МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБДОДАТКУ ДЛЯ ПЛАНУВАННЯ ТА КОНТРОЛЮ ВИКОНАННЯ ЗАДАЧ»**

Виконав: студент 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
В.О. Слєпцов
(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 завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

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)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Слєпцову Владиславу Олексійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебдодатку для планування та контролю виконання задач

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та постановка задачі.
2. Проєктування вебдодатку для планування та контролю виконання задач.
3. Розробка та реалізація вебдодатку для планування та контролю виконання задач.

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 09.12.2025 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.	19.12.2025	
2.	Збір вихідних даних.	12.01.2026	
3.	Обробка методичних та теоретичних	20.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	23.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2026	

Студент _____ В.О. Слєпцов
(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 для планування та контролю виконання задач»: 51 с., 26 рис., 1 табл., 30 джерел.

ВЕБДОДАТОК, ВИКОНАННЯ ЗАДАЧ, ІНТЕРФЕЙС КОРИСТУВАЧА, КОНТРОЛЬ ЗАДАЧ, ПЛАНУВАННЯ, ПОСТАВЛЕННЯ ЗАВДАНЬ, СИСТЕМА УПРАВЛІННЯ ЗАДАЧАМИ, УПРАВЛІННЯ ПРОЄКТАМИ, JWT-АВТЕНТИФІКАЦІЯ, POSTGRESQL.

Об’єкт дослідження – процес планування, організації та контролю виконання задач у веборієнтованих інформаційних системах.

Мета роботи: розробка вебдодатку для ефективного планування, розподілу та контролю виконання задач, який забезпечує зручну взаємодію користувачів, моніторинг стану виконання завдань та підвищення продуктивності роботи.

Методи дослідження – аналіз та узагальнення науково-технічної літератури, системний аналіз предметної області, об’єктно-орієнтоване проєктування інформаційної системи, моделювання бази даних, розробка та тестування програмного забезпечення.

У кваліфікаційній роботі проведено аналіз сучасних підходів до планування та контролю виконання задач, а також досліджено існуючі програмні рішення у сфері управління проєктами та персональною продуктивністю. На основі результатів аналізу сформульовано вимоги до програмного продукту та спроєктовано архітектуру вебдодатку. У роботі розроблено вебдодаток, що забезпечує реєстрацію та автентифікацію користувачів, створення й редагування задач, встановлення пріоритетів та термінів виконання, зміну статусів задач, пошук і фільтрацію даних, а також перегляд статистики виконання завдань.

SUMMARY

Bachelor’s qualifying paper “Development of a Web Application for Task Planning and Monitoring”: 51 pages, 26 figures, 1 table, 30 references.

WEB APPLICATION, TASK EXECUTION, USER INTERFACE, TASK CONTROL, PLANNING, TASK ASSIGNMENT, TASK MANAGEMENT SYSTEM, PROJECT MANAGEMENT, JWT AUTHENTICATION, POSTGRESQL.

The object of research is the process of planning, organizing, and controlling task execution in web-based information systems.

The purpose of the work is to develop a web application for effective planning, distribution, and control of task execution, which provides convenient user interaction, monitoring of task progress, and improvement of work productivity.

Research methods include the analysis and generalization of scientific and technical literature, system analysis of the subject area, object-oriented design of the information system, database modeling, software development, and software testing.

In the qualification thesis, an analysis of modern approaches to task planning and execution control was conducted, and existing software solutions in the field of project management and personal productivity were examined. Based on the results of the analysis, the requirements for the software product were formulated and the architecture of the web application was designed. The thesis presents the development of a web application that provides user registration and authentication, task creation and editing, setting priorities and deadlines, changing task statuses, searching and filtering data, as well as viewing task completion statistics.

ЗМІСТ

Завдання на кваліфікаційну роботу……..…..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз предметної області та постановка задачі 9
1.1 Аналіз предметної області 9
1.2 Огляд існуючих систем планування та контролю виконання задач.. 11
1.3 Порівняльний аналіз аналогів..... 13
1.4 Постановка задачі та формування вимог до системи..... 13
2 Проєктування вебдодатку для планування та контролю виконання задач.. 16
2.1 Вибір технологій розробки вебдодатку 16
2.2 Проєктування архітектури системи 17
2.3 Проєктування бази даних..... 21
2.4 Моделювання функціональності системи 24
2.5 Проєктування користувацького інтерфейсу 27
3 Розробка та реалізація вебдодатку для планування та контролю виконання задач..... 32
3.1 Реалізація серверної частини вебдодатку..... 32
3.2 Реалізація клієнтської частини вебдодатку..... 35
3.3 Реалізація механізмів автентифікації та авторизації..... 38
3.4 Реалізація функціоналу управління задачами..... 42
Висновки 47
Перелік посилань..... 49

ВСТУП

У сучасних умовах цифрової трансформації суспільства інформаційні технології стали невід'ємною складовою діяльності організацій, підприємств та окремих користувачів. Ефективне управління завданнями є важливим фактором підвищення продуктивності праці, забезпечення своєчасного виконання робіт та досягнення поставлених цілей. Зі збільшенням кількості проєктів, завдань та учасників процесу виникає потреба у використанні спеціалізованих програмних засобів, які дозволяють планувати, розподіляти та контролювати виконання завдань у зручному та зрозумілому форматі.

Сьогодні існує значна кількість систем управління завданнями та проєктами, однак багато з них мають надлишковий функціонал, складний інтерфейс або потребують платної підписки для повноцінного використання. Тому актуальним є створення вебдодатку, який забезпечуватиме основні можливості для планування та контролю виконання задач, матиме зручний інтерфейс користувача та буде доступним через веббраузер без необхідності встановлення додаткового програмного забезпечення.

Вебтехнології надають широкі можливості для розробки сучасних інформаційних систем, що працюють у режимі реального часу та забезпечують доступ до даних з різних пристроїв. Використання вебдодатків для управління задачами дозволяє централізовано зберігати інформацію, швидко оновлювати дані, організовувати взаємодію між користувачами та контролювати хід виконання поставлених завдань.

Об'єктом дослідження є процес планування, організації та контролю виконання задач у веборієнтованих інформаційних системах.

Предметом дослідження є методи, технології та програмні засоби розробки вебдодатків для управління задачами та контролю їх виконання.

Метою роботи є розробка вебдодатку для планування та контролю виконання задач, який забезпечує зручне створення, редагування та

відстеження завдань, а також підвищує ефективність організації робочого процесу користувачів.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

– проаналізувати сучасні підходи до організації процесу планування та контролю виконання задач;
– дослідити існуючі програмні рішення та визначити їх переваги і недоліки;
– сформувати функціональні та нефункціональні вимоги до системи;
– спроєктувати архітектуру вебдодатку та структуру бази даних;
– реалізувати програмне забезпечення із використанням сучасних вебтехнологій;
– забезпечити механізми автентифікації та авторизації користувачів;
– провести тестування розробленого програмного продукту та оцінити результати його роботи.

Методи дослідження включають аналіз та узагальнення науково-технічної літератури, системний аналіз предметної області, об’єктно-орієнтоване проєктування інформаційної системи, моделювання бази даних, розробку та тестування програмного забезпечення.

Практичне значення одержаних результатів полягає у створенні вебдодатку, який може бути використаний для організації особистих, навчальних або робочих завдань, забезпечуючи ефективне планування діяльності та контроль виконання поставлених цілей.

Структура кваліфікаційної роботи складається зі вступу, трьох розділів основної частини, висновків та списку використаних джерел. У роботі розглянуто теоретичні аспекти предметної області, виконано проєктування та реалізацію вебдодатку, а також проведено тестування розробленої системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

У сучасному світі ефективне управління завданнями є важливою складовою успішної діяльності як окремих користувачів, так і організацій різного масштабу. Зростання обсягів інформації, збільшення кількості проєктів та необхідність координації роботи між учасниками процесу створюють потребу у використанні спеціалізованих інформаційних систем для планування та контролю виконання задач. Такі системи дозволяють впорядковувати робочі процеси, розподіляти обов’язки між виконавцями та контролювати виконання поставлених цілей.

Планування задач є одним із ключових етапів управління діяльністю [4]. Воно передбачає визначення переліку завдань, встановлення термінів їх виконання, призначення відповідальних осіб та визначення пріоритетності. Грамотно організований процес планування сприяє раціональному використанню часу та ресурсів, зменшує ризик виникнення помилок і забезпечує своєчасне досягнення поставлених результатів [1].

Контроль виконання задач є невід’ємною частиною процесу управління [3]. Його основною метою є відстеження поточного стану виконання завдань, виявлення можливих відхилень від плану та своєчасне реагування на проблеми, що виникають у процесі роботи. Використання автоматизованих засобів контролю дозволяє оперативно отримувати інформацію про виконання задач, аналізувати продуктивність роботи користувачів та приймати обґрунтовані управлінські рішення.

Традиційні методи організації роботи, такі як паперові записники, таблиці або текстові документи, часто виявляються недостатньо ефективними в умовах великої кількості завдань та учасників процесу. Такі підходи ускладнюють контроль виконання робіт, створюють ризик втрати інформації

та не забезпечують належного рівня взаємодії між користувачами. Саме тому все більшого поширення набувають цифрові системи управління задачами.

Сучасні веборієнтовані інформаційні системи дозволяють автоматизувати процеси створення, редагування та контролю виконання задач. Завдяки використанню вебтехнологій користувачі можуть отримувати доступ до системи з будь-якого пристрою, який має підключення до мережі Інтернет. Це забезпечує мобільність, зручність використання та можливість оперативного обміну інформацією між учасниками робочого процесу.

Предметна область управління задачами включає декілька основних сутностей. Центральним елементом є задача, яка містить опис роботи, термін виконання, статус, пріоритет та іншу службову інформацію. Користувачі системи можуть створювати нові задачі, змінювати їх параметри, відстежувати прогрес виконання та переглядати історію змін. Для більш ефективної організації роботи задачі можуть об’єднуватися у проєкти або категорії.

Важливим аспектом сучасних систем планування є використання механізмів автентифікації та авторизації користувачів [24]. Це дозволяє забезпечити безпечний доступ до даних, розмежувати права користувачів та захистити інформацію від несанкціонованого доступу. Крім того, система повинна гарантувати цілісність даних та стабільність роботи навіть при одночасній роботі великої кількості користувачів.

Аналіз предметної області показав, що існує потреба у створенні вебдодатку, який поєднує зручний користувацький інтерфейс, ефективні засоби планування та контролю виконання задач, а також забезпечує надійне зберігання інформації. Такий програмний продукт дозволить підвищити продуктивність роботи користувачів, спростити процес організації діяльності та покращити контроль за виконанням поставлених завдань.

Таким чином, розробка вебдодатку для планування та контролю виконання задач є актуальним напрямом розвитку інформаційних систем, що відповідає сучасним потребам користувачів та тенденціям цифровізації управлінських процесів.

1.2 Огляд існуючих систем планування та контролю виконання задач

На сьогоднішній день існує велика кількість програмних продуктів, призначених для планування, організації та контролю виконання задач. Такі системи використовуються як окремими користувачами для особистого планування, так і командами різного масштабу для управління проєктами та координації спільної діяльності. Їх основною метою є підвищення ефективності роботи шляхом автоматизації процесів створення, розподілу та моніторингу виконання завдань.

Однією з найбільш відомих систем управління задачами є Trello [19]. Дана платформа використовує принцип канбан-дошок, що дозволяє візуалізувати процес виконання завдань за допомогою карток і списків. Користувачі можуть створювати задачі, призначати виконавців, встановлювати терміни виконання та відстежувати прогрес роботи. Основними перевагами Trello є простота використання, зручний інтерфейс та можливість інтеграції з іншими сервісами. Водночас при роботі зі складними проєктами функціональність системи може бути недостатньою без використання додаткових розширень.

Іншою популярною системою є Asana [19], яка надає розширені можливості для управління проєктами та завданнями. Система дозволяє створювати проєкти, групувати задачі, формувати залежності між ними, контролювати терміни виконання та аналізувати результати роботи команди. Перевагою Asana є широкий набір інструментів для організації командної роботи. Недоліком можна вважати відносно складний інтерфейс для нових користувачів та обмеження безкоштовної версії.

Серед професійних систем управління проєктами особливе місце займає Jira [26]. Вона широко використовується у сфері розробки програмного забезпечення та підтримує різні методології управління проєктами, зокрема Agile та Scrum. Система надає можливість створення складних робочих

процесів, управління спринтами, формування звітності та контролю виконання завдань. Незважаючи на високу функціональність, Jira може бути надмірно складною для невеликих команд або індивідуального використання.

Також значного поширення набула система Microsoft Planner [30], яка інтегрується з екосистемою Microsoft 365. Вона дозволяє створювати завдання, розподіляти їх між учасниками команди, контролювати строки виконання та отримувати узагальнену інформацію про стан проєкту. Основною перевагою є тісна інтеграція з іншими продуктами Microsoft, проте використання системи часто потребує наявності корпоративної підписки.

Для особистого планування широко використовуються такі сервіси, як Todoist та Microsoft To Do. Вони забезпечують створення списків справ, встановлення нагадувань, сортування завдань за категоріями та пріоритетами. Дані системи характеризуються простотою використання, проте їх функціональні можливості можуть бути недостатніми для організації складних командних проєктів.

Проведений огляд показує, що сучасні системи управління задачами пропонують широкий спектр можливостей для планування та контролю виконання робіт. Водночас більшість існуючих рішень мають певні недоліки, серед яких надмірна складність інтерфейсу, обмеження безкоштовних версій, залежність від сторонніх сервісів або наявність функціоналу, який не завжди потрібний кінцевому користувачеві.

З огляду на результати аналізу існуючих програмних продуктів можна зробити висновок про доцільність розробки власного вебдодатку для планування та контролю виконання задач. Така система повинна поєднувати простоту використання, достатню функціональність, зручний інтерфейс та можливість ефективного управління завданнями як для окремих користувачів, так і для невеликих команд.

1.3 Порівняльний аналіз аналогів

Для більш детального аналізу функціональних можливостей існуючих систем управління задачами було проведено їх порівняння за основними критеріями [19, 26, 30]. Результати порівняльного аналізу наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння існуючих систем управління задачами

Функціональна можливість	**Trello**	**Asana**	**Jira**	**Microsoft Planner**	**Розроблюваний вебдодаток**
Створення задач	+	+	+	+	+
Редагування задач	+	+	+	+	+
Призначення виконавців	+	+	+	+	+
Встановлення дедлайнів	+	+	+	+	+
Категорії та пріоритети	+	+	+	Частково	+
Пошук та фільтрація задач	+	+	+	+	+
Авторизація користувачів	+	+	+	+	+
Статистика виконання	Частково	+	+	Частково	+

Як видно з таблиці 1.1, усі розглянуті системи мають базові засоби для управління задачами, однак відрізняються набором додаткових можливостей та складністю використання.

