МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБДОДАТКУ З ПЛАНУВАННЯ ТА ОПТИМІЗАЦІЇ НАВЧАЛЬНОГО ПРОЦЕСУ»**

Виконала: студентка 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
С.П. Ємеліна
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Мильцев О.М.
(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 – 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 СТУДЕНТЦІ

Ємеліній Софії Павлівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебдодатку з планування та оптимізації навчального процесу

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н.
(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 існуючих рішень.
2. Обґрунтування вибору засобів реалізації.
3. Проєктування та реалізація системи.
4. Реалізація та тестування web-застосунку Study Planner.

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.	28.01.2026	
3.	Обробка методичних та теоретичних	20.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	08.04.2026	
5.	Розробка третього та четвертого розділу.	15.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2026	

Студент _____ С.П. Ємеліна
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебдодатку з планування та оптимізації навчального процесу»: 62 с., 15 рис., 19 табл., 19 джерел.

АВТОРИЗАЦІЯ, ВЕБДОДАТОК, ДЕДЛАЙН, ЗАДАЧА, КАЛЕНДАР, НАВЧАЛЬНЕ ПЛАНУВАННЯ, POSTGRESQL, REACT, REST API, STUDY PLANNER.

Об’єкт дослідження – процес організації навчальної діяльності студента.

Мета роботи: розробити клієнт-серверний вебдодаток Study Planner для планування навчальних задач, дедлайнів і календарних подій студента.

Методи дослідження – аналіз предметної області, порівняння програмних рішень і технологій, проєктування системи, програмна реалізація та тестування.

Кваліфікаційну роботу присвячено розробці вебдодатку Study Planner для особистого планування навчальної діяльності студента. У роботі визначено вимоги до системи, спроєктовано архітектуру, модель даних, основні сценарії використання та інтерфейс. Клієнтську частину реалізовано на React, серверну частину – на Node.js та Express.js, базу даних – на PostgreSQL. Взаємодія між частинами системи виконується через REST API.

Розроблений вебдодаток дозволяє створювати й редагувати навчальні задачі, встановлювати дедлайни, додавати календарні події, використовувати пошук, фільтрацію, статистику та персональні налаштування. Тестування підтвердило, що основні функції Study Planner працюють коректно.

SUMMARY

Bachelor's qualifying paper "Development of a Web Application for Planning and Optimizing the Educational Process": 62 pages, 15 figures, 19 tables, 19 references.

AUTHORIZATION, WEB APPLICATION, DEADLINE, TASK, CALENDAR, LEARNING PLANNING, POSTGRESQL, REACT, REST API, STUDY PLANNER.

The object of the study is the process of organizing a student's learning activity.

The aim of the study is to develop a client-server web application Study Planner for planning students' learning tasks, deadlines and calendar events.

The methods of research include subject area analysis, comparison of software solutions and technologies, system design, software implementation and testing.

The bachelor's qualifying paper is devoted to the development of the Study Planner web application for personal planning of a student's learning activity. The paper defines system requirements and describes the architecture, data model, main use cases and interface. The client side was implemented with React, the server side with Node.js and Express.js, and the database with PostgreSQL. The interaction between system parts is performed through REST API.

The developed web application allows users to create and edit learning tasks, set deadlines, add calendar events, use search, filtering, statistics and personal settings. Testing confirmed that the main functions of Study Planner work correctly.

ЗМІСТ

Завдання на кваліфікаційну роботу ……..…….2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз предметної області та існуючих рішень..... 10
1.1 Постановка задачі 10
1.2 Аналіз існуючих рішень 10
1.3 Результати проведеного аналізу 13
1.4 Висновки до розділу 1 16
2 Обґрунтування вибору засобів реалізації 17
2.1 Вибір технологій для клієнтської частини 17
2.1.1 HTML, CSS та JavaScript у клієнтській частині 18
2.1.2 React як основна бібліотека для побудови інтерфейсу 19
2.1.3 Порівняння використаних технологій з альтернативами 19
2.2 Вибір технологій для серверної частини 21
2.3 Вибір бази даних 23
2.4 Вибір засобів розгортання та організації середовища 24
2.5 Висновки до розділу 2 25
3 Проєктування та реалізація системи 27
3.1 Аналіз вимог до системи 27
3.2 Проєктування прецедентів використання 31
3.3 Проєктування моделі даних 34
3.4 Проєктування поведінки системи 38
3.5 Проєктування загальної структури та компонентів системи 40
3.6 Проєктування користувацького інтерфейсу 43
3.7 Висновки до розділу 3 44
4 Реалізація та тестування web-застосунку Study planner..... 46

4.1 Середовище розробки та використані технології 46
4.2 Реалізація серверної частини (Backend) 47
4.3 Реалізація клієнтської частини (Frontend) 49
4.4 Реалізація основного функціоналу системи 53
4.5 Тестування програмного забезпечення 54
4.6 Опис результатів тестування 55
4.7 Інструкція користувача 56
4.8 Висновки до розділу 4 57
Висновки 58
Перелік посилань 61

ВСТУП

Навчання у закладі вищої освіти пов’язане з постійною роботою із задачами, дедлайнами та календарними подіями. Протягом семестру студент отримує лабораторні, практичні та індивідуальні завдання, готується до контрольних робіт, консультацій, заліків і екзаменів. Кожна така активність має власний термін, складність і пріоритет, тому для нормального навчального процесу важливо не тільки виконувати завдання, а й бачити їх загальну картину.

На практиці навчальна інформація часто зберігається в різних місцях. Частина завдань залишається в повідомленнях, частина записується в нотатках, дедлайни можуть бути внесені в календар, а окремі матеріали зберігаються у файлах. Коли таких джерел стає багато, студенту складно швидко зрозуміти, що потрібно зробити сьогодні, які задачі наближаються до дедлайну та як розподілене навантаження на тиждень.

Актуальність теми полягає в потребі створення простого інструменту для особистого навчального планування. Універсальні сервіси для календаря або задач корисні, але вони не завжди зручно поєднують навчальні задачі, дедлайни, статуси, теги, статистику та персональні налаштування. Студенту потрібен застосунок, який не перевантажує інтерфейс і дозволяє швидко працювати саме з навчальними справами.

Метою роботи є розроблення клієнт-серверного вебзастосунку Study Planner для особистого планування навчальної діяльності студента.

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

– проаналізувати предметну область;
– розглянути існуючі рішення;
– обґрунтувати вибір технологій;
– спроєктувати архітектуру системи та базу даних;
– реалізувати основні функціональні модулі;
– протестувати основний функціонал системи.

Об’єктом дослідження є процес організації навчальної діяльності студента.

Предметом дослідження є методи та програмні засоби створення вебзастосунку для планування навчальних задач, дедлайнів і календарних подій.

Практичне значення роботи полягає в тому, що Study Planner може використовуватися студентом як персональний інструмент для організації навчального процесу. Застосунок допомагає зберігати задачі, бачити дедлайни, додавати календарні події, контролювати статуси, використовувати теги, фільтрувати список і переглядати статистику виконання.

Клієнтська частина Study Planner реалізована за допомогою React. Серверна частина створена на Node.js з використанням Express.js. Для збереження даних використовується PostgreSQL, а взаємодія між клієнтом і сервером відбувається через REST API. Доступ до персональних даних користувача надається тільки після авторизації.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків та списку використаних джерел. Загальний обсяг роботи становить 62 сторінки. Робота містить 15 рисунків, 19 таблиць і 19 використаних джерел.

У першому розділі проаналізовано предметну область та існуючі рішення для планування задач і календарних подій. У другому розділі обґрунтовано вибір засобів реалізації вебзастосунку Study Planner. У третьому розділі описано проєктування системи, модель даних, основні сценарії використання, поведінку системи та користувацький інтерфейс. У четвертому розділі розглянуто реалізацію клієнтської та серверної частин, основний функціонал вебзастосунку, результати тестування та коротку інструкцію користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Постановка задачі

Планування навчальної діяльності здається простою задачею тільки на початку семестру, коли кількість завдань ще невелика. З часом студент отримує нові роботи з різних дисциплін, у кожної з яких є власні вимоги, формат виконання та дедлайн. Якщо не фіксувати ці дані в єдиній системі, навчальне навантаження швидко стає важко контрольованим.

Типова ситуація виглядає так: розклад занять зберігається окремо, частина завдань записана в нотатках, частина залишилася в повідомленнях, дедлайни внесені в календар, а файли з описами робіт лежать у різних папках. Кожен із цих інструментів може бути корисним окремо, але разом вони не створюють цілісної картини.

Неструктуроване планування призводить до кількох практичних проблем. Збільшується ризик пропустити важливий термін, складно оцінити навантаження на найближчі дні, а також важче зрозуміти, які задачі вже виконані, які перебувають у процесі, а які ще не розпочаті.

Для вирішення цієї проблеми потрібно створити вебзастосунок Study Planner, який об’єднує навчальні задачі, дедлайни, календар, статуси, теги, пошук, фільтрацію, статистику і налаштування в одному середовищі. Такий застосунок не повинен перетворюватися на складну систему керування навчальним закладом. Його завдання простіше: допомогти одному студенту впорядкувати власні навчальні справи.

1.2 Аналіз існуючих рішень

Google Calendar є одним із найпоширеніших інструментів для планування календарних подій [1]. Його основне призначення полягає у

створенні подій, налаштуванні нагадувань, повторюваних активностей і перегляді розкладу за днями, тижнями або місяцями. Для студента цей сервіс може бути корисним під час збереження розкладу занять, консультацій, контрольних робіт, заліків, екзаменів або інших подій, які мають чітку прив'язку до дати й часу. Перевагою Google Calendar є простота використання, синхронізація між пристроями та можливість швидко отримувати нагадування про заплановані події.

Однак Google Calendar не є повноцінним менеджером навчальних задач. Його логіка більше орієнтована на події, а не на процес виконання завдань. Наприклад, у ньому незручно працювати зі статусами задач, відстежувати етапи виконання, використовувати теги для поділу завдань за дисциплінами або аналізувати статистику виконання. Якщо студент хоче бачити не лише дату дедлайну, а й загальний стан задачі, її категорію, прогрес і зв'язок з іншими навчальними активностями, можливостей Google Calendar стає недостатньо. Тому цей сервіс добре вирішує задачу календарного планування, але не повністю закриває потребу в комплексному навчальному планувальнику.

Todoist є популярним сервісом для керування списками задач. Він дозволяє швидко створювати завдання, встановлювати терміни виконання, додавати пріоритети, групувати задачі за проєктами та позначати їх як виконані [2]. Для студента Todoist може бути корисним як простий список справ, у якому можна записувати домашні завдання, лабораторні роботи, підготовку до контрольних заходів або інші навчальні активності. Сильна сторона цього сервісу полягає у швидкому додаванні задач і зрозумілому інтерфейсі.

Проте Todoist залишається універсальним менеджером задач і не орієнтований саме на навчальний процес. У ньому можна зручно вести список справ, але не завжди зручно поєднувати ці задачі з календарними подіями, навчальними дедлайнами та загальною статистикою виконання. Для студента важливо бачити не тільки окремі завдання, а й загальне навантаження за

певний період, кількість виконаних і невиконаних задач, а також зв’язок між дедлайнами та календарем. У Todoist частина цих можливостей є обмеженою або потребує додаткових налаштувань, тому сервіс не повністю відповідає задачам Study Planner.

Notion є гнучким інструментом для організації інформації. Його можна використовувати для створення сторінок, баз даних, таблиць, списків задач, календарів, навчальних конспектів і персональних систем планування [3]. Для студента Notion може бути корисним завдяки можливості зберігати в одному середовищі різні типи навчальної інформації: нотатки, дедлайни, посилання, таблиці, матеріали до занять і плани виконання завдань. Основною перевагою Notion є його гнучкість, оскільки користувач може самостійно створити структуру, яка відповідає його стилю навчання.

Водночас саме ця гнучкість може бути недоліком. Щоб Notion став зручним навчальним планувальником, студенту потрібно самостійно створити бази даних, налаштувати властивості, зв’язки, подання, фільтри та шаблони. Це потребує часу й певного досвіду роботи з сервісом. Для щоденного швидкого використання Notion може виглядати перевантаженим, особливо якщо студенту потрібно просто створити задачу, встановити дедлайн, змінити статус або переглянути найближчі події. На відміну від Notion, Study Planner має надавати вже готову логіку роботи із задачами, календарем і статистикою без необхідності ручного налаштування всієї структури.

Trello побудований на основі візуальних дошок, списків і карток. Його часто використовують для організації проєктів, командної роботи або відстеження процесу виконання задач за етапами [4]. Наприклад, завдання можна розміщувати в колонках “Заплановано”, “В процесі” та “Виконано”, переміщуючи картки між ними. Для студента Trello може бути корисним під час роботи над великими проєктами, курсовими завданнями або груповими роботами, де важливо бачити етапи виконання.

Однак для персонального навчального планування Trello підходить не завжди. Його основна логіка пов’язана з дошками та картками, а не з

календарним контролем дедлайнів і статистикою навчального прогресу. Календарне представлення, аналітика виконання та робота з навчальними подіями не є центральними можливостями цього сервісу. Якщо студенту потрібно швидко побачити дедлайни на тиждень, оцінити кількість активних задач або переглянути статистику виконання, Trello може вимагати додаткових налаштувань або інтеграцій. Тому цей сервіс зручний для візуального керування задачами, але менш зручний як цілісний навчальний планувальник.

Microsoft To Do є простим сервісом для створення списків справ [5]. Він дозволяє додавати задачі, встановлювати терміни виконання, створювати окремі списки та позначати завдання як завершені. Його перевагою є мінімалістичний інтерфейс і простота використання. Для студента Microsoft To Do може бути корисним у випадках, коли потрібно швидко записати список навчальних справ або щоденних задач без складних налаштувань.

Недоліком Microsoft To Do є обмеженість функціональних можливостей для комплексного навчального планування. Сервіс добре працює як простий список справ, але не надає достатньо можливостей для роботи з тегами, розгорнутою статистикою, календарним навантаженням і гнучким аналізом виконання задач. Для Study Planner важливо не лише зберігати задачі, а й показувати їх у зв’язку з дедлайнами, подіями, статусами та загальним прогресом студента. Тому Microsoft To Do можна розглядати як зручний базовий інструмент, але його можливостей недостатньо для реалізації повноцінної системи навчального планування.

1.3 Результати проведеного аналізу

Після аналізу існуючих рішень можна зробити висновок, що кожен із розглянутих сервісів має власну сильну сторону. Google Calendar краще підходить для календарних подій, Todoist і Microsoft To Do – для списків задач,

Notion – для гнучкого зберігання інформації, а Trello – для візуального керування процесами. Проте жоден із цих сервісів не вирішує повністю задачу персонального навчального планування в тому вигляді, який передбачено у Study Planner. Для більш наочного порівняння було сформовано таблицю 1.1.

Таблиця 1.1 – Порівняння існуючих рішень із Study Planner

Критерій	**Google Calendar**	**Todoist**	**Notion**	**Trello**	**Microsoft To Do**	**Study Planner**
Створення задач	+/–	+	+	+	+	+
Календар	+	+/–	+	+/–	+/–	+
Фільтрації	+/–	+	+	+	+/–	+
Пошук	+	+	+	+	+	+
Статистика	–	+/–	+/–	+/–	–	+
Персоналізація і налаштування	+/–	+	+	+	+/–	+
Простота використання	+	+	+/–	+/–	+	+
Орієнтація на студентів	–	–	–	–	–	+

З таблиці видно, що більшість існуючих сервісів добре реалізує окремі функції, але не поєднує їх у простій формі, орієнтованій саме на студента. Наприклад, Google Calendar має сильну календарну частину, але не є повноцінним менеджером задач. Todoist і Microsoft To Do зручні для списків справ, але мають обмежені можливості для аналізу навчального прогресу. Notion є гнучким, однак потребує попереднього налаштування структури, а Trello більше підходить для візуального керування процесами.

Отже, розроблення Study Planner є доцільним, оскільки застосунок поєднує задачі, календарні події, фільтрацію, пошук, статистику та персональні налаштування в одному середовищі. При цьому система орієнтована не на універсальне керування будь-якими процесами, а саме на потреби студента у щоденному навчальному плануванні.

Після порівняння існуючих рішень можна окремо виділити функціональні потреби, які є найбільш важливими для користувача Study Planner. Вони пов’язані не лише зі створенням задач, а й з контролем дедлайнів, переглядом навантаження, структуруванням навчальної інформації та оцінкою власного прогресу, що продемонстровано в таблиці 1.2.

Таблиця 1.2 – Функціональні потреби користувача Study Planner

Потреба користувача	**Опис**	**Як реалізовано у Study Planner**
Контроль дедлайнів	Користувач повинен бачити терміни виконання задач	У задачі зберігається дата дедлайну
Перегляд навантаження	Користувач повинен бачити задачі й події за датами	Реалізовано календар
Структурування задач	Задачі повинні мати статуси й теги	Реалізовано статуси та теги
Швидкий пошук	Користувач повинен швидко знаходити потрібні задачі	Реалізовано пошук і фільтрацію
Оцінка прогресу	Користувач повинен бачити кількість виконаних задач	Реалізовано сторінку статистики
Персоналізація	Користувач повинен зберігати власні налаштування	Реалізовано сторінку налаштувань

Отже, Study Planner має бути не універсальним сервісом для всіх типів задач, а простим інструментом для навчального планування.

Саме тому основний акцент у застосунку зроблено на дедлайнах, статусах, тегах, календарному перегляді, пошуку, фільтрації, статистиці виконання та персональних налаштуваннях.

Такий набір функцій дозволяє студенту працювати з основною навчальною інформацією в одному середовищі без зайвого ускладнення інтерфейсу.

1.4 Висновки до розділу 1

У першому розділі було розглянуто проблему організації навчальних задач і календарних подій студента. Було визначено, що під час навчання студент працює з великою кількістю інформації: дедлайнами, завданнями з різних дисциплін, контрольними роботами, консультаціями, індивідуальними проєктами та іншими подіями. Якщо ці дані зберігаються в різних місцях, наприклад у нотатках, месенджерах, календарях або окремих файлах, студенту стає складніше контролювати власне навчальне навантаження.

У процесі аналізу було розглянуто існуючі рішення для планування задач і подій, зокрема Google Calendar, Todoist, Notion, Trello та Microsoft To Do. Кожен із цих сервісів має власні переваги. Google Calendar добре підходить для роботи з календарними подіями, Todoist і Microsoft To Do зручні для ведення списків задач, Notion дає можливість гнучко організовувати інформацію, а Trello корисний для візуального керування процесами. Проте ці рішення або є занадто універсальними, або потребують додаткового налаштування, або не поєднують у простій формі задачі, дедлайни, календарні події, статуси, теги та статистику виконання.

Проведене порівняння показало, що студенту потрібен не перевантажений універсальний сервіс, а простий інструмент для щоденного навчального планування. Такий інструмент повинен дозволяти швидко створювати задачі, встановлювати дедлайни, переглядати навантаження в календарі, змінювати статуси виконання, використовувати теги, знаходити потрібні задачі через пошук і фільтрацію, а також оцінювати власний прогрес за допомогою статистики.

Отже, створення вебзастосунку Study Planner є доцільним. Його розроблення дозволяє об’єднати основні функції, необхідні студенту для організації навчального процесу, в одному середовищі. На відміну від розглянутих сервісів, Study Planner орієнтований саме на персональне навчальне планування, тому його функціональність має бути простою, зрозумілою і практично корисною для щоденного використання.

2 ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Вибір технологій для клієнтської частини

Клієнтська частина вебзастосунку Study Planner відповідає за взаємодію користувача із системою. Саме через інтерфейс студент створює навчальні задачі, встановлює дедлайни, змінює статуси виконання, працює з календарем, використовує пошук, фільтрацію, переглядає статистику та змінює персональні налаштування. Через це клієнтська частина повинна бути не лише візуально зрозумілою, а й достатньо динамічною, щоб швидко реагувати на дії користувача.

У розробленому вебзастосунку для клієнтської частини було використано HTML [6], CSS [7], JavaScript та бібліотеку React. HTML застосовується для побудови структури сторінок, CSS – для оформлення інтерфейсу, а JavaScript – для реалізації інтерактивної логіки. React використано як основний інструмент для створення компонентного інтерфейсу, роботи зі станом застосунку та оновлення даних без повного перезавантаження сторінки.

Такий набір технологій є доцільним для Study Planner, оскільки застосунок має багато повторюваних елементів інтерфейсу: форми створення задач, картки задач, кнопки зміни статусу, фільтри, елементи календаря та блоки статистики. Використання React дозволяє винести ці елементи в окремі компоненти й повторно використовувати їх у різних частинах застосунку. Це робить структуру клієнтської частини більш зрозумілою та спрощує подальше розширення функціональності. Також React зручно працює зі зміною стану інтерфейсу, тому список задач, фільтри, календар і статистика можуть оновлюватися без перезавантаження сторінки. Для користувача це робить роботу із Study Planner швидшою та більш схожою на звичайний застосунок, а не на статичний сайт.

2.1.1 HTML, CSS та JavaScript у клієнтській частині

HTML, CSS і JavaScript є базовими технологіями, на яких будується клієнтська частина вебзастосунку. У Study Planner HTML використовується для формування структури сторінок і розміщення основних елементів інтерфейсу: форм, кнопок, полів введення, списків задач, блоків статистики та календарних елементів.

CSS відповідає за зовнішній вигляд застосунку. За допомогою стилів задається розміщення елементів, відступи, розміри, кольори, оформлення кнопок, таблиць, карток задач і сторінок. Для навчального планувальника важливо, щоб інтерфейс був не перевантаженим, тому стилізація має підтримувати просту й зрозумілу структуру сторінок.

JavaScript використовується для інтерактивної поведінки застосунку. Він забезпечує обробку дій користувача, роботу з формами, надсилання запитів на сервер, отримання даних через API та оновлення інформації на сторінці. Наприклад, після створення задачі клієнтська частина надсилає дані на сервер, отримує відповідь і оновлює список задач [8].

Однак використання лише HTML, CSS і JavaScript без додаткових бібліотек ускладнило б реалізацію Study Planner.

Застосунок має багато динамічних частин: авторизацію, задачі, календар, фільтри, статистику, налаштування та постійне оновлення даних. Якщо реалізовувати все вручну, код став би менш структурованим і складнішим для підтримки. Тому базові технології використовуються як основа клієнтської частини, а для організації інтерфейсу було застосовано React. Крім того, без компонентного підходу було б складніше повторно використовувати однакові елементи інтерфейсу, наприклад кнопки, форми або картки задач.

React дозволяє краще впорядкувати ці частини застосунку й зменшити дублювання коду. Завдяки цьому клієнтська частина Study Planner стає більш гнучкою для подальшого розвитку та внесення змін.

2.1.2 React як основна бібліотека для побудови інтерфейсу

React є бібліотекою JavaScript для створення користувацьких інтерфейсів. У Study Planner React використано як основний інструмент для побудови клієнтської частини. Його головною перевагою є компонентний підхід, за якого інтерфейс поділяється на окремі частини. Кожен компонент відповідає за певний елемент або блок застосунку [9].

У Study Planner компонентами можуть бути форма створення задачі, список задач, картка задачі, фільтр за статусом, календарний елемент, блок статистики або сторінка налаштувань. Такий підхід дозволяє не дублювати однакову логіку в різних місцях і робить код більш зрозумілим.

React також зручний для роботи зі станом застосунку. Стан змінюється тоді, коли користувач створює нову задачу, редагує її, змінює статус, застосовує фільтр або переходить між сторінками. Завдяки React інтерфейс може оновлюватися без повного перезавантаження сторінки. Це важливо для Study Planner, оскільки користувач повинен швидко бачити результат своїх дій.

2.1.3 Порівняння використаних технологій з альтернативами

Під час вибору клієнтських технологій було також розглянуто альтернативні варіанти, зокрема Angular і Vue. Ці інструменти також дозволяють створювати динамічні вебзастосунки, однак для Study Planner було обрано React, оскільки він забезпечує достатню гнучкість, компонентний підхід і не є надмірно складним для реалізації дипломного проєкту.

Angular є повноцінним фреймворком із великою кількістю вбудованих можливостей [10]. Він добре підходить для великих корпоративних систем, але для Study Planner може бути надлишковим. Vue є легшим і також підтримує компонентний підхід, однак у межах проєкту React був більш

доцільним через зручність роботи зі станом, популярність і широку екосистему [11].

Для більш наочного порівняння використаних технологій та альтернатив було сформовано таблицю 2.1.

Таблиця 2.1 – Порівняння технологій клієнтської частини

Критерій	**HTML, CSS, JavaScript**	**React**	**Angular**	**Vue**
Роль у проєкті	+	+	–	–
Створення структури сторінок	+	+	+	+
Оформлення інтерфейсу	+	+	+	+
Динамічне оновлення даних	+/–	+	+	+
Компонентний підхід	+/–	+	+	+
Робота зі списками задач, формами й фільтрами	+/–	+	+/–	+
Складність для дипломного проєкту	+	+	–	+/–
Доцільність для Study Planner	+	+	–	+/–

З таблиці видно, що HTML, CSS і JavaScript є необхідною основою клієнтської частини, однак для повної реалізації Study Planner їх недостатньо без додаткової організації коду. React доповнює базові вебтехнології та дозволяє зручно будувати інтерфейс із компонентів, працювати зі станом і оновлювати дані без перезавантаження сторінки.

Angular і Vue також могли б бути використані для створення клієнтської частини, проте в межах цієї роботи вони не були обрані. Angular є більш складним і надлишковим для невеликого навчального вебзастосунку, а Vue, хоча й є зручним інструментом, не мав суттєвих переваг над React для поставленої задачі.

Отже, для клієнтської частини Study Planner було використано поєднання HTML, CSS, JavaScript і React. Базові вебтехнології забезпечують

структуру, стилізацію та інтерактивність сторінок, а React дозволяє організувати інтерфейс у вигляді компонентів і зручно працювати з динамічними даними. Такий вибір відповідає потребам Study Planner і є доцільним для реалізації вебзастосунку навчального планування.

2.2 Вибір технологій для серверної частини

Серверна частина вебзастосунку Study Planner відповідає за обробку запитів від клієнтської частини, перевірку авторизації користувача, взаємодію з базою даних PostgreSQL і повернення результатів React-клієнту. Саме серверна частина приймає дані з форм, обробляє створення та редагування задач, працює з календарними подіями, тегами, статистикою і персональними налаштуваннями користувача.

У розробленому проєкті для серверної частини було використано Node.js та Express.js. Node.js дає можливість виконувати JavaScript на сервері, а Express.js використовується для створення REST API, опису маршрутів, обробки HTTP-запитів і підключення проміжних обробників. Такий підхід добре підходить для Study Planner, оскільки клієнтська частина також реалізована на JavaScript за допомогою React. Завдяки цьому клієнтська і серверна частини працюють в одній мовній екосистемі.

Node.js у проєкті виконує роль середовища для запуску серверної логіки [12]. Через нього сервер приймає запити від клієнта, обробляє дані користувача та взаємодіє з базою даних. Наприклад, коли студент створює нову задачу, React-клієнт надсилає запит на сервер, а серверна частина перевіряє отримані дані, визначає авторизованого користувача і зберігає задачу в PostgreSQL.

Express.js використовується як основний фреймворк для побудови API. За його допомогою реалізуються маршрути для реєстрації, входу в систему, отримання списку задач, створення нової задачі, редагування, видалення,

роботи з календарними подіями, статистикою та налаштуваннями. Також Express.js дозволяє використовувати middleware для перевірки авторизації перед виконанням запитів, які працюють із персональними даними користувача [13].

Для більш наочного порівняння використаної серверної технології з можливими альтернативами було сформовано таблицю 2.2.

Таблиця 2.2 – Порівняння серверної вибору серверної частини

Критерій	**Node.js / Express.js**	**Python / FastAPI**	**PHP / Laravel**
Роль у проєкті	+	–	–
Створення REST API	+	+	+/–
Взаємодія з React-клієнтом	+	+/–	+/–
Авторизація користувача	+	+	+
Робота з PostgreSQL	+	+	+
Єдина JavaScript-екосистема	+	–	–
Складність для дипломного проєкту	+	+/–	–
Доцільність для Study Planner	+	+/–	–

З таблиці видно, що всі розглянуті технології можуть бути використані для створення серверної частини вебзастосунку. Проте для Study Planner найбільш доцільним варіантом є Node.js та Express.js, оскільки вони дозволяють реалізувати простий і зрозумілий REST API, зручно обробляти запити від React-клієнта, працювати з PostgreSQL і перевіряти авторизацію користувача.

Використання JavaScript як на клієнтській, так і на серверній частині спрощує розроблення та підтримку застосунку. Саме тому для серверної частини Study Planner було обрано Node.js та Express.js.

2.3 Вибір бази даних

База даних у Study Planner використовується для збереження основної інформації застосунку: користувачів, навчальних задач, календарних подій, тегів, статусів і персональних налаштувань.

Оскільки ці дані пов’язані між собою, база даних повинна підтримувати структуроване збереження інформації, зв’язки між таблицями та цілісність даних.

У розробленому проєкті використано PostgreSQL [14]. Ця система керування базами даних добре підходить для Study Planner, оскільки застосунок має реляційну структуру даних.

Один користувач може мати багато задач і календарних подій, одна задача може мати кілька тегів, а налаштування повинні бути пов’язані з конкретним користувачем. Такі зв’язки зручно реалізовувати саме в реляційній базі даних.

PostgreSQL у проєкті виконує роль основного сховища даних. У ній зберігаються облікові записи користувачів, задачі з назвами, описами, дедлайнами і статусами, календарні події, теги та персональні налаштування. Завдяки використанню зовнішніх ключів можна забезпечити зв’язок між таблицями та гарантувати, що кожна задача або подія належить конкретному користувачу.

Важливою перевагою PostgreSQL є підтримка цілісності даних. Це дозволяє уникати ситуацій, коли задача посилається на неіснуючого користувача або зв’язок між задачею і тегом створюється некоректно.

Для Study Planner це має велике значення, оскільки застосунок працює з персональними навчальними даними, які повинні бути впорядкованими й захищеними від змішування між різними користувачами.

Для більш наочного порівняння використаної бази даних з можливими альтернативами було сформовано таблицю 2.3.

Таблиця 2.3 – Порівняння баз даних для Study Planner

Критерій	**PostgreSQL**	**MySQL**	**SQLite**	**MongoDB**
Роль у проєкті	+	–	–	–
Тип бази даних	+	+	+/–	+/–
Підтримка зв’язків між даними	+	+	+/–	–
Цілісність даних	+	+	+/–	+/–
Придатність для багатокористувацького вебзастосунку	+	+	–	+
Зручність для задач, подій і тегів	+	+	+/–	+/–
Можливість подальшого розширення	+	+	–	+/–
Вибір для Study Planner	+	–	–	–

З таблиці видно, що PostgreSQL найкраще відповідає структурі Study Planner. Вона дозволяє зберігати дані у вигляді пов’язаних таблиць, забезпечувати зв’язок між користувачами, задачами, календарними подіями, тегами й налаштуваннями, а також підтримувати цілісність даних.

PostgreSQL було обрано тому, що це реляційна СКБД, яка добре підходить для структурованих даних, підтримує зв’язки між таблицями, обмеження цілісності та складні запити. Для Study Planner це є найбільш логічним рішенням, оскільки застосунок працює з персональними навчальними даними, які мають чітку структуру та зв’язки між собою.

2.4 Вибір засобів розгортання та організації середовища

Окрім клієнтської частини, серверної частини та бази даних, у проєкті використано засоби для організації середовища запуску. Для цього застосовано Docker Compose та Nginx.

Docker Compose використовується для запуску кількох частин системи як окремих сервісів [15]. У конфігурації проєкту передбачено сервіси frontend,

backend і db. Сервіс frontend відповідає за клієнтську частину, backend запускає сервер Express.js, а db містить PostgreSQL. Такий підхід спрощує запуск застосунку, оскільки всі основні компоненти системи можна підняти разом. Nginx використовується для роздачі зібраної клієнтської частини та проксування API-запитів [16]. Користувач відкриває вебзастосунок через браузер, а запити до /api/ передаються на серверну частину. Завдяки цьому клієнтська та серверна частини працюють як єдина система.

Використання Docker Compose і Nginx робить проєкт зручнішим для запуску та демонстрації. Засоби для реалізації наведені в таблиці 2.4.

Таблиця 2.4 – Засоби розгортання Study Planner

Засіб	**Роль у проєкті**	**Призначення**
Docker Compose	Організація запуску сервісів	Одночасний запуск frontend, backend і PostgreSQL
Dockerfile frontend	Збирання клієнтської частини	Формування готової React-збірки
Nginx	Вебсервер і проксі	Роздача frontend і перенаправлення /api/ на backend
PostgreSQL volume	Збереження даних	Збереження даних БД між перезапусками контейнