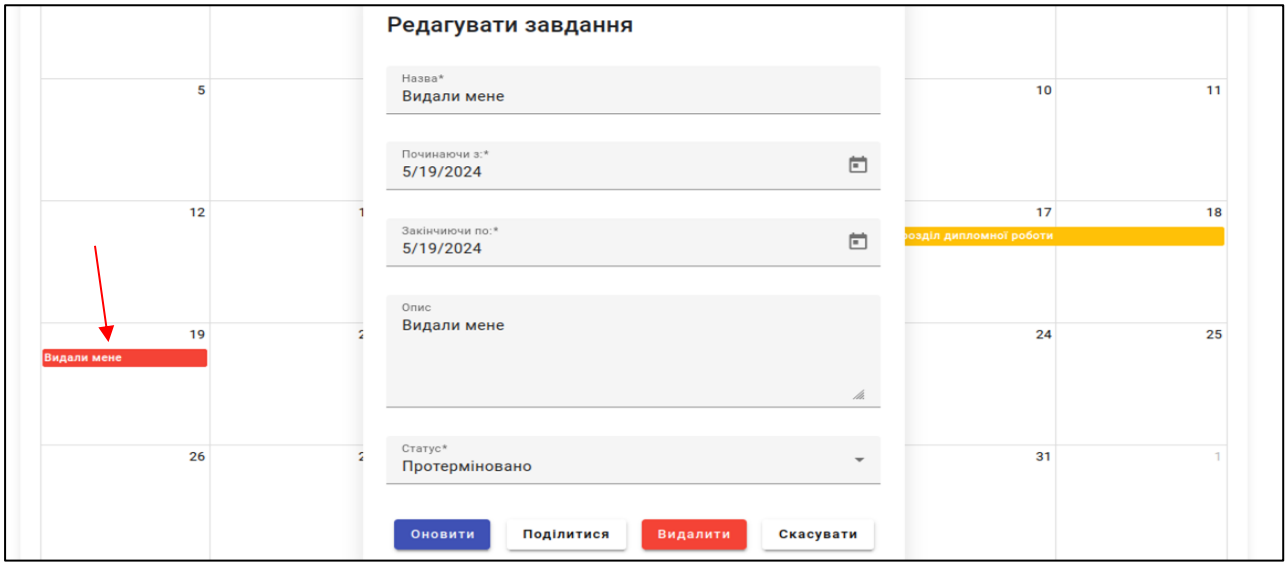

Рисунок 3.22 – Завдання на видалення

Наступним кроком необхідно натиснути на кнопку «Видалити» (рис. 3.23).

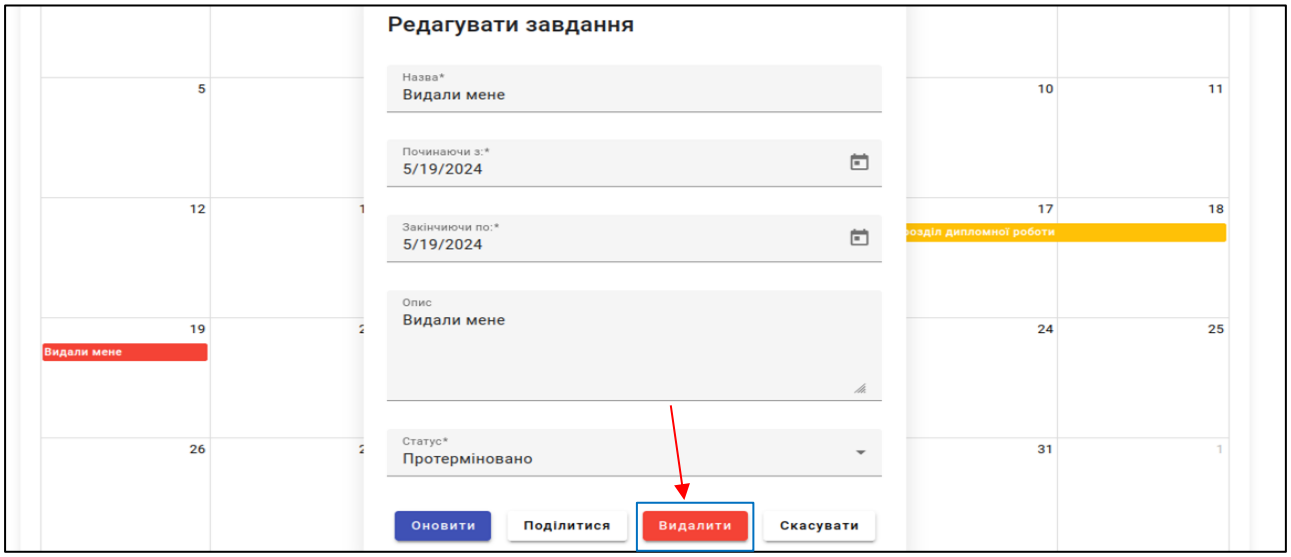

Рисунок 3.23 – Кнопка «Видалити»

Система видаляє дані завдання, відображає повідомлення про успішну операцію та відображає планувальник завдань (рис. 3.24).

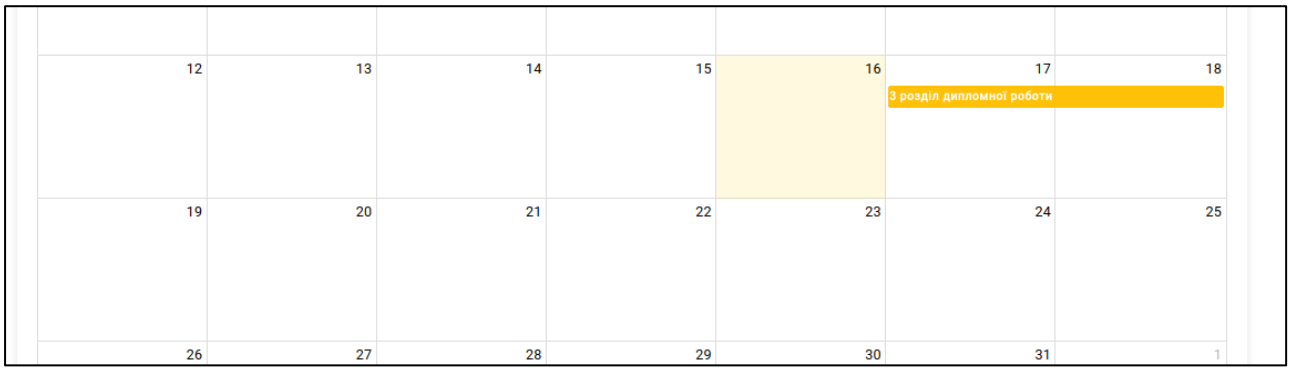

Рисунок 3.24 – Планувальник після видалення завдання

3.5.7 Поділитися завданням

Для того щоб поділитися заданням, необхідно натиснути на потрібне завдання в планувальнику. Система відображає форму редагування (рис. 3.25).

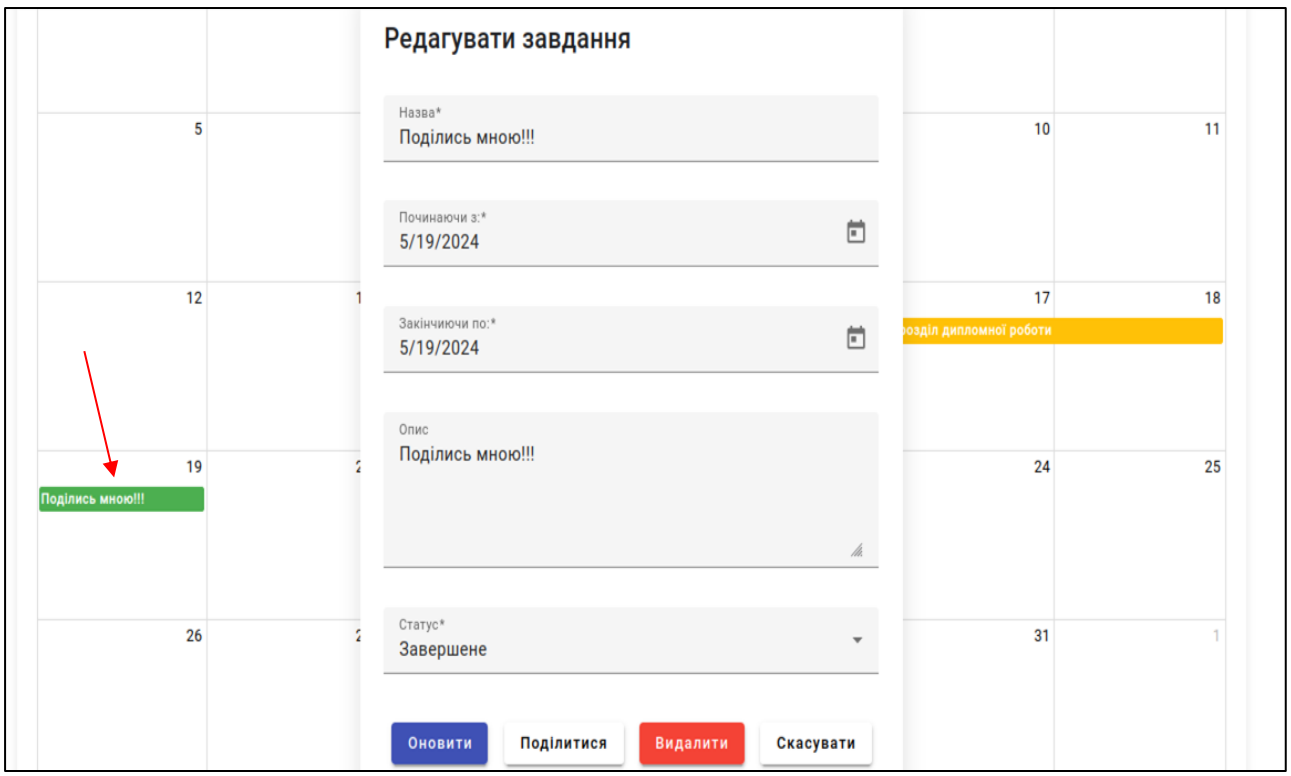

Рисунок 3.25 – Вибір завдання

Наступним кроком необхідно натиснути на кнопку «Поділитися». Система відображає форму, де можна ввести email отримувача (рис. 3.26).

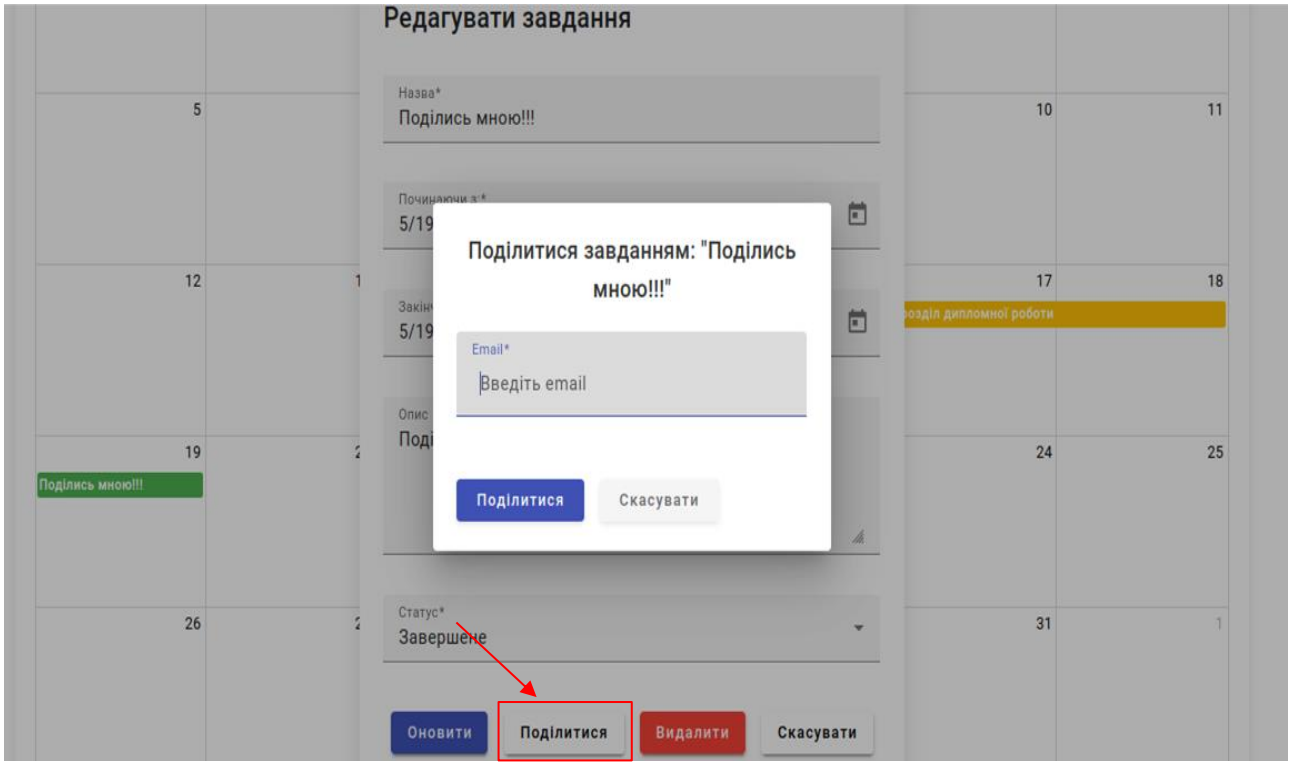

Рисунок 3.26 – Форма вводу email

Після введення email отримувача, необхідно натиснути на кнопку «Поділитися» (рис. 3.27).

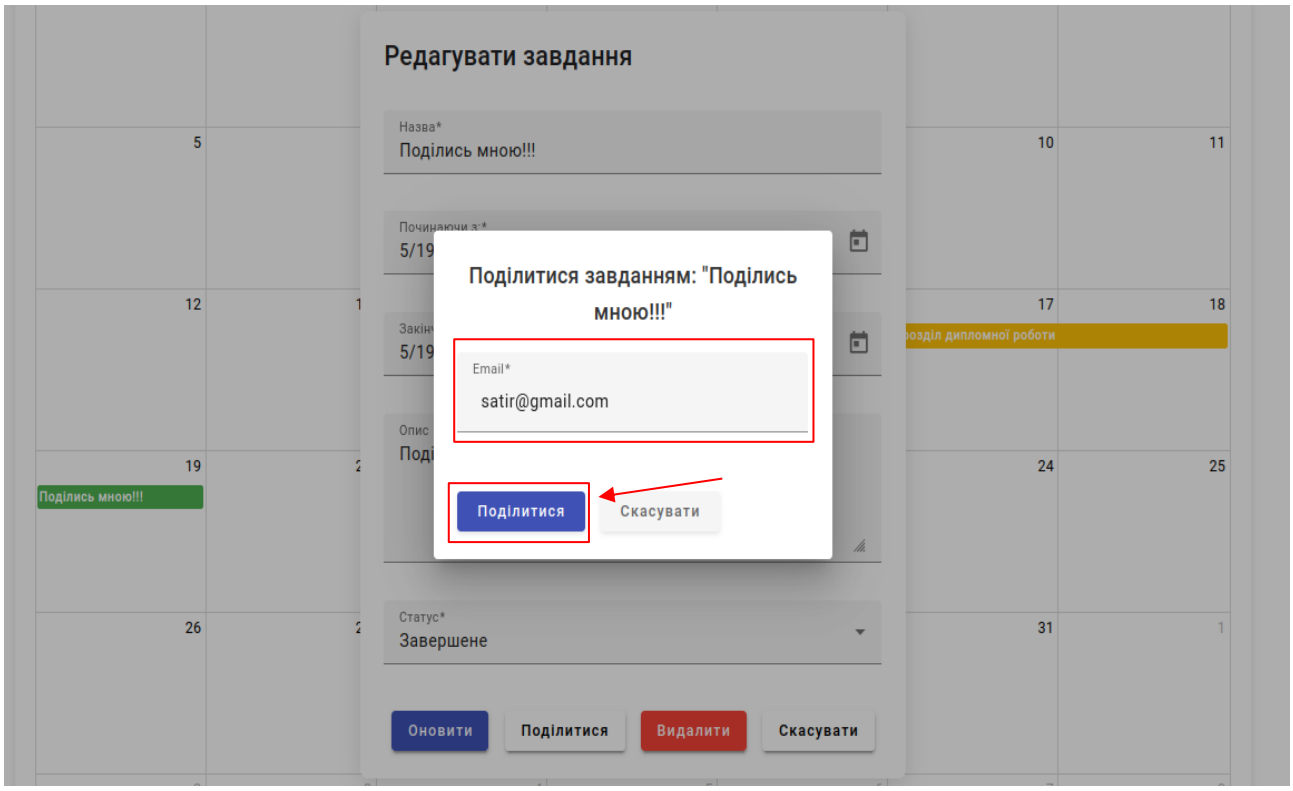

Рисунок 3.27 – Поділитися завданням

Система надсилає завдання та відображає повідомлення про успішну операцію (рис. 3.28).

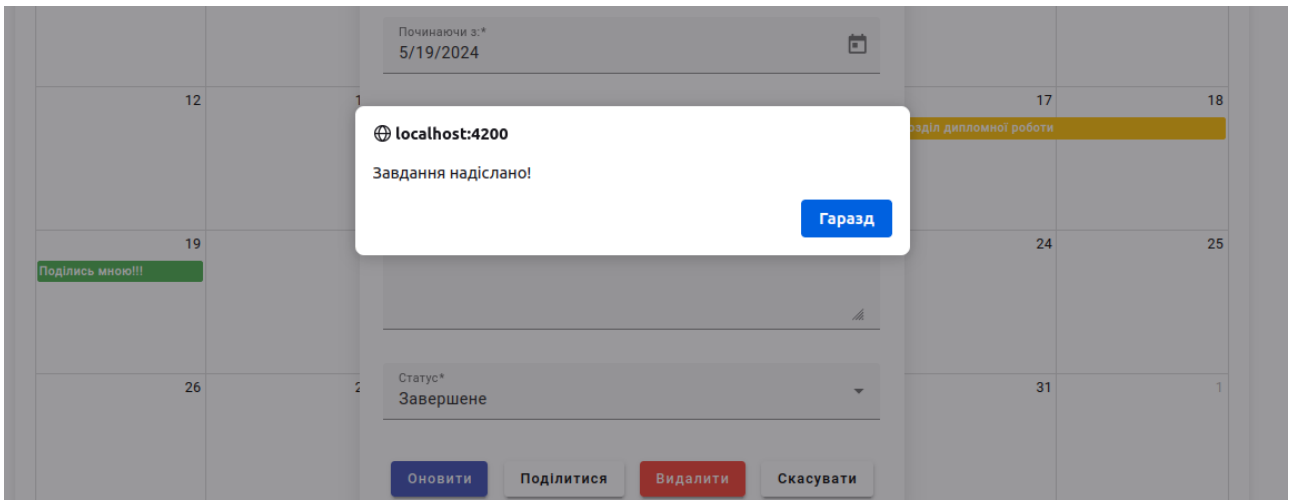

Рисунок 3.28 – Повідомлення про успіх

На пошту адресата прийде повідомлення, в якому буде зазначено деталі завдання (рис. 3.29).

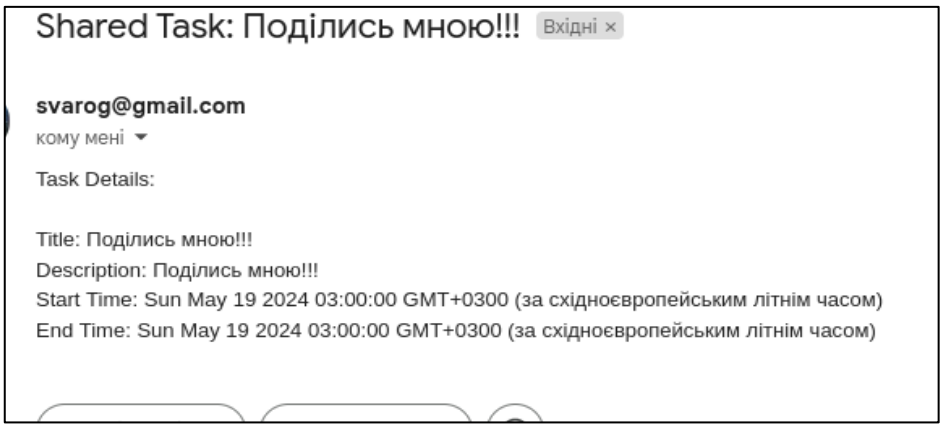

Рисунок 3.29 – Повідомлення на пошті

3.5.8 Переглянути список завдань

Для того щоб переглянути список завдань, необхідно натиснути на кнопку «Переглянути список завдань». Система відображає список створених завдань користувача з можливістю фільтрації (рис. 3.30).

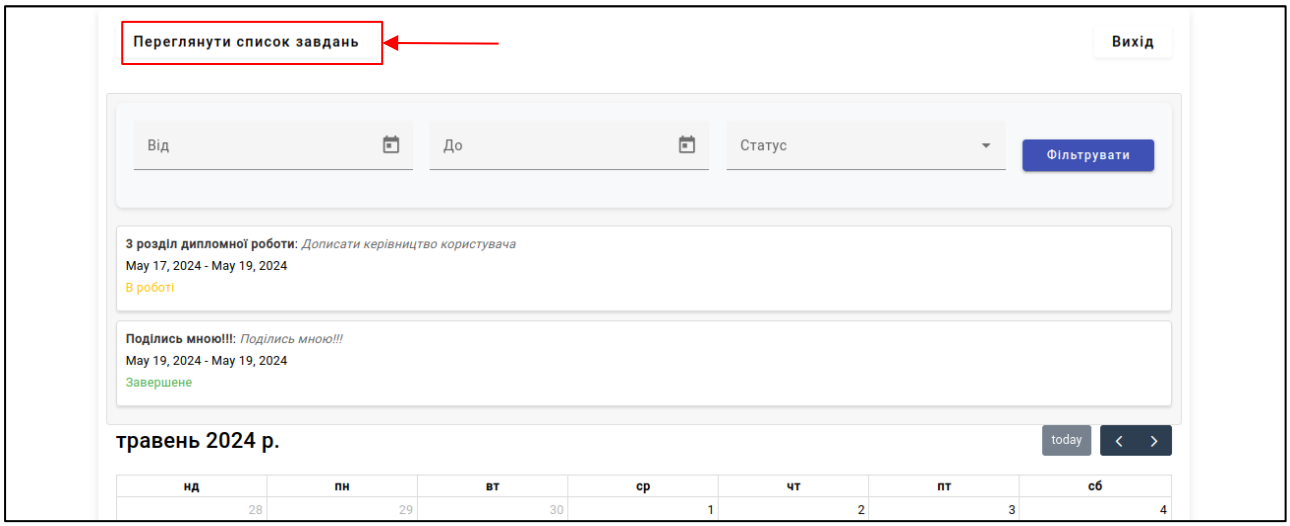

Рисунок 3.30 – Список завдань

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було успішно розроблено вебдодаток планувальника завдань, який реалізовано засобами Angular та Node.js. Проведений аналіз предметної області, проєктування системи та її реалізація дали змогу досягти головної мети роботи – створити функціональний, надійний та зручний інструмент для планування завдань.

У відповідності з метою кваліфікаційної роботи було розроблено вебдодаток планувальника завдань з використаннями наступних технологій:

– MongoDB для зберігання даних;
– Node.js для реєстрації/авторизації 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 вебдодатку:

– сформовані вимоги до вебдодатку (визначено функціональні та нефункціональні вимоги до вебдодатку (інтерфейс користувача, продуктивність, безпеку та підтримка різних браузерів));
– спроєктована та побудована структура вебдодатку (створено діаграми: варіантів використання, діяльності, послідовності та розгортання; описані прецеденти системи);
– реалізовано вебдодаток планувальника завдань (наведена інструкція по налаштуванню MongoDB Atlas та надано керівництво користувача);
– протестована робота вебдодатку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Angular Developer Documentation. URL: https://angular.io/docs/ (дата звернення: 14.03.2024).

2. Asana. URL: https://asana.com/ (дата звернення: 23.03.2024).

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. FullCalendar Documentation. URL: https://fullcalendar.io/docs (дата звернення: 05.04.2024).

7. Google Tasks. URL: https://support.google.com/tasks/answer/7675772 (дата звернення: 24.03.2024).

8. 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.

9. Microsoft To Do. URL: https://to-do.office.com/tasks/ (дата звернення: 24.03.2024).

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

11. Node.js Developer Documentation. URL: https://nodejs.org/en/docs (дата звернення: 14.03.2024).

12. Parvin R. Jasmine Testing for Angular 4 Apps: A Practical Guide to Unit and Integration Testing for Reliable Angular Deployments. Michigan : Independently published, 2024. 305 p.

13. Prodromou F. Mastering Angular Reactive Forms: Build Solid Expertise in Reactive Forms using Form Control, Form Group, Form Array, Validators, Testing and more with ... Real-World Use Cases. New Delhi : BPB Publications, 2021. 180 p.

14. 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.

15. Todoist. URL: https://todoist.com/ (дата звернення: 23.03.2024).

16. Trello. URL: https://trello.com/ (дата звернення: 25.03.2024).

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

ДОДАТОК А

Серверна частина

server.js

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const schedule = require('node-schedule');
const authRoutes = require('./routes/authRoutes');
const taskRoutes = require('./routes/taskRoutes');
const {checkTasksAndNotify} = require("./reminderService");
require('dotenv').config()

const app = express();
app.use(cors());
app.use(express.json());

mongoose.connect('mongodb+srv://admin:admin@notification.mfpappx.mongodb.net/').then(() =>
console.log('MongoDB connected'))
  .catch(err => console.log(err));

app.use('/api', authRoutes);
app.use('/api', taskRoutes);

// Планування перевірки завдань кожну годину
schedule.scheduleJob('* * * * *', function() {
  console.log('Running the task check');
  checkTasksAndNotify();
});


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

taskRoutes.js

```
const express = require('express');
const router = express.Router();
const taskController = require('../controllers/taskController');
const emailService = require('../emailService');
const Task = require('../models/Task');
const authMiddleware = require('../middleware/authMiddleware');

router.post('/tasks', authMiddleware, taskController.createTask);
router.get('/tasks', authMiddleware, taskController.getTasks);
router.put('/tasks/:id', authMiddleware, taskController.updateTask);
router.delete('/tasks/:id', authMiddleware, taskController.deleteTask);
router.get('/tasks/:id', authMiddleware, taskController.getTaskById);
router.post('/tasks/:id/share', authMiddleware, async (req, res) => {
  try {
    const task = await Task.findById(req.params.id);
    if (!task) {
      return res.status(404).json({ message: 'Task not found' });
    }
    const subject = `Shared Task: ${task.title}`;
    const text = `Task Details:\n\nTitle: ${task.title}\nDescription: ${task.description}\nStart Time:
${task.startTime}\nEnd Time: ${task.endTime}`;
    await emailService.sendEmail(req.body.to, subject, text);
    res.json({ message: 'Task shared via email' });  // Відправляємо JSON відповідь
  } catch (error) {
    res.status(500).json({ message: 'Error sharing task via email: ' + error.message });
  }
});

module.exports = router;
```

taskController.js

```
const Task = require('../models/Task');

exports.createTask = async (req, res) => {
  const { title, description, startTime, endTime, status } = req.body;
  const userId = req.userId;
  const userEmail = req.userEmail;
  try {
```

```
    const newTask = new Task({
      title,
      description,
      startTime,
      endTime,
      userId,
      userEmail,
      status
    });
    const savedTask = await newTask.save();
    res.json(savedTask);
  } catch (error) {
    res.status(500).send(error);
  }
};

// Оновлення завдання
exports.updateTask = async (req, res) => {
  const { title, description, startTime, endTime, userId, userEmail, status } = req.body;
  try {
    const task = await Task.findById(req.params.id);
    if (!task) {
      return res.status(404).send('Task not found');
    }

    task.title = title;
    task.description = description;
    task.startTime = startTime;
    task.endTime = endTime;
    task.userId = req.userId;
    task.userEmail = req.userEmail;
    task.status = status;
    await task.save();
    res.json(task);
  } catch (error) {
    res.status(500).send('Server error');
  }
};
```

```
exports.getTasks = async (req, res) => {
  try {
    const tasks = await Task.find({ userId: req.userId });
    res.json(tasks);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

exports.deleteTask = async (req, res) => {
  try {
    await Task.findByIdAndDelete(req.params.id);
    res.status(204).send();
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};



exports.getTaskById = async (req, res) => {
  try {

    const task = await Task.findById(req.params.id);
    if (!task) {
      return res.status(404).send('Task not found');
    }
    res.json(task);
  } catch (error) {
    console.log(error);
    res.status(500).send('Server error');
  }
};
```

Task.js

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

const taskSchema = new mongoose.Schema({
  title: { type: String, required: true },
  description: { type: String },
```

```
  startTime: { type: Date, required: true },
  endTime: { type: Date, required: true },
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  userEmail: { type: String },
  notified: { type: Boolean, default: false },
  status: {
    type: String,
    required: true,
    enum: ['new', 'in_progress', 'completed', 'overdue'],
    default: 'new'
  }
});

module.exports = mongoose.model('Task', taskSchema);
```

ДОДАТОК Б

Тестування TaskFormComponent

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { TaskFormComponent } from './task-form.component';
import { FormBuilder, ReactiveFormsModule } from '@angular/forms';
import { TaskService } from '../../services/task.service';
import { MatDialog } from '@angular/material/dialog';
import { of, throwError } from 'rxjs';
import {Task} from "../../models/Task";
describe('TaskFormComponent', () => {
  let component: TaskFormComponent;
  let fixture: ComponentFixture<TaskFormComponent>;
  let taskService: jasmine.SpyObj<TaskService>;
  let dialog: jasmine.SpyObj<MatDialog>;
  beforeEach(async () => {
    const taskServiceSpy = jasmine.createSpyObj('TaskService', ['addTask', 'updateTask', 'deleteTask']);
    const dialogSpy = jasmine.createSpyObj('MatDialog', ['open']);
    await TestBed.configureTestingModule({
      declarations: [ TaskFormComponent ],
      imports: [ ReactiveFormsModule ],
      providers: [
        { provide: TaskService, useValue: taskServiceSpy },
        { provide: MatDialog, useValue: dialogSpy },
        FormBuilder
      ]
    })
      .compileComponents();
    fixture = TestBed.createComponent(TaskFormComponent);
    component = fixture.componentInstance;
    taskService = TestBed.inject(TaskService) as jasmine.SpyObj<TaskService>;
    dialog = TestBed.inject(MatDialog) as jasmine.SpyObj<MatDialog>;
  });
  it('should create', () => {
    expect(component).toBeTruthy();
  });
```

```
  it('should initialize the form on ngOnInit with empty task', () => {
    component.ngOnInit();
    expect(component.taskForm).toBeDefined();
    expect(component.taskForm.get('title')?.value).toEqual('');
  });
  describe('onSubmit', () => {
    it('should call updateTask if task exists', () => {
      const task: Task = { _id: '123', title: 'Test Task', description: 'Do something', startTime: new Date(),
endTime: new Date(), userId: 'user1', userEmail: 'user@example.com', notified: true, status: 'new' };
      component.task = task;
      component.ngOnInit();
      taskService.updateTask.and.returnValue(of(task));
      component.onSubmit();
      expect(taskService.updateTask).toHaveBeenCalled();
    });
    it('should call addTask if task does not exist', () => {
      component.ngOnInit();
      taskService.addTask.and.returnValue(of({ _id: '123', ...component.taskForm.value } as Task));
      component.onSubmit();
      expect(taskService.addTask).toHaveBeenCalled();
    });
    it('should handle errors during task addition', () => {
      component.ngOnInit();
      taskService.addTask.and.returnValue(throwError(() => new Error('Failed to add task')));
      spyOn(window, 'alert');
      component.onSubmit();
      expect(window.alert).toHaveBeenCalledWith('Failed to add task: Failed to add task');
    });
  });
  it('should emit formClose true when close is called', () => {
    spyOn(component.formClose, 'emit');
    component.close();
    expect(component.formClose.emit).toHaveBeenCalledWith(true);
  });
});
```