1.4 Постановка задачі та формування вимог до системи

Проведений аналіз предметної області та огляд існуючих систем планування і контролю виконання задач дозволив визначити основні проблеми, які виникають під час організації робочого процесу. До таких проблем належать складність контролю великої кількості завдань, недостатня наочність інформації про стан їх виконання, обмеження безкоштовних версій популярних сервісів, а також надлишкова функціональність окремих програмних продуктів, яка ускладнює їх використання.

Для вирішення зазначених проблем доцільно розробити вебдодаток для планування та контролю виконання задач, який забезпечуватиме зручний доступ до інформації про завдання, можливість їх створення та редагування, контроль термінів виконання та моніторинг поточного стану роботи. Розроблювана система повинна бути орієнтована на використання як окремими користувачами, так і невеликими командами.

Основною задачею розробки є створення вебдодатку, що дозволить автоматизувати процес управління задачами та забезпечить зручний інструмент для організації діяльності користувачів. Система повинна сприяти підвищенню продуктивності роботи шляхом централізованого зберігання інформації про завдання та надання засобів для контролю їх виконання.

На основі проведеного аналізу сформовано **функціональні вимоги** до системи [6, 7]:

– реєстрація та автентифікація користувачів;
– авторизація користувачів відповідно до їх прав доступу;
– створення, редагування та видалення задач;
– встановлення назв, описів, пріоритетів та термінів виконання задач;
– зміна статусу виконання завдань;
– перегляд списку власних задач;
– пошук та фільтрація задач за різними параметрами;
– відображення інформації про поточний стан виконання завдань;
– формування статистичних даних щодо виконання задач;
– збереження інформації у базі даних.

Крім функціональних вимог, система повинна відповідати таким **нефункціональним вимогам** [18]:

– забезпечення надійного зберігання даних;
– захист персональної інформації користувачів;
– простота та зручність користувацького інтерфейсу;
– висока швидкодія під час виконання основних операцій;
– можливість доступу через сучасні веббраузери;

– масштабованість та можливість подальшого розширення функціоналу;
– підтримка багатокористувацького режиму роботи.

Для реалізації зазначених вимог доцільно використовувати сучасні вебтехнології, що забезпечують ефективну взаємодію між клієнтською та серверною частинами системи. Важливим аспектом є також використання реляційної бази даних для зберігання інформації про користувачів та задачі, а також механізмів автентифікації та авторизації для забезпечення безпеки доступу до даних.

Таким чином, постановка задачі полягає у розробці вебдодатку для планування та контролю виконання задач, який забезпечуватиме необхідний функціонал для управління завданнями, матиме зручний інтерфейс користувача, гарантуватиме безпечне зберігання даних та дозволятиме ефективно організовувати робочий процес користувачів.

2 ПРОЄКТУВАННЯ ВЕБДОДАТКУ ДЛЯ ПЛАНУВАННЯ ТА КОНТРОЛЮ ВИКОНАННЯ ЗАДАЧ

2.1 Вибір технологій розробки вебдодатку

Одним із важливих етапів створення програмного забезпечення є вибір технологій, які будуть використовуватися під час розробки системи. Від правильності цього вибору залежать продуктивність, масштабованість, безпека та зручність подальшого супроводу програмного продукту. Для розробки вебдодатку планування та контролю виконання задач було обрано сучасний стек технологій, який забезпечує ефективну взаємодію між клієнтською та серверною частинами системи.

Для реалізації клієнтської частини вебдодатку доцільно використовувати бібліотеку React. Дана технологія дозволяє створювати сучасні односторінкові вебзастосунки з високою швидкістю роботи та зручним користувацьким інтерфейсом. Основною перевагою React є компонентний підхід до розробки, який забезпечує повторне використання елементів інтерфейсу та спрощує підтримку програмного коду. Крім того, React має велику спільноту розробників та широкий набір додаткових бібліотек для реалізації різноманітних функціональних можливостей.

Для розробки серверної частини було обрано фреймворк NestJS. Даний фреймворк побудований на платформі Node.js та підтримує використання мови програмування TypeScript. NestJS забезпечує модульну архітектуру застосунку, що сприяє кращій організації програмного коду та полегшує його подальший розвиток. Також фреймворк надає вбудовані механізми для створення REST API, обробки запитів користувачів та реалізації засобів автентифікації й авторизації.

Для зберігання інформації про користувачів, задачі та інші дані системи було обрано систему керування базами даних PostgreSQL. Вона є однією з

найбільш популярних реляційних баз даних завдяки високій надійності, продуктивності та підтримці складних операцій обробки даних. PostgreSQL забезпечує цілісність даних, підтримує транзакції та дозволяє ефективно працювати з великими обсягами інформації.

Для взаємодії серверної частини з базою даних доцільно використовувати ORM-засіб, який спрощує виконання операцій створення, читання, оновлення та видалення даних. Застосування ORM дозволяє зменшити обсяг програмного коду та підвищити швидкість розробки програмного забезпечення.

Для забезпечення безпеки системи використовується механізм автентифікації на основі JWT-токенів. Такий підхід дозволяє безпечно ідентифікувати користувачів під час взаємодії з вебдодатком та контролювати доступ до функціональних можливостей системи. Використання JWT є одним із найпоширеніших рішень для сучасних вебзастосунків.

Передача даних між клієнтською та серверною частинами системи здійснюється за допомогою REST API. Даний підхід забезпечує стандартизований спосіб обміну інформацією та дозволяє легко інтегрувати систему з іншими програмними продуктами за необхідності.

Обрані технології відповідають сучасним вимогам до веброзробки та забезпечують можливість створення надійного, безпечного та масштабованого вебдодатку для планування та контролю виконання задач. Використання React, NestJS та PostgreSQL дозволяє реалізувати необхідний функціонал системи, забезпечити зручність роботи користувачів та створити основу для подальшого розвитку програмного продукту.

2.2 Проєктування архітектури системи

Проєктування архітектури є одним із ключових етапів розробки програмного забезпечення, оскільки саме на цьому етапі визначається

структура майбутньої системи, взаємозв’язки між її компонентами та способи обробки даних. Грамотно спроєктована архітектура забезпечує зручність супроводу програмного продукту, можливість його масштабування та ефективне використання ресурсів.

Для розроблюваного вебдодатку було обрано трирівневу архітектуру, яка складається з клієнтського рівня, серверного рівня та рівня зберігання даних. Загальна структура вебдодатку представлена на рисунку 2.1. Такий підхід є одним із найбільш поширених у сучасній веброзробці та дозволяє чітко розділити функціональність системи між окремими компонентами.

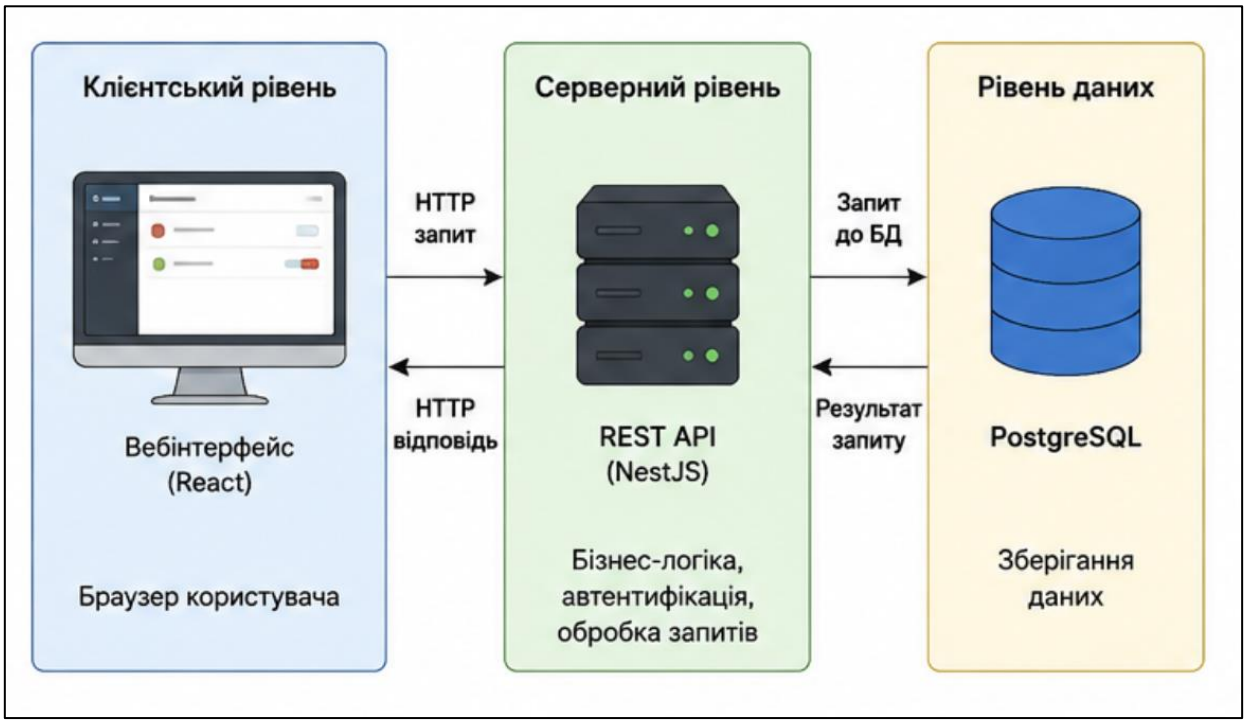


Рисунок 2.1 – Загальна архітектура системи

Клієнтський рівень представлений вебінтерфейсом користувача, який забезпечує взаємодію із системою через браузер. Основними функціями клієнтської частини є відображення інформації про задачі, отримання даних від користувача, формування запитів до серверної частини та відображення результатів обробки отриманих даних. Інтерфейс користувача повинен бути інтуїтивно зрозумілим, зручним та забезпечувати швидкий доступ до основних функцій системи.

Серверний рівень реалізує бізнес-логіку вебдодатку. Він відповідає за обробку запитів користувачів, перевірку коректності введених даних, виконання операцій над задачами, управління обліковими записами користувачів та взаємодію з базою даних. Крім того, серверна частина забезпечує механізми автентифікації та авторизації користувачів, що дозволяє контролювати доступ до ресурсів системи.

Рівень зберігання даних представлений реляційною базою даних PostgreSQL. У базі даних зберігається інформація про користувачів, задачі, статуси завдань, пріоритети та інші сутності системи. Використання централізованого сховища даних дозволяє забезпечити цілісність інформації та швидкий доступ до необхідних відомостей.

Обмін даними між клієнтською та серверною частинами здійснюється за допомогою REST API. Послідовність взаємодії компонентів під час обробки запитів користувача показана на рисунку 2.2. При виконанні певної дії користувачем клієнтська частина формує HTTP-запит та надсилає його на сервер. Після обробки запиту сервер виконує необхідні операції та повертає результат у вигляді відповіді, яка відображається користувачеві через вебінтерфейс.

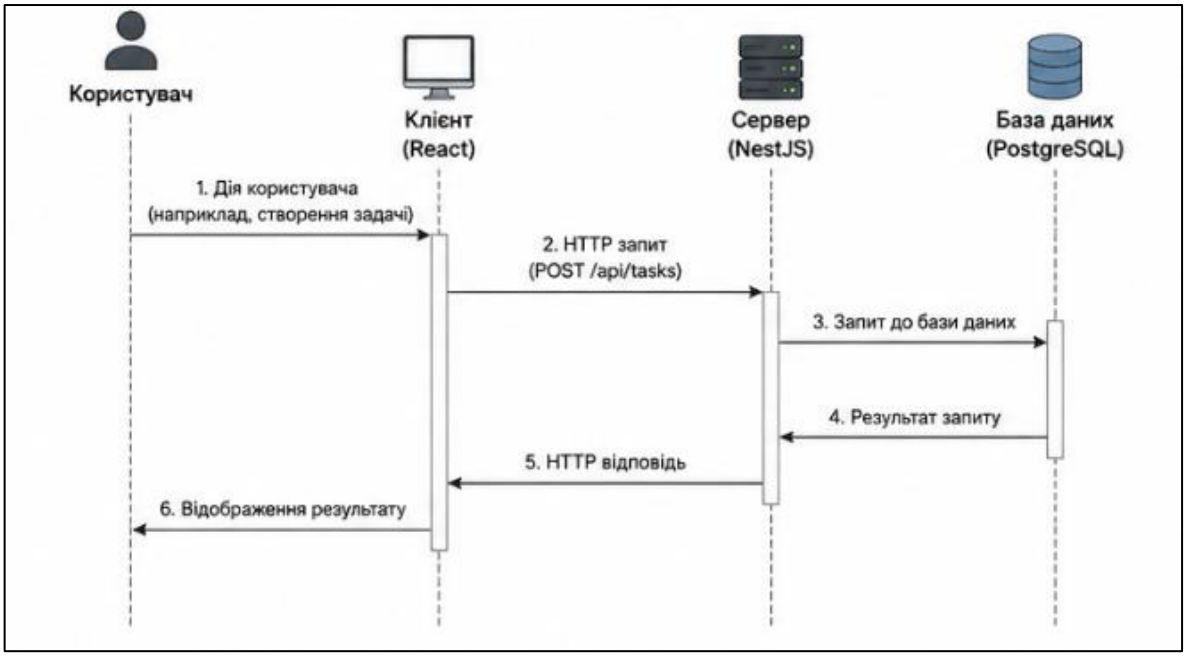


Рисунок 2.2 – Послідовність взаємодії компонентів системи

Для підвищення безпеки системи використовується механізм автентифікації на основі JWT-токенів. Схему автентифікації користувачів із використанням JWT-токенів наведено на рисунку 2.3.

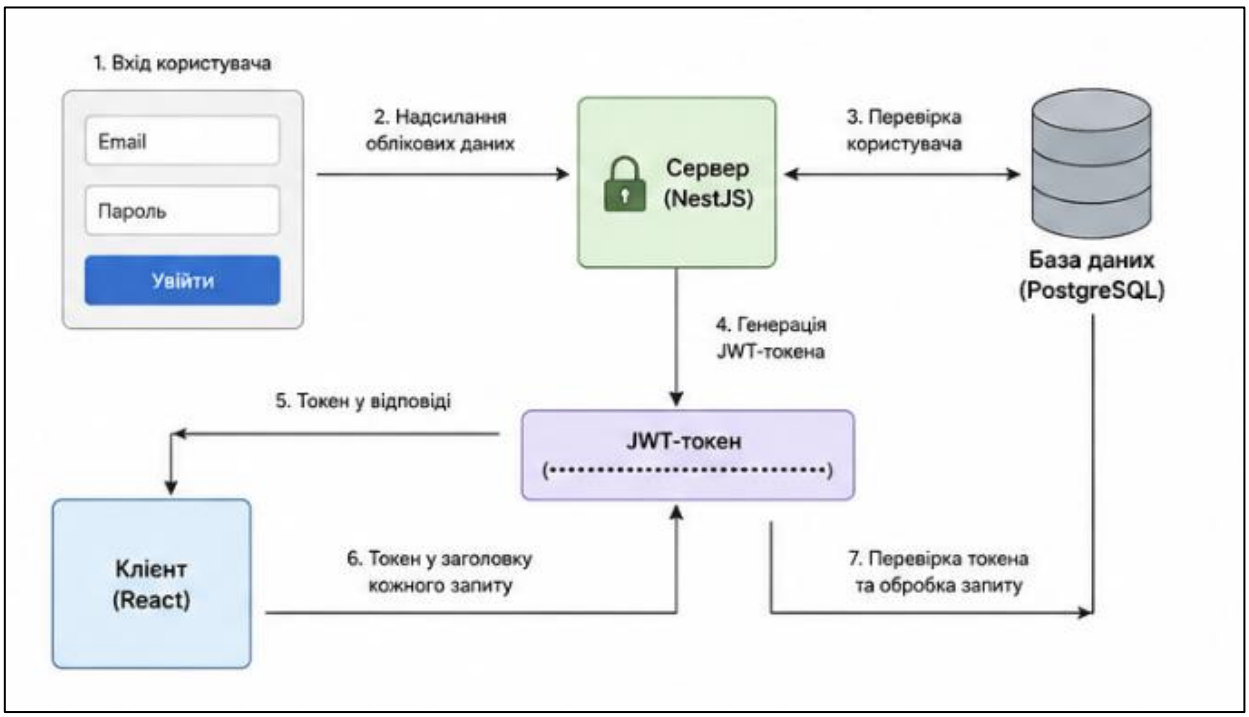


Рисунок 2.3 – Механізм автентифікації користувачів за допомогою JWT

Після успішного входу до системи користувач отримує токен доступу, який використовується під час подальших звернень до серверної частини. Це дозволяє перевіряти права доступу користувача та захищати систему від несанкціонованого використання.

З точки зору логічної структури система складається з декількох основних модулів (див. рис. 2.4). До них належать модуль управління користувачами, модуль автентифікації та авторизації, модуль управління задачами, модуль статистики та модуль роботи з базою даних. Кожен із модулів виконує визначений набір функцій та взаємодіє з іншими компонентами системи через стандартизовані інтерфейси.

Обрана архітектура забезпечує незалежність окремих компонентів системи, що спрощує її модернізацію та розширення. За необхідності можуть бути додані нові функціональні модулі без суттєвих змін існуючої структури

програмного забезпечення. Це дозволяє адаптувати вебдодаток до змін вимог користувачів та забезпечувати його подальший розвиток.



Рисунок 2.4 – Логічна структура системи

Таким чином, використання трирівневої архітектури дозволяє створити надійну, масштабовану та безпечну систему для планування та контролю виконання задач, яка відповідає сучасним вимогам до веборієнтованих інформаційних систем.

2.3 Проєктування бази даних

База даних є одним із ключових компонентів будь-якої інформаційної системи, оскільки забезпечує зберігання, обробку та отримання інформації, необхідної для функціонування програмного забезпечення. Під час розробки вебдодатку для планування та контролю виконання задач особлива увага приділялася проєктуванню структури бази даних, яка повинна забезпечувати надійне зберігання даних, цілісність інформації та високу швидкість виконання запитів.

Для реалізації системи було обрано реляційну систему керування базами даних PostgreSQL. Даний вибір обумовлений високою продуктивністю, підтримкою транзакцій, широкими можливостями масштабування та високим рівнем безпеки. Крім того, PostgreSQL є безкоштовною системою керування базами даних з відкритим програмним кодом, що робить її популярним рішенням для сучасних вебзастосунків.

Для наочного представлення структури бази даних використовується ER-діаграма, яка демонструє основні сутності системи та зв’язки між ними. ER-діаграма дозволяє краще зрозуміти логіку організації даних та є важливим елементом проєктної документації.

Структура бази даних була розроблена на основі аналізу предметної області та функціональних вимог до системи. Основними сутностями, що використовуються у вебдодатку, є користувачі та задачі. Саме вони забезпечують реалізацію основного функціоналу системи та взаємодію між її компонентами.

Сутність «Користувач» призначена для зберігання інформації про зареєстрованих користувачів системи. Для кожного користувача зберігаються унікальний ідентифікатор, ім’я, адреса електронної пошти, пароль у зашифрованому вигляді, дата створення облікового запису та інші службові дані. Використання унікальної електронної пошти дозволяє однозначно ідентифікувати користувача під час автентифікації.

Сутність «Задача» містить інформацію про завдання, які створюються користувачами. Для кожної задачі зберігаються назва, опис, дата створення, кінцевий термін виконання, пріоритет, поточний статус та посилання на користувача, який є власником або виконавцем завдання. Така структура дозволяє ефективно організовувати процес планування та контролю виконання робіт.

Між сутностями «Користувач» та «Задача» встановлюється зв’язок типу «один до багатьох». Це означає, що один користувач може мати багато задач, тоді як кожна задача належить лише одному користувачу. Подібний підхід

дозволяє забезпечити логічну організацію даних та спрощує виконання запитів до бази даних.

Структуру бази даних та зв’язки між основними сутностями системи наведено на рисунку 2.5.

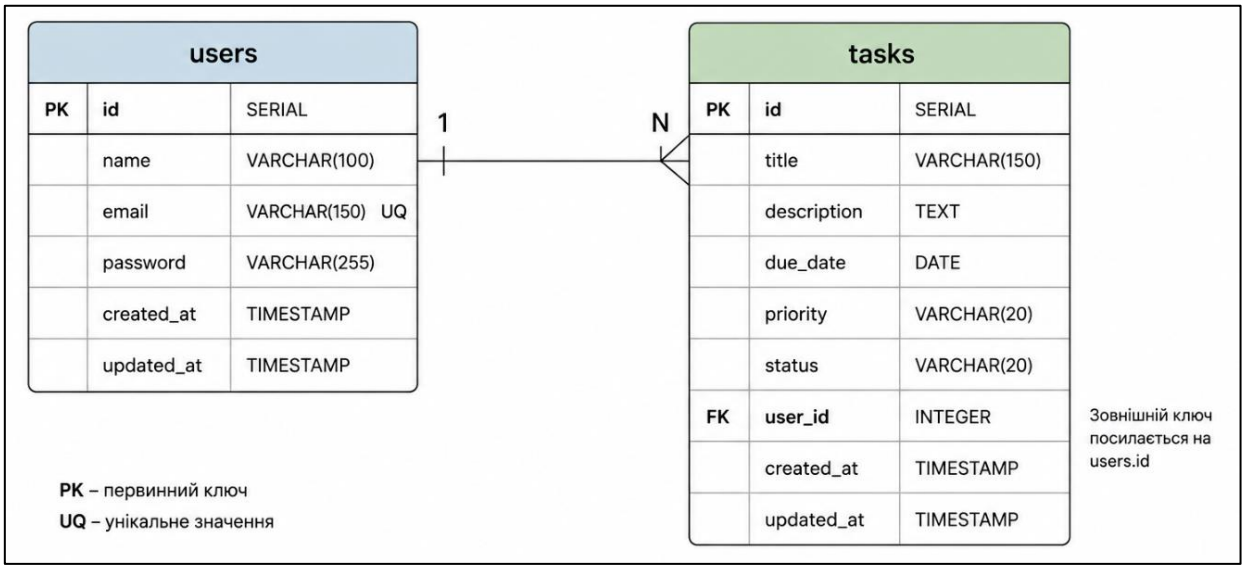


Рисунок 2.5 – ER-діаграма бази даних вебдодатку

Для забезпечення цілісності даних використовуються первинні та зовнішні ключі. Первинні ключі забезпечують унікальність записів у таблицях, а зовнішні ключі підтримують зв’язки між сутностями та запобігають виникненню неузгоджених даних. Крім того, для окремих полів застосовуються обмеження унікальності та перевірки коректності введених значень.

На етапі проєктування бази даних також передбачено можливість подальшого розширення функціональності системи. За необхідності до структури можуть бути додані нові таблиці для реалізації коментарів до задач, категорій, тегів, повідомлень або командної роботи над проєктами. Завдяки цьому система зберігає гнучкість та можливість адаптації до нових вимог користувачів.

Таким чином, розроблена структура бази даних забезпечує ефективне зберігання інформації про користувачів та задачі, підтримує цілісність даних і

створює основу для стабільної роботи вебдодатку планування та контролю виконання задач.

2.4 Моделювання функціональності системи

Після визначення архітектури програмного забезпечення та проєктування структури бази даних необхідно виконати моделювання функціональності системи. Даний етап дозволяє визначити основні сценарії взаємодії користувачів із вебдодатком, встановити функціональні межі системи та забезпечити більш ефективне проєктування програмного продукту.

Основною метою вебдодатку є забезпечення користувачів інструментами для створення, редагування та контролю виконання задач. Для реалізації цієї мети система повинна підтримувати набір функцій, які охоплюють усі основні етапи роботи із завданнями – від створення облікового запису до завершення виконання задачі та аналізу результатів роботи.

Головним учасником взаємодії із системою є користувач. Після проходження процедури реєстрації та автентифікації користувач отримує доступ до функціональних можливостей вебдодатку. Він може створювати нові задачі, переглядати наявні завдання, редагувати їх параметри, змінювати статус виконання та видаляти непотрібні записи. Крім того, користувач має можливість здійснювати пошук задач та застосовувати різні фільтри для швидкого отримання необхідної інформації.

Для наочного представлення взаємодії користувача із системою використовується UML-діаграма варіантів використання (Use Case Diagram) [10]. Вона відображає основні функціональні можливості вебдодатку та взаємозв’язки між користувачем і системою. Діаграму варіантів використання наведено на рисунку 2.6.

Як видно з рисунка, користувач має можливість виконувати основні операції із задачами, зокрема створення, редагування, видалення та зміну

статусу завдань, а також перегляд статистики та використання засобів пошуку і фільтрації. Більшість функцій системи передбачає проходження процедури автентифікації користувача.

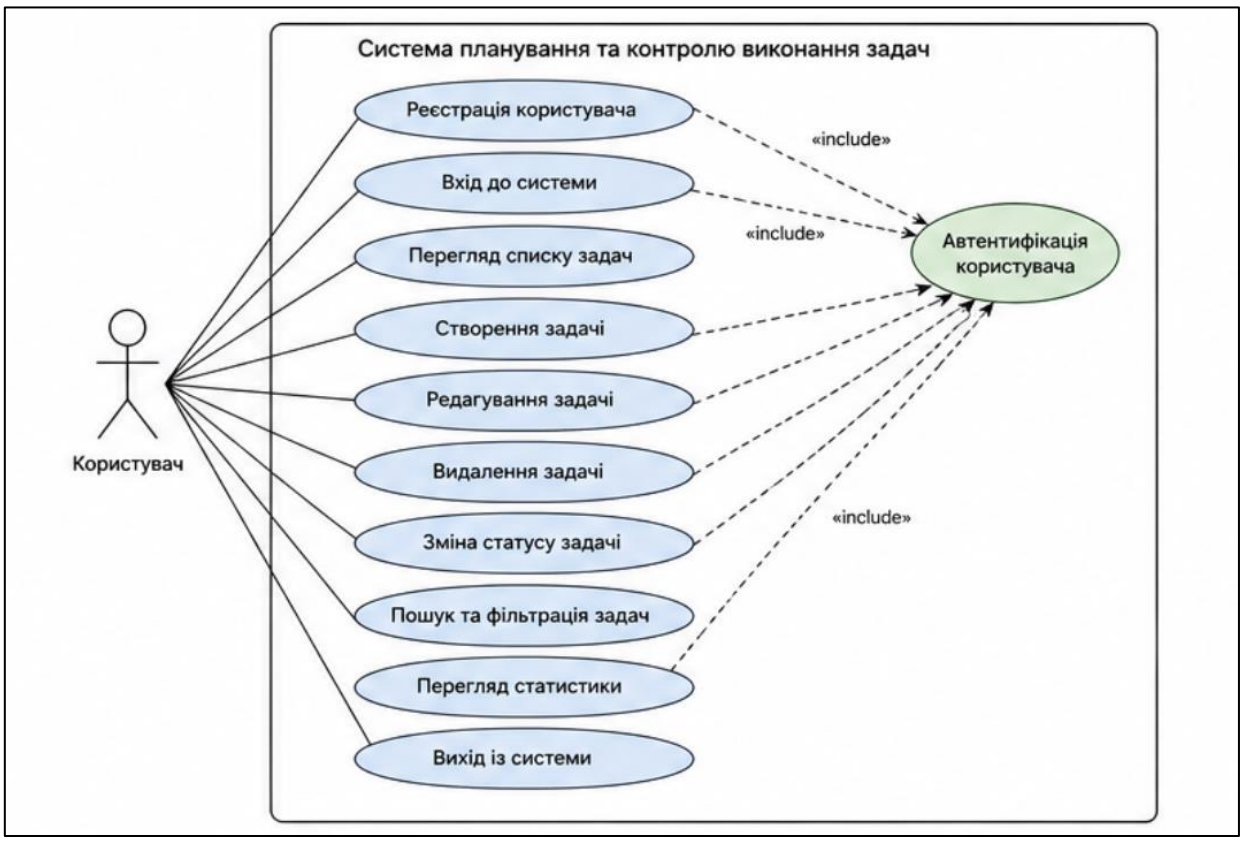


Рисунок 2.6 – UML-діаграма варіантів використання (Use Case Diagram)

Для детального опису процесу створення задачі використовується діаграма діяльності (Activity Diagram), яка демонструє послідовність виконання окремих операцій та логіку прийняття рішень у процесі роботи системи [10]. Діаграму діяльності наведено на рисунку 2.7.

З рисунка 2.7 видно, що процес створення задачі починається з відкриття форми введення даних.

Після заповнення необхідної інформації система виконує перевірку коректності введених даних.

У разі успішної перевірки задача зберігається у базі даних та відображається у списку завдань. Якщо введені дані містять помилки, користувач отримує відповідне повідомлення.

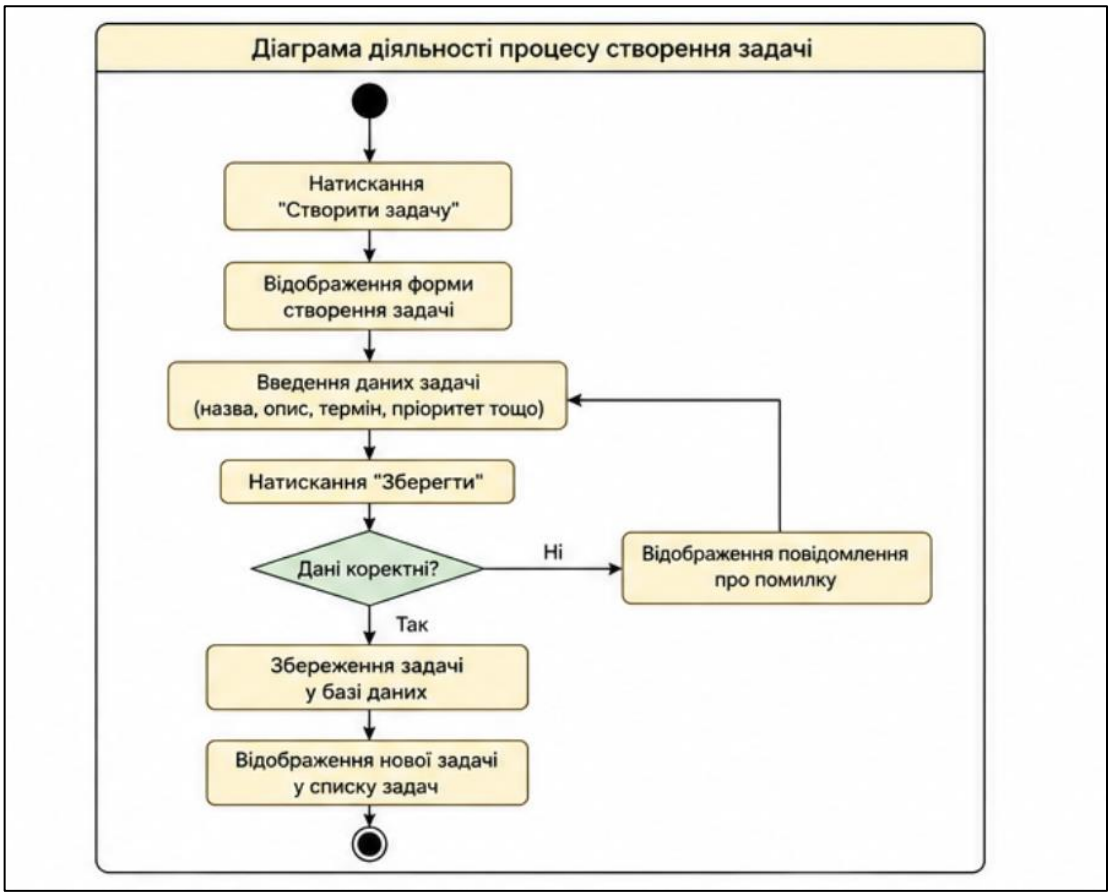


Рисунок 2.7 – Діаграма діяльності процесу створення задачі (Activity Diagram)

Для відображення взаємодії між компонентами системи під час створення задачі використовується діаграма послідовності (Sequence Diagram) [10]. Вона демонструє обмін повідомленнями між користувачем, клієнтською частиною, серверним API, сервісом обробки задач та базою даних. Діаграму послідовності наведено на рисунку 2.8.

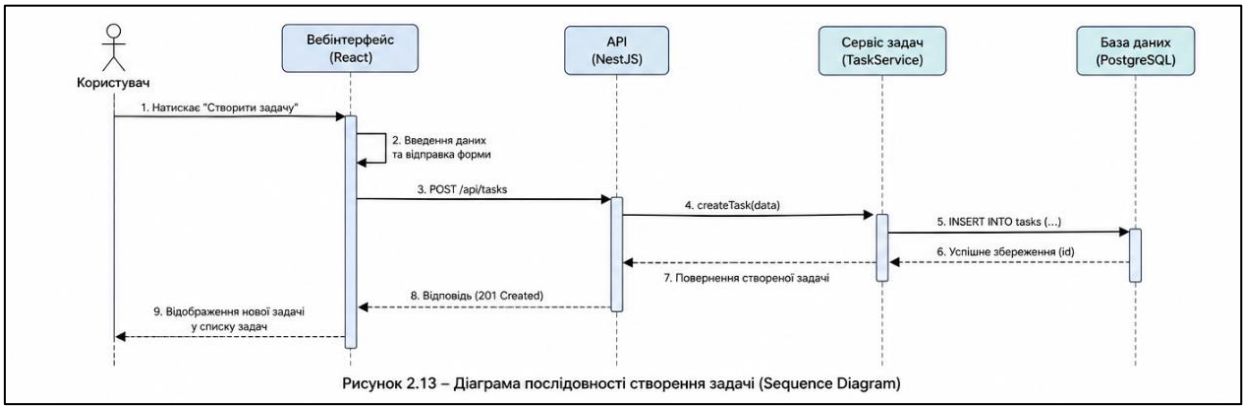


Рисунок 2.8 – Діаграма послідовності створення задачі (Sequence Diagram)

Як показано на рисунку 2.8, після введення даних користувачем клієнтська частина надсилає HTTP-запит до серверного API. Сервер виконує обробку запиту, передає дані до сервісу управління задачами та здійснює запис інформації до бази даних. Після успішного збереження система повертає відповідь клієнту, а новостворена задача відображається у користувацькому інтерфейсі.

Функціональна модель системи дозволяє визначити взаємозв’язки між користувачами, програмними модулями та базою даних. Вона виступає основою для подальшої реалізації програмного забезпечення та спрощує процес тестування функціональних можливостей вебдодатку.

Таким чином, моделювання функціональності системи дозволило визначити основні сценарії використання вебдодатку, сформувати перелік функцій, які повинні бути реалізовані, та підготувати основу для подальшої програмної реалізації системи планування та контролю виконання задач.

2.5 Проєктування користувацького інтерфейсу

Користувацький інтерфейс є важливою складовою будь-якої інформаційної системи, оскільки саме через нього здійснюється взаємодія користувача з програмним продуктом. Від якості проєктування інтерфейсу залежить зручність використання системи, швидкість виконання операцій та загальне враження користувачів від роботи з вебдодатком [29]. Тому під час розробки вебдодатку для планування та контролю виконання задач особлива увага приділялася створенню простого, зрозумілого та функціонального інтерфейсу.

Основною метою проєктування інтерфейсу є забезпечення швидкого доступу до основних функцій системи та мінімізація часу, необхідного для виконання типових операцій. Інтерфейс повинен бути інтуїтивно зрозумілим навіть для користувачів, які раніше не працювали з подібними системами. Для

досягнення цієї мети використовується принцип логічного групування елементів керування та послідовного розташування інформації на сторінках вебдодатку [29].

Під час проєктування було визначено основні сторінки системи. До них належать сторінка реєстрації користувача, сторінка авторизації, головна сторінка зі списком задач, сторінка створення та редагування завдань, а також сторінка статистики виконання задач. Кожна сторінка виконує окрему функцію та містить набір елементів, необхідних для реалізації відповідних можливостей системи.

Сторінка авторизації забезпечує вхід користувачів до системи шляхом введення адреси електронної пошти та пароля. Після успішної автентифікації користувач отримує доступ до основного функціоналу вебдодатку. Сторінка реєстрації дозволяє створити новий обліковий запис шляхом введення персональних даних та облікової інформації. Макет сторінки авторизації наведено на рисунку 2.9.

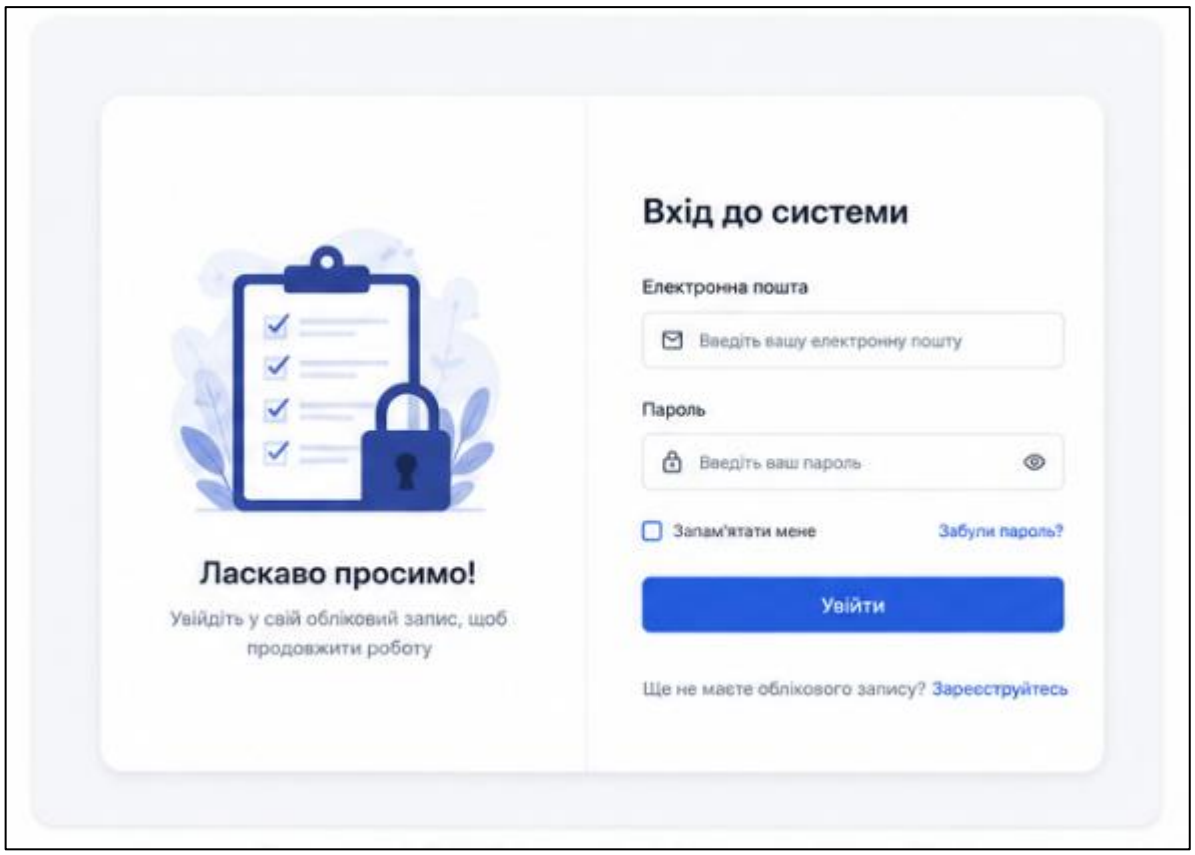


Рисунок 2.9 – Макет сторінки авторизації

Головна сторінка системи є центральним елементом інтерфейсу та призначена для перегляду списку задач. На ній відображаються назва завдання, його пріоритет, статус та термін виконання. Для підвищення зручності використання передбачено можливість сортування, пошуку та фільтрації задач за різними параметрами. Загальний вигляд головної сторінки зі списком задач представлено на рисунку 2.10.

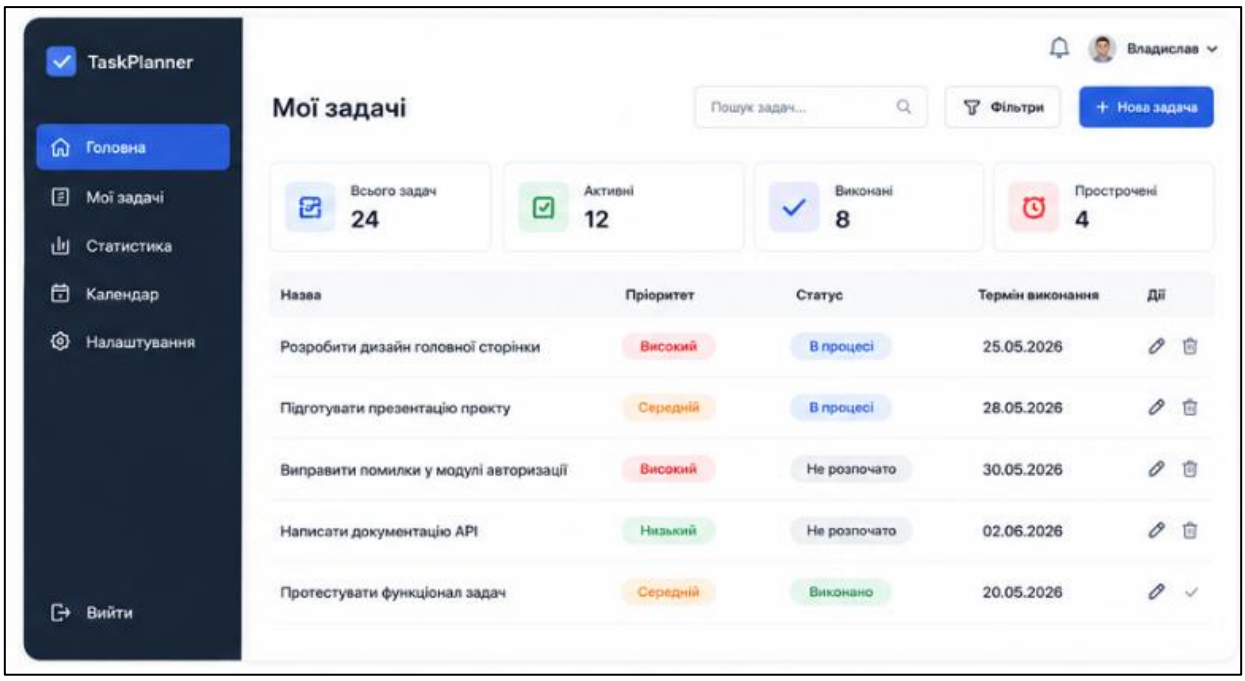


Рисунок 2.10 – Макет головної сторінки зі списком задач

Форма створення та редагування задач забезпечує введення необхідної інформації про завдання. Користувач має можливість вказати назву задачі, її опис, пріоритет та кінцевий термін виконання. Після підтвердження введених даних система виконує їх перевірку та зберігає інформацію у базі даних. Макет форми створення задачі наведено на рисунку 2.11.

Для візуального відображення результатів роботи користувача передбачено сторінку статистики, на якій відображається інформація про кількість виконаних, активних та прострочених задач. Використання графічних елементів дозволяє спростити аналіз поточного стану виконання завдань та підвищити наочність подання інформації. Приклад реалізації сторінки статистики виконання задач представлено на рисунку 2.12.

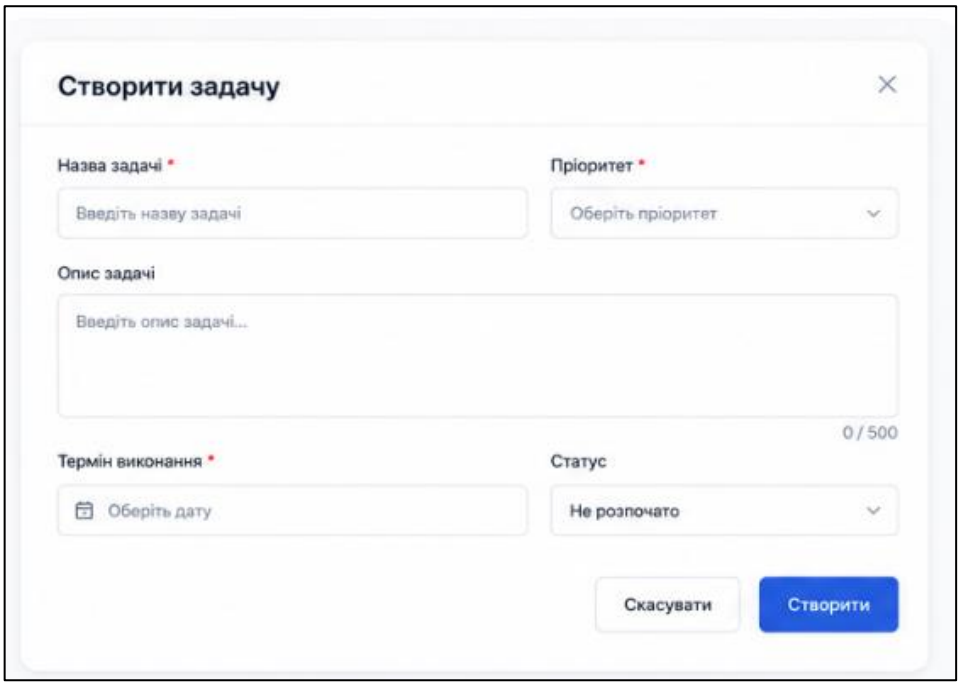


Рисунок 2.11 – Макет форми створення задачі

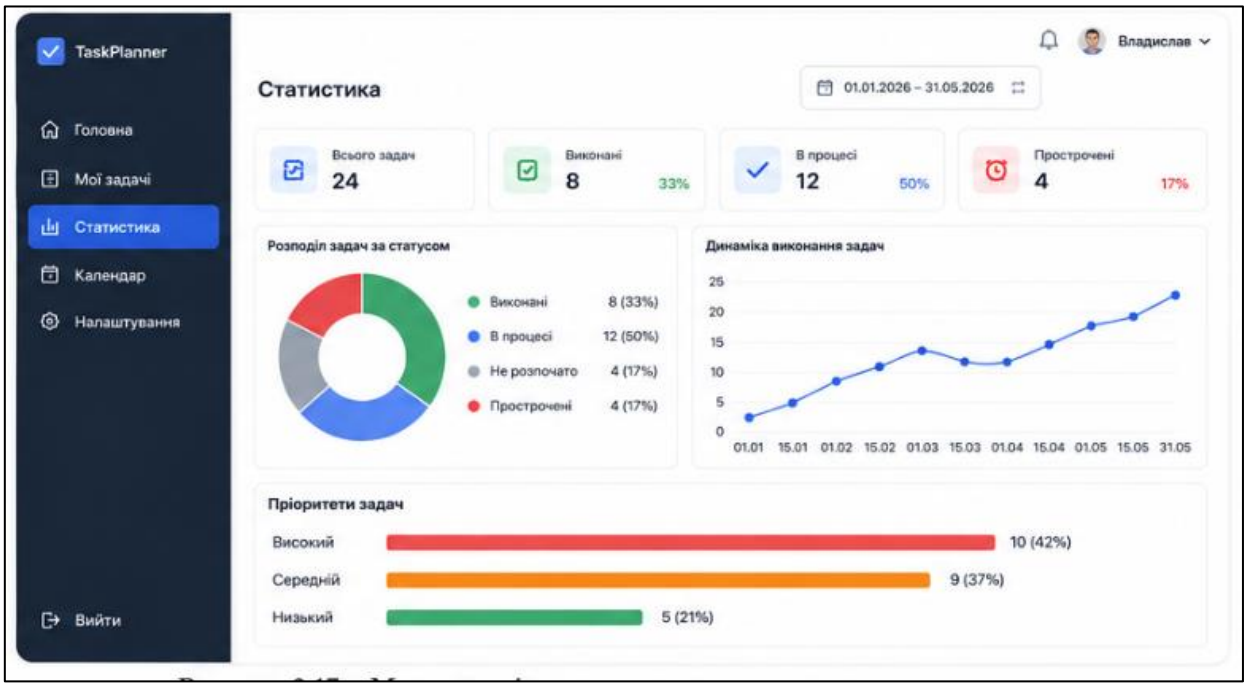


Рисунок 2.12 – Макет сторінки статистики виконання задач

Як видно з рисунків 2.9 – 2.12, інтерфейс вебдодатку побудований за принципами простоти, зручності та логічної організації елементів керування. Користувач отримує швидкий доступ до основних функцій системи, а структура сторінок забезпечує ефективне виконання типових операцій із задачами та перегляд статистичної інформації.

Під час розробки інтерфейсу враховувалися сучасні принципи UX/UI-дизайну, серед яких простота навігації, адаптивність для різних розмірів екранів, єдиний стиль оформлення та мінімізація кількості дій, необхідних для виконання основних операцій [27, 29]. Це дозволяє забезпечити комфортну роботу користувачів як на персональних комп’ютерах, так і на мобільних пристроях.

Для наочного представлення структури інтерфейсу були розроблені макети основних сторінок системи. Вони дозволяють оцінити розташування елементів керування, логіку переходів між сторінками та загальний вигляд вебдодатку ще до початку його програмної реалізації.

Таким чином, проєктування користувацького інтерфейсу дозволило сформувати зручну та зрозумілу структуру вебдодатку, яка забезпечує ефективну взаємодію користувачів із системою та сприяє підвищенню продуктивності роботи під час планування та контролю виконання задач.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ВЕБДОДАТКУ ДЛЯ ПЛАНУВАННЯ ТА КОНТРОЛЮ ВИКОНАННЯ ЗАДАЧ

3.1 Реалізація серверної частини вебдодатку

Серверна частина вебдодатку є одним із ключових компонентів системи, оскільки відповідає за обробку запитів користувачів, реалізацію бізнес-логіки, взаємодію з базою даних та забезпечення безпеки доступу до інформації. Під час розробки вебдодатку для планування та контролю виконання задач серверну частину було реалізовано з використанням фреймворку NestJS [20], який працює на платформі Node.js [23] та підтримує використання мови програмування TypeScript.

Вибір NestJS обумовлений його модульною архітектурою, високою продуктивністю та можливістю створення масштабованих серверних застосунків. Фреймворк дозволяє організувати програмний код у вигляді незалежних модулів, що спрощує розробку, тестування та подальший супровід програмного забезпечення.

Архітектура серверної частини побудована відповідно до принципу розділення відповідальностей. Для кожного функціонального модуля створено окремі контролери, сервіси та моделі даних. Контролери відповідають за обробку HTTP-запитів і формування відповідей клієнтам, сервіси реалізують бізнес-логіку системи, а моделі даних забезпечують взаємодію з базою даних.

Основними модулями серверної частини є модуль користувачів, модуль автентифікації та модуль управління задачами. Модуль користувачів забезпечує реєстрацію нових облікових записів та отримання інформації про користувачів системи. Модуль автентифікації відповідає за перевірку облікових даних користувачів та генерацію JWT-токенів доступу. Модуль управління задачами реалізує функції створення, редагування, перегляду та

видалення задач. Структуру основних модулів серверної частини вебдодатку наведено на рисунку 3.1.

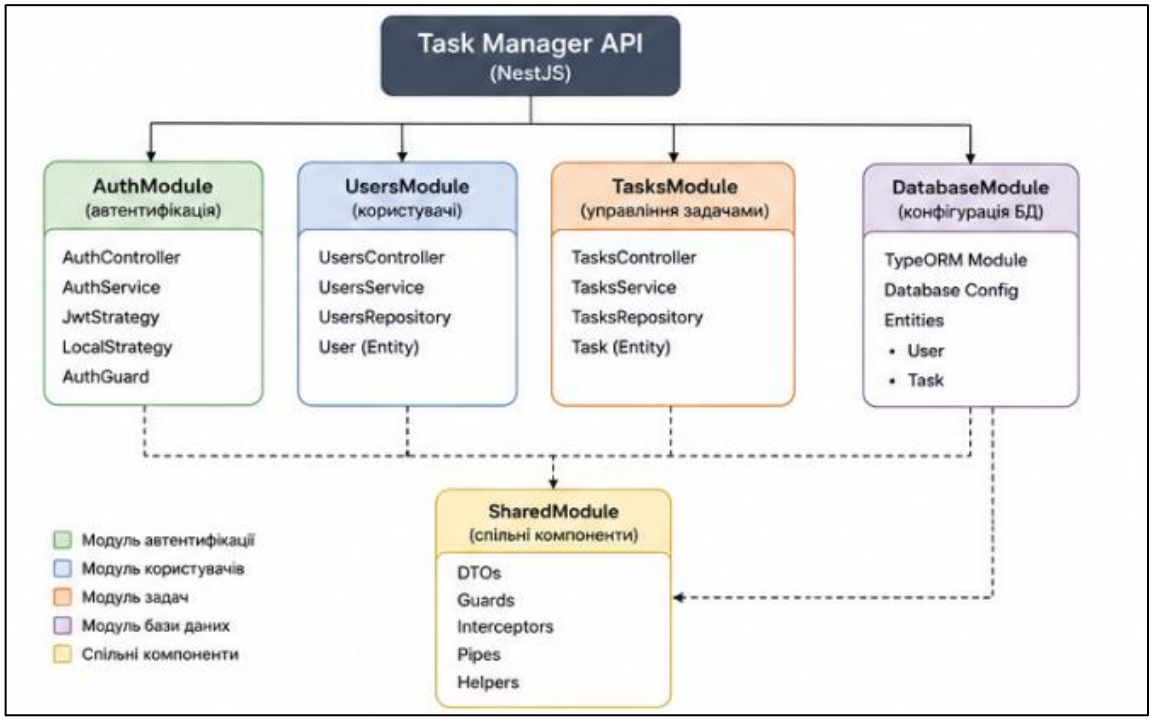


Рисунок 3.1 – Структура серверних модулів NestJS

Як видно з рисунка 3.1, серверна частина побудована за модульним принципом та складається з незалежних функціональних модулів, що спрощує підтримку та подальше розширення системи.

Взаємодія між клієнтською та серверною частинами здійснюється через REST API [25, 26]. Для кожної функціональної можливості системи реалізовано відповідні HTTP-методи. Метод POST використовується для створення нових записів, GET – для отримання даних, PUT або PATCH – для оновлення інформації, а DELETE – для видалення записів із бази даних. Схему взаємодії клієнтської частини, REST API та бази даних наведено на рисунку 3.2.

З рисунка 3.2 видно, що всі запити користувачів проходять через REST API, де виконується перевірка прав доступу, обробка бізнес-логіки та взаємодія з базою даних. Реалізацію контролера для роботи із задачами наведено в рисунку 3.3.

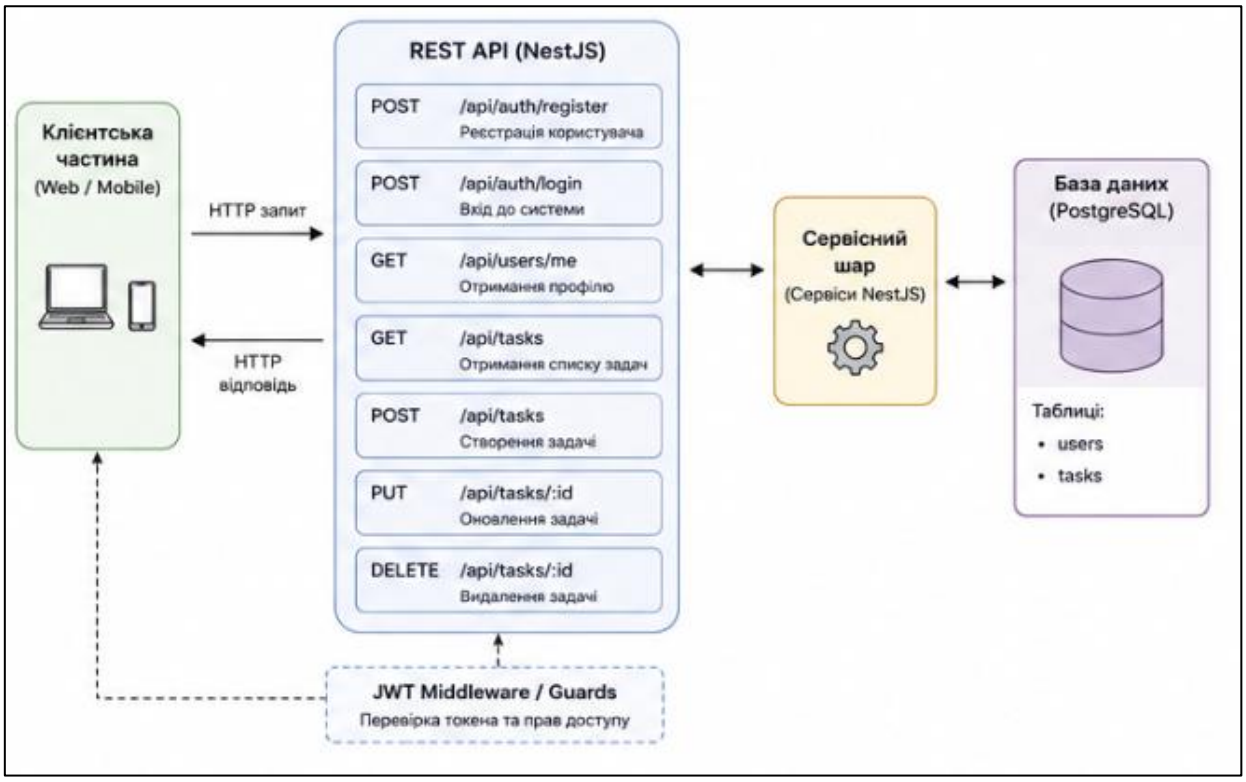


Рисунок 3.2 – Схема взаємодії REST API та бази даних

```
import { Controller, Get, Post, Body, Param, Put, Delete, UseGuards } from '@nestjs/common';
import { TasksService } from './tasks.service';
import { CreateTaskDto, UpdateTaskDto } from './dto';
import { JwtAuthGuard } from '../auth/jwt-auth.guard';

@Controller('api/tasks')
@UseGuards(JwtAuthGuard)
export class TasksController {
  constructor(private readonly tasksService: TasksService) {}

  @Get()
  findAll() {
    return this.tasksService.findAll();
  }

  @Post()
  create(@Body() createTaskDto: CreateTaskDto) {
    return this.tasksService.create(createTaskDto);
  }

  @Put(':id')
  update(@Param('id') id: string, @Body() updateTaskDto: UpdateTaskDto) {
    return this.tasksService.update(id, updateTaskDto);
  }

  @Delete(':id')
  remove(@Param('id') id: string) {
    return this.tasksService.remove(id);
  }
}
```

Рисунок 3.3 – контролер TasksController

Для забезпечення доступу до даних використовується ORM-засіб, який дозволяє працювати з базою даних за допомогою об’єктної моделі замість написання складних SQL-запитів. Такий підхід підвищує продуктивність розробки та зменшує ймовірність виникнення помилок під час роботи з даними. Реалізацію сервісу для управління задачами наведено в рисунку 3.4.

```ts
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typorrm';
import { Task } from './task.entity';
import { CreateTaskDto, UpdateTaskDto } from './dto';

@Injectable()
export class TasksService {
  constructor(
    @InjectRepository(Task)
    private readonly taskRepository: Repository<Task>,
  ) {}

  findAll(): Promise<Task[]> {
    return this.taskRepository.find({ order: { created_at: 'DESC' } });
  }
  async create(createTaskDto: CreateTaskDto): Promise<Task> {
    const task = this.taskRepository.create(createTaskDto);
    return this.taskRepository.save(task);
  }

  async update(id: string, updateTaskDto: UpdateTaskDto): Promise<Task> {
    const task = await this.taskRepository.findOne({ where: { id } });
    if (!task) throw new NotFoundException('Задача не знайдена');
    Object.assign(task, updateTaskDto);
    return this.taskRepository.save(task);
  }

  async remove(id: string): Promise<void> {
    const result = await this.taskRepository.delete(id);
    if (result.affected === 0) throw new NotFoundException('Задача не знайдена');
  }
}
```

Рисунок 3.4 – сервіс TasksService

Особлива увага приділялася забезпеченню безпеки системи. Паролі користувачів не зберігаються у відкритому вигляді, а проходять процедуру хешування перед записом до бази даних. Для контролю доступу до захищених ресурсів використовуються JWT-токени, які передаються у заголовках HTTP-запитів та перевіряються сервером під час кожного звернення до API.

У результаті реалізації серверної частини було створено надійну основу для роботи вебдодатку, яка забезпечує ефективну обробку даних, безпечний доступ користувачів до функціоналу системи та стабільну взаємодію з базою даних.

3.2 Реалізація клієнтської частини вебдодатку

Клієнтська частина вебдодатку забезпечує взаємодію користувача із системою та відповідає за відображення інформації, введення даних і надсилання запитів до серверної частини. Саме клієнтський інтерфейс формує враження користувача про програмний продукт, тому під час його реалізації

особлива увага приділяється зручності використання, швидкодії та адаптивності.

Для реалізації клієнтської частини вебдодатку було використано бібліотеку React [19] із підтримкою мови програмування TypeScript. Використання React дозволяє створювати сучасні односторінкові застосунки, які забезпечують швидке оновлення даних без необхідності повного перезавантаження сторінки. Застосування TypeScript сприяє підвищенню якості програмного коду завдяки використанню статичної типізації та додаткових механізмів перевірки помилок на етапі розробки.

Архітектура клієнтської частини побудована на основі компонентного підходу. Кожен елемент інтерфейсу реалізується у вигляді окремого компонента, що забезпечує повторне використання програмного коду та спрощує подальшу підтримку системи. До основних компонентів належать сторінки авторизації та реєстрації, список задач, форма створення задачі, сторінка статистики та навігаційне меню.

Для організації переходів між сторінками використовується механізм маршрутизації. Це дозволяє користувачу швидко переміщуватися між різними розділами системи без перезавантаження вебсторінки. Завдяки цьому підвищується зручність використання вебдодатку та покращується загальна продуктивність інтерфейсу.

Важливою складовою клієнтської частини є взаємодія із серверним API. Для отримання та оновлення інформації використовуються HTTP-запити, які надсилаються до серверної частини системи. Після отримання відповіді від сервера інтерфейс автоматично оновлюється та відображає актуальні дані користувачу.

Особлива увага приділялася реалізації сторінки управління задачами. Користувач має можливість переглядати список створених завдань, виконувати пошук, застосовувати фільтри за статусом або пріоритетом, а також створювати нові задачі чи редагувати вже існуючі. Такий підхід дозволяє ефективно організувати процес роботи із завданнями та забезпечує швидкий доступ до необхідної інформації.

Для підвищення зручності використання інтерфейс адаптовано для роботи на різних типах пристроїв. Елементи сторінок автоматично змінюють своє розташування залежно від розміру екрана, що забезпечує комфортну роботу як на персональних комп’ютерах, так і на планшетах та мобільних пристроях.

Під час реалізації клієнтської частини використовувалися сучасні принципи UX/UI-дизайну. Всі елементи інтерфейсу мають єдиний стиль оформлення, зрозумілі назви та логічне розташування. Для інформування користувача про результати виконання операцій застосовуються повідомлення про успішне виконання дій або виникнення помилок. Перед початком програмної реалізації було розроблено дизайн-макет інтерфейсу, який дозволив визначити розташування основних елементів сторінок та забезпечити єдиний стиль оформлення системи. Приклад макета інтерфейсу наведено на рисунку 3.5.

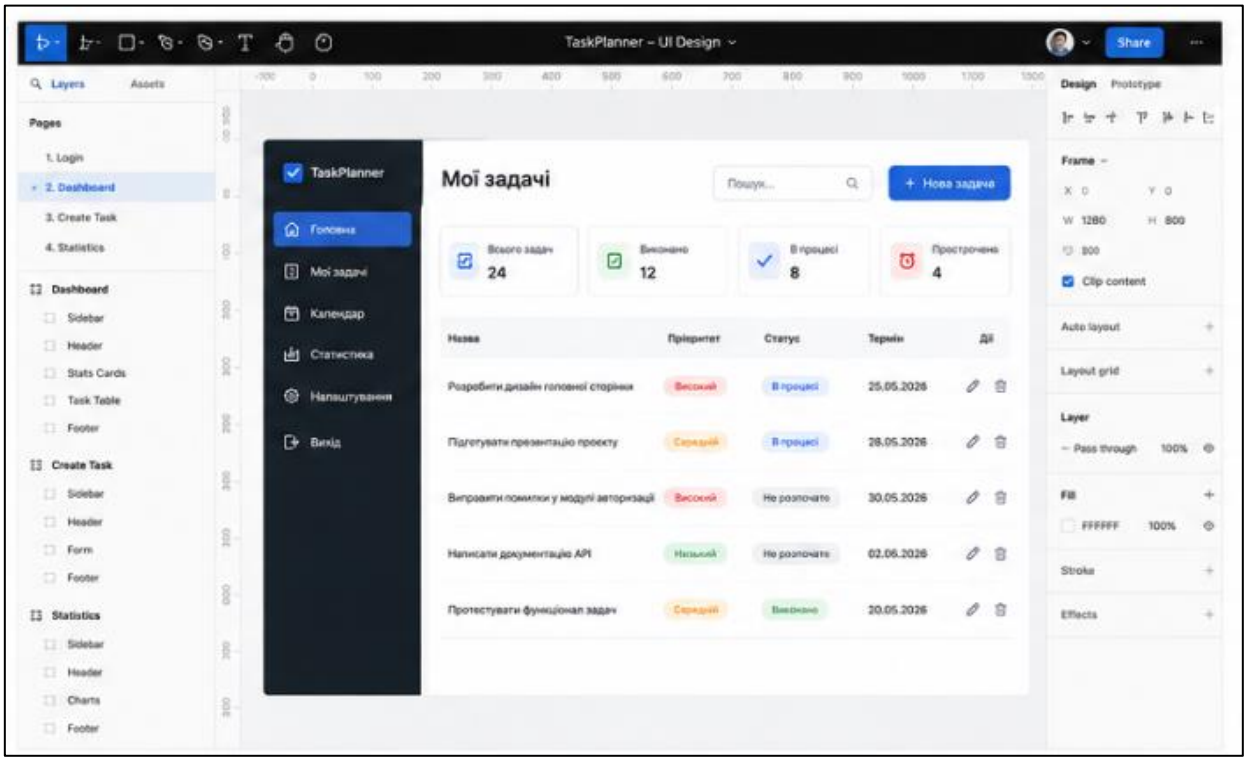


Рисунок 3.5 – Створення інтерфейсу (дизайн-макет у Figma)

Після завершення етапу проєктування було виконано програмну реалізацію користувацького інтерфейсу з використанням бібліотеки React та

мови програмування TypeScript. Реалізацію інтерфейсу у середовищі розробки Visual Studio Code наведено на рисунку 3.6.

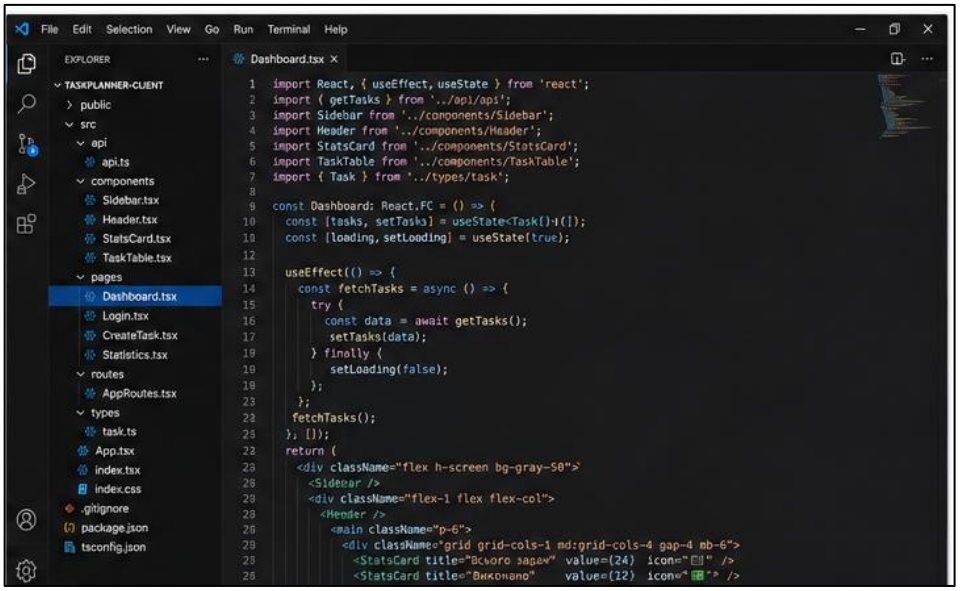


Рисунок 3.6 – Реалізація інтерфейсу (написання коду у VS Code)

Для зберігання даних автентифікації використовується JWT-токен, який отримується після успішного входу до системи. Токен зберігається на стороні клієнта та автоматично додається до запитів, що потребують авторизації користувача. Це забезпечує безпечну взаємодію між клієнтською та серверною частинами вебдодатку.

У результаті реалізації клієнтської частини було створено сучасний та зручний користувацький інтерфейс, який забезпечує швидкий доступ до функціональних можливостей системи, ефективну взаємодію з серверною частиною та комфортну роботу користувачів із задачами.

3.3 Реалізація механізмів автентифікації та авторизації

Одним із найважливіших аспектів розробки сучасних вебдодатків є забезпечення безпеки доступу до даних та функціональних можливостей системи. Для цього використовуються механізми автентифікації та

авторизації, які дозволяють перевіряти особу користувача та визначати його права доступу до ресурсів системи. У розробленому вебдодатку для планування та контролю виконання задач реалізовано безпечний механізм доступу на основі JWT-токенів [24].

Автентифікація є процесом перевірки особи користувача під час входу до системи. Для отримання доступу до функціональних можливостей вебдодатку користувач повинен пройти процедуру реєстрації та створити власний обліковий запис. Під час реєстрації користувач вводить необхідні дані, зокрема ім’я, адресу електронної пошти та пароль. Після перевірки коректності введеної інформації обліковий запис зберігається у базі даних.

Для забезпечення безпеки паролі користувачів не зберігаються у відкритому вигляді. Перед записом до бази даних вони проходять процедуру хешування за допомогою спеціалізованого алгоритму. Це дозволяє захистити облікові дані користувачів навіть у разі несанкціонованого доступу до бази даних.

Під час входу до системи користувач вводить адресу електронної пошти та пароль. Серверна частина виконує перевірку введених даних шляхом порівняння отриманого пароля із його хешованим значенням, що зберігається у базі даних. У разі успішної перевірки система генерує JWT-токен та надсилає його клієнтській частині. Послідовність виконання процесу автентифікації користувача із використанням JWT-токенів наведено на рисунку 3.7.

Як видно з рисунка 3.7, після введення облікових даних користувача сервер виконує перевірку інформації, звертається до бази даних та у разі успішної автентифікації генерує JWT-токен. Отриманий токен зберігається на стороні клієнта та використовується для подальших звернень до захищених ресурсів системи.

Отриманий JWT-токен використовується для підтвердження особи користувача під час подальших звернень до серверного API. Токен містить службову інформацію про користувача та підписується секретним ключем сервера. Це дозволяє перевіряти його достовірність без необхідності

зберігання додаткової інформації на сервері. Реалізацію контролера автентифікації наведено в рисунку 3.8.

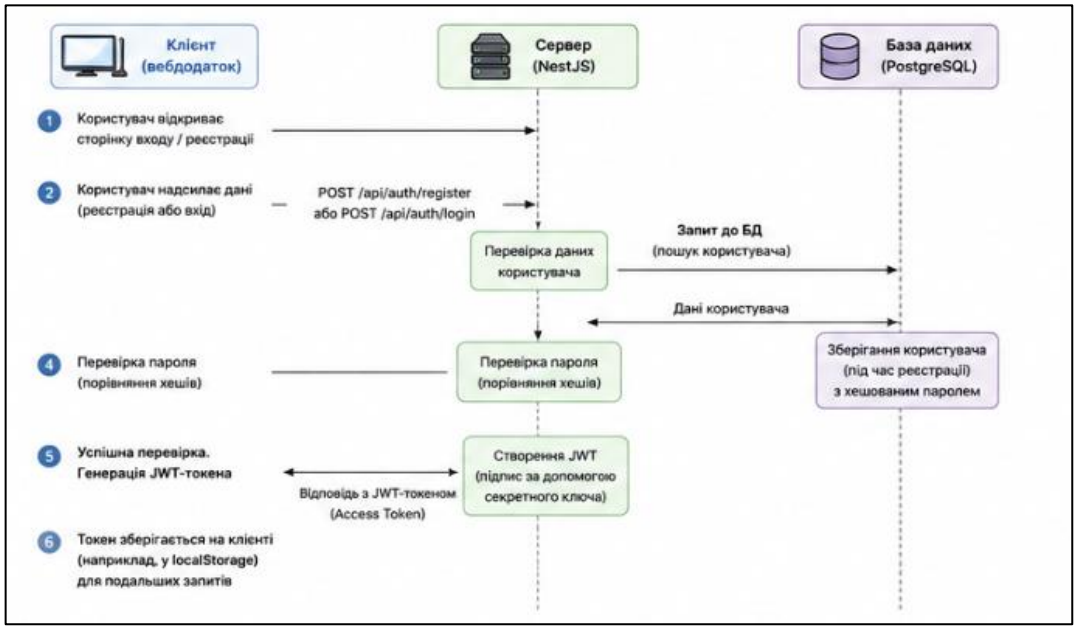


Рисунок 3.7 – Схема процесу автентифікації користувача з використанням JWT

```
import { Controller, Post, Body, HttpCode, HttpStatus } from '@nestjs/common';
import { AuthService } from './auth.service';
import { RegisterDto, LoginDto } from './dto/auth.dto';
import { AuthResponseDto } from './dto/auth-response.dto';

@Controller('api/auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('register')
  @HttpCode(HttpStatus.CREATED)
  async register(@Body() registerDto: RegisterDto): Promise<AuthResponseDto> {
    return this.authService.register(registerDto);
  }

  @Post('login')
  @HttpCode(HttpStatus.OK)
  async login(@Body() loginDto: LoginDto): Promise<AuthResponseDto> {
    return this.authService.login(loginDto);
  }
}

// AuthResponseDto містить: accessToken (JWT) та інформацію про користувача

```

Рисунок 3.8 – Реалізація AuthController

Контролер автентифікації відповідає за обробку запитів реєстрації та входу користувачів, а також формування відповідей із JWT-токенами доступу.

Авторизація є процесом перевірки прав доступу користувача до певних ресурсів або функцій системи. Після отримання JWT-токена він автоматично додається до заголовків HTTP-запитів. Під час обробки кожного запиту сервер перевіряє коректність токена та визначає, чи має користувач право виконувати запитувану операцію.

Для контролю доступу до захищених ресурсів у серверній частині використовуються спеціальні механізми захисту маршрутів. Вони дозволяють обмежити доступ до функціоналу системи лише для авторизованих користувачів.

У разі відсутності або недійсності токена сервер повертає повідомлення про помилку авторизації.

Процес авторизації та перевірки прав доступу користувача під час звернення до захищених ресурсів представлено на рисунку 3.9.

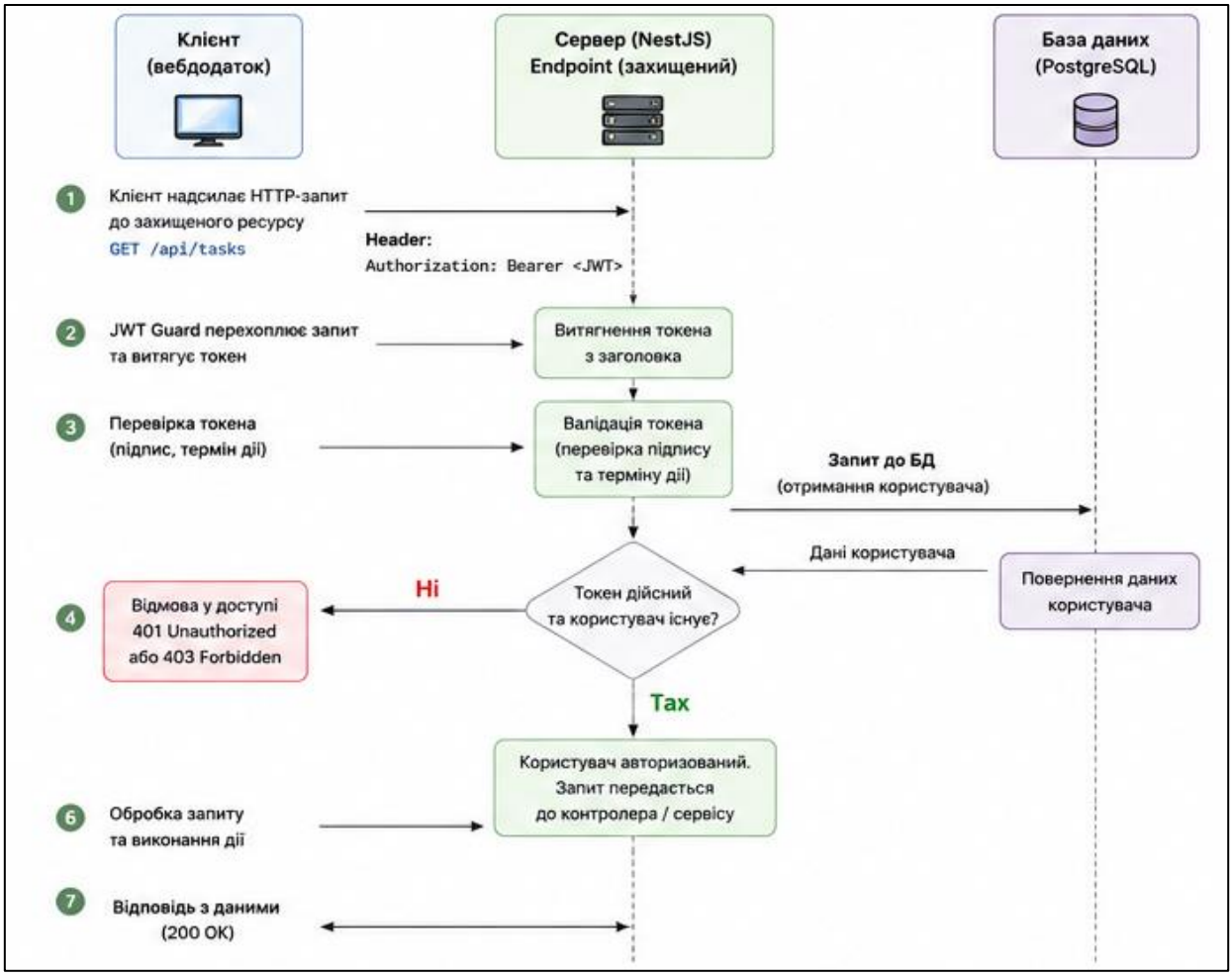


Рисунок 3.9 – Схема авторизації та перевірки прав доступу

З рисунка 3.9 видно, що кожен запит до захищеного ресурсу проходить перевірку JWT-токена. Лише після успішної валідації токена та підтвердження існування користувача система надає доступ до запитуваних функцій. Реалізацію механізму перевірки JWT-токенів наведено на рисунку 3.10.

```
import { Injectable, CanActivate, ExecutionContext, UnauthorizedException } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { Request } from 'express';

@Injectable()
export class JwtAuthGuard implements CanActivate {
  constructor(private readonly jwtService: JwtService) {}

  canActivate(context: ExecutionContext): boolean {
    const request = context.switchToHttp().getRequest<Request>();
    const authHeader = request.headers['authorization'];

    if (!authHaader || !authHeader.startsWith('Bearer ')) {
      throw new UnauthorizedException('Токен не надано');
    }

    const token = authHeader.split(' ')[1];

    try {
      const payload = this.jwtService.verify(token);
      request['user'] = payload; // зберігаємо дані користувача в request
      return true;
    } catch (error) {
      throw new UnauthorizedException('Недійсний або прострочений токен');
    }
  }
}
```

Рисунок 3.10 – Реалізація JWT Guard

Реалізований механізм автентифікації та авторизації забезпечує належний рівень захисту персональних даних користувачів і запобігає несанкціонованому доступу до функціональних можливостей системи. Використання JWT-токенів дозволяє створити безпечну та масштабовану систему керування доступом, яка відповідає сучасним вимогам до веборієнтованих інформаційних систем.

Таким чином, реалізовані механізми автентифікації та авторизації забезпечують безпечну взаємодію користувачів із вебдодатком та гарантують захист інформації під час роботи системи.

3.4 Реалізація функціоналу управління задачами

Основним призначенням розробленого вебдодатку є забезпечення ефективного управління задачами користувачів. Саме тому реалізація

функціоналу створення, редагування, видалення та контролю виконання завдань є центральною частиною системи. Розроблений функціонал дозволяє користувачам організовувати власну діяльність, відстежувати прогрес виконання робіт та своєчасно реагувати на зміни у робочому процесі.

Після успішної автентифікації користувач отримує доступ до списку власних задач. На головній сторінці системи відображаються всі створені завдання із зазначенням назви, пріоритету, статусу та терміну виконання. Для підвищення зручності роботи реалізовано можливість швидкого пошуку та фільтрації задач за різними параметрами.

Загальний вигляд реалізованої сторінки управління задачами наведено на рисунку 3.11.

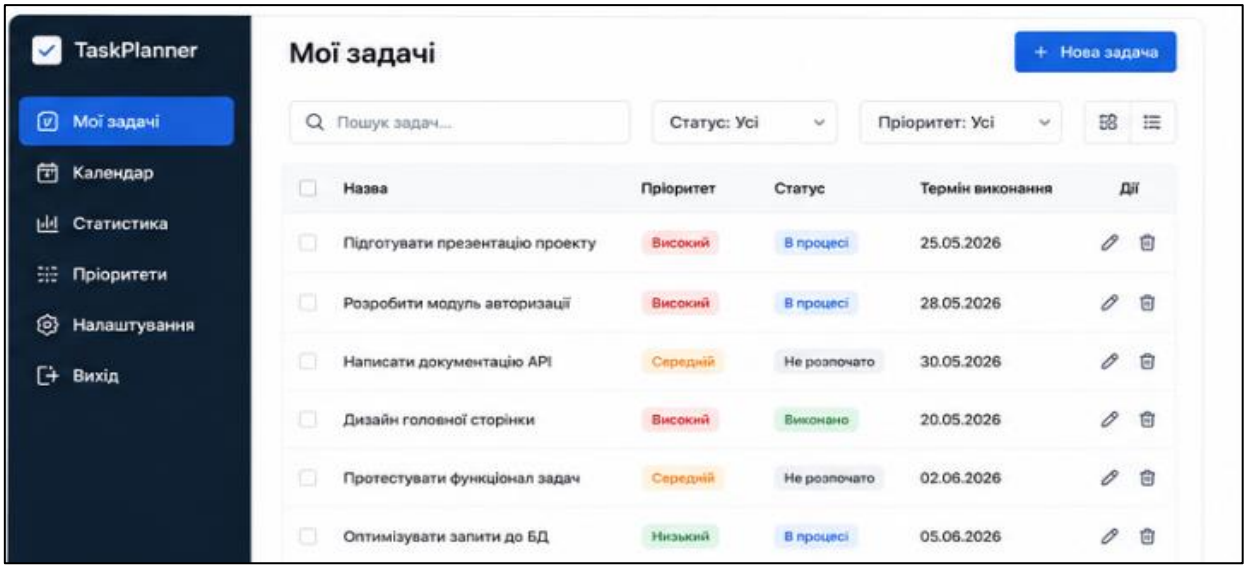


Рисунок 3.11 – Сторінка управління задачами

Як видно з рисунка 3.11, користувач має можливість переглядати список задач, використовувати пошук, фільтрацію за статусом та пріоритетом, а також виконувати операції редагування та видалення завдань.

Кожна задача містить набір основних атрибутів, необхідних для організації роботи. До них належать назва задачі, її опис, дата створення, термін виконання, пріоритет та поточний статус. Така структура дозволяє користувачеві отримувати всю необхідну інформацію про завдання без додаткових переходів між сторінками.

Для створення нової задачі користувач заповнює спеціальну форму, в якій вказує необхідні параметри майбутнього завдання. Після перевірки коректності введених даних інформація надсилається до серверної частини та зберігається у базі даних. Процес створення задачі реалізовано таким чином, щоб мінімізувати кількість дій користувача та забезпечити максимальну зручність використання системи. Реалізована форма створення нової задачі наведена на рисунку 3.12.

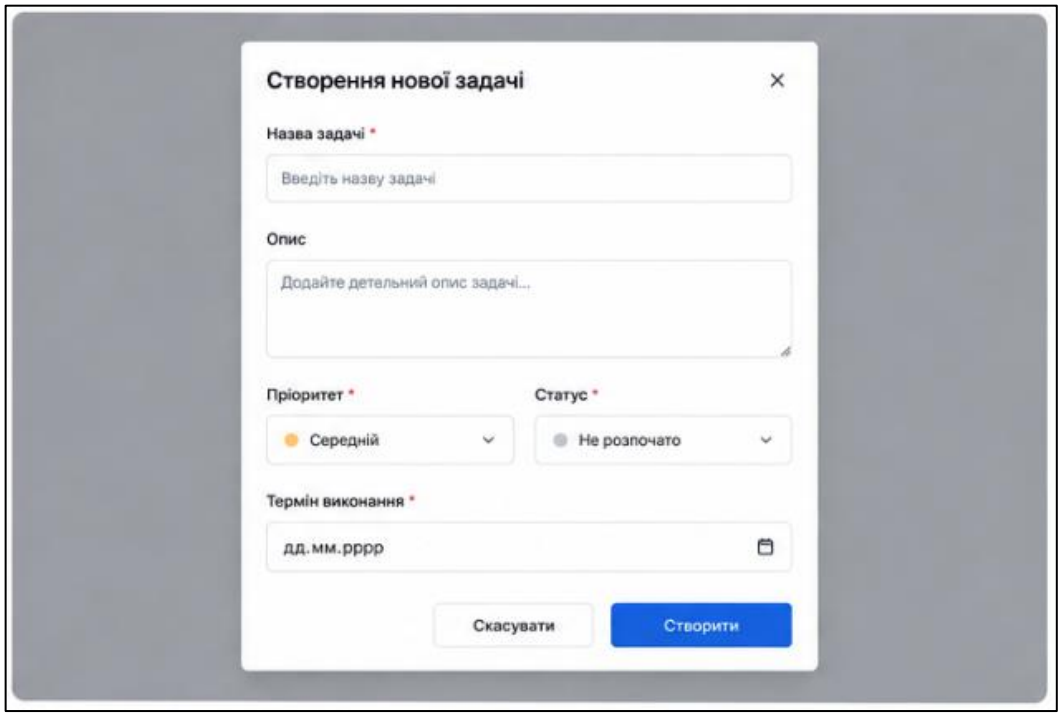


Рисунок 3.12 – Форма створення нової задачі

Форма дозволяє ввести назву задачі, її опис, встановити пріоритет, статус та термін виконання, що забезпечує повний опис завдання ще на етапі його створення.

Однією з важливих функцій системи є можливість редагування існуючих завдань. Користувач може змінювати назву задачі, її опис, пріоритет, термін виконання або поточний статус. Усі зміни автоматично зберігаються у базі даних та стають доступними після оновлення інформації на сторінці.

Крім редагування, система підтримує видалення задач. Перед видаленням користувач отримує запит на підтвердження дії, що дозволяє

уникнути випадкової втрати важливої інформації. Після підтвердження задача видаляється з бази даних та більше не відображається у списку завдань.

Для контролю виконання роботи реалізовано систему статусів задач. Користувач може змінювати стан завдання відповідно до етапу його виконання. У системі використовуються такі основні статуси: «Не розпочато», «В процесі виконання» та «Виконано». Завдяки цьому користувач може швидко оцінити поточний стан усіх завдань та визначити пріоритети подальшої роботи. Приклад процесу зміни статусу задачі наведено на рисунку 3.13.

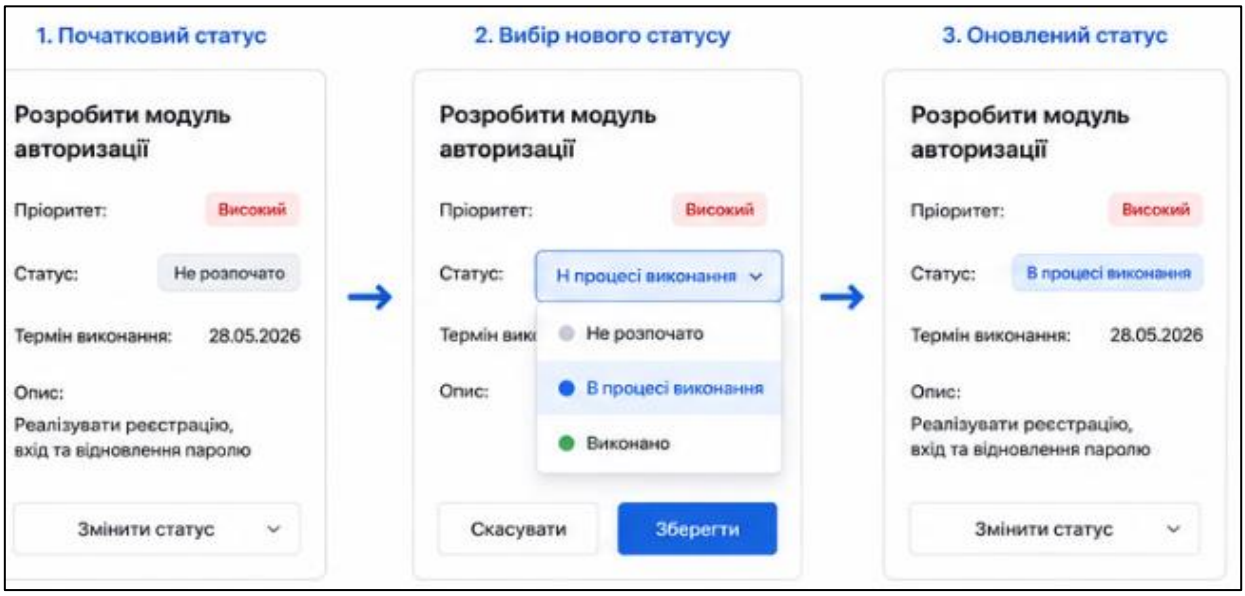


Рисунок 3.13 – Процес зміни статусу задачі

На рисунку 3.13 продемонстровано зміну статусу задачі від початкового стану до етапу виконання. Після збереження змін інформація автоматично оновлюється у базі даних та відображається в інтерфейсі користувача.

Для взаємодії клієнтської та серверної частин реалізовано набір REST API-запитів. За допомогою HTTP-методів GET, POST, PATCH та DELETE здійснюються операції отримання, створення, оновлення та видалення задач відповідно. Такий підхід забезпечує чітке розділення функціональності між компонентами системи та спрощує її подальше розширення. Схему виконання CRUD-операцій для задач наведено на рисунку 3.14.

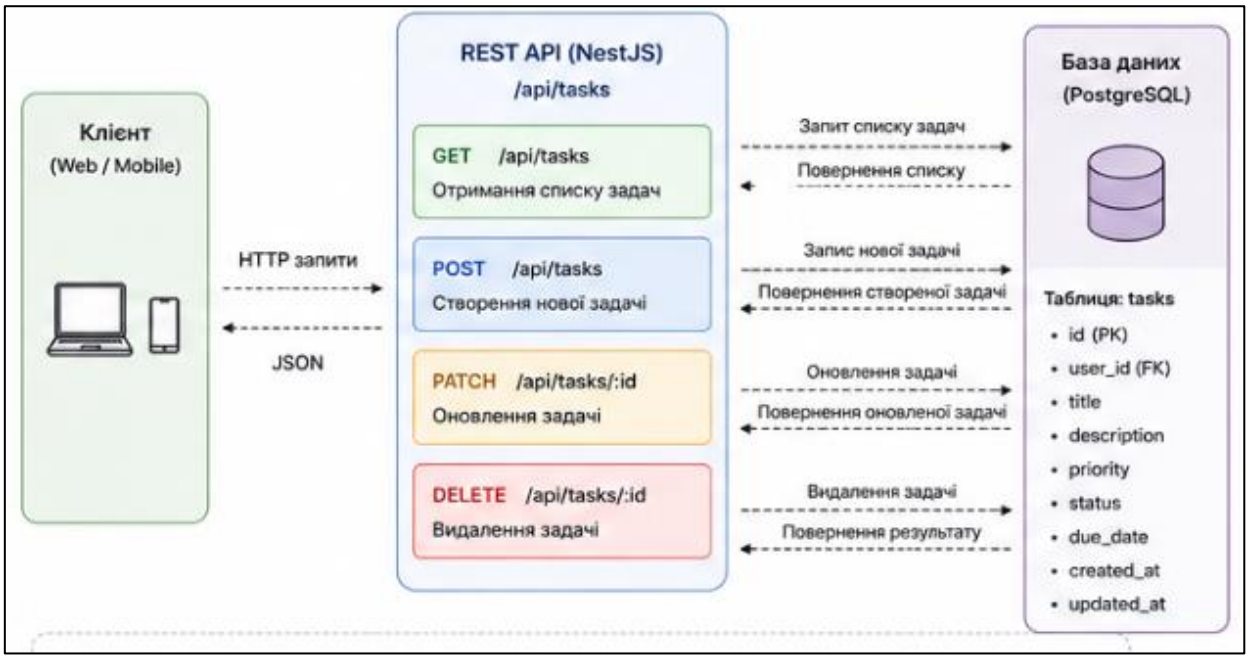


Рисунок 3.14 – Схема CRUD-операцій для задач

Як показано на рисунку 3.14, усі операції над задачами виконуються через REST API із використанням відповідних HTTP-методів. Це забезпечує стандартизований обмін даними між клієнтською частиною вебдодатку та сервером.

З метою підвищення продуктивності користувачів у системі реалізовано механізми сортування та фільтрації задач. Користувач може відображати лише завдання з певним статусом або рівнем пріоритету, що значно спрощує роботу з великими списками задач та дозволяє швидко знаходити необхідну інформацію.

Таким чином, реалізований функціонал управління задачами забезпечує повний цикл роботи із завданнями – від їх створення до завершення виконання [6, 7]. Використання сучасних вебтехнологій дозволило створити зручний та ефективний інструмент для організації роботи користувачів і контролю виконання поставлених завдань.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено вебдодаток для планування та контролю виконання задач, який забезпечує ефективну організацію робочого процесу, управління завданнями та моніторинг їх виконання. Досягнуто поставленої мети роботи, що полягала у створенні сучасної веборієнтованої інформаційної системи для підтримки процесів планування, розподілу та контролю виконання задач.

У першому розділі було проведено аналіз предметної області, досліджено особливості процесів планування та контролю виконання задач, а також розглянуто сучасні підходи до організації управління завданнями. Виконано огляд існуючих програмних продуктів, зокрема Trello, Asana, Jira та Microsoft Planner. На основі проведеного порівняльного аналізу визначено їх переваги та недоліки, а також сформовано функціональні та нефункціональні вимоги до розроблюваної системи.

У другому розділі виконано проєктування вебдодатку. Обґрунтовано вибір технологій розробки, серед яких React для реалізації клієнтської частини, NestJS для серверної частини та PostgreSQL для зберігання даних. Спроєктовано архітектуру системи, структуру бази даних та функціональну модель вебдодатку. Також розроблено макети користувацького інтерфейсу, що дозволили визначити структуру сторінок та логіку взаємодії користувача із системою.

У третьому розділі здійснено реалізацію програмного забезпечення відповідно до сформованих вимог. Розроблено серверну частину із використанням модульної архітектури NestJS та реалізовано REST API для взаємодії між клієнтською частиною та базою даних. Створено клієнтську частину вебдодатку на основі React, яка забезпечує зручний користувацький інтерфейс та швидку роботу системи. Реалізовано механізми автентифікації та авторизації користувачів із використанням JWT-токенів, що забезпечують належний рівень безпеки та захисту даних.

У межах розробки реалізовано основний функціонал управління задачами, який включає створення, редагування, видалення, пошук, фільтрацію та зміну статусу завдань. Також забезпечено можливість перегляду статистичної інформації щодо виконання задач, що дозволяє користувачам контролювати власну продуктивність та ефективніше організовувати робочий процес.

Розроблений вебдодаток має зручний та інтуїтивно зрозумілий інтерфейс, підтримує адаптивне відображення на різних типах пристроїв та забезпечує швидкий доступ до необхідної інформації. Використання сучасних технологій веброзробки дозволило створити масштабовану систему, яка може бути розширена додатковими функціональними можливостями у майбутньому.

Практичне значення отриманих результатів полягає у створенні програмного продукту, який може використовуватися для особистого планування, навчальної діяльності або організації роботи невеликих команд. Вебдодаток сприяє підвищенню ефективності управління завданнями, оптимізації робочого процесу та покращенню контролю за виконанням поставлених цілей.

ПЕРЕЛІК ПОСИЛАНЬ

1. Басюк Т. М., Жежнич П. І. Методи та засоби інженерії програмного забезпечення. Львів : Видавництво Львівської політехніки, 2018. 220 с.

2. Бондаренко М. Ф. Організація баз даних та знань. Харків : Компанія СМІТ, 2019. 384 с.

3. Грицюк Ю. І. Проєктування інформаційних систем. Львів : Новий Світ-2000, 2020. 410 с.

4. Згуровський М. З., Панкратова Н. Д. Основи системного аналізу : підручник. Київ : Видавнича група BHV, 2018. 544 с.

5. Кравченко В. М. Інформаційні системи та технології : навч. посіб. Київ : Каравела, 2021. 344 с.

6. Sommerville I. Software Engineering : 10th ed. Boston : Pearson, 2016. 816 p.

7. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach : 9th ed. New York : McGraw-Hill Education, 2019. 992 p.

8. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Boston : Prentice Hall, 2018. 432 p.

9. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley Professional, 2018. 560 p.

10. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.

11. Date C. J. An Introduction to Database Systems : 8th ed. Boston : Pearson Education, 2019. 1024 p.

12. Elmasri R., Navathe S. B. Fundamentals of Database Systems : 7th ed. Boston : Pearson, 2017. 1272 p.

13. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts : 7th ed. New York : McGraw-Hill Education, 2019. 1376 p.

14. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. 560 p.

15. Richards M., Ford N. Fundamentals of Software Architecture. Sebastopol : O’Reilly Media, 2020. 432 p.

16. Nielsen J. Usability Engineering. San Diego : Academic Press, 2012. 362 p.

17. Freeman E., Robson E. Head First Design Patterns : 2nd ed. Sebastopol : O’Reilly Media, 2020. 694 p.

18. ISO/IEC 25010:2011. Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models.

19. React Documentation. URL: https://react.dev/ (дата звернення: 12.01.2026).

20. NestJS Documentation. URL: https://docs.nestjs.com/ (дата звернення: 12.01.2026).

21. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 12.01.2026).

22. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 14.02.2026).

23. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 14.02.2026).

24. JSON Web Tokens. Introduction to JWT. URL: https://jwt.io/introduction (дата звернення: 15.02.2026).

25. Mozilla Developer Network. HTTP Overview. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення: 16.02.2026).

26. Mozilla Developer Network. REST APIs. URL: https://developer.mozilla.org/en-US/docs/Glossary/REST (дата звернення: 23.03.2026).

27. Figma Help Center. URL: https://help.figma.com/ (дата звернення: 23.03.2026).

28. HTML Living Standard. URL: https://html.spec.whatwg.org/ (дата звернення: 20.04.2026).

29. Cascading Style Sheets (CSS). URL: https://www.w3.org/Style/CSS/ (дата звернення: 20.04.2026).

30. Oracle Database Documentation. URL: https://docs.oracle.com/ (дата звернення: 10.05.2026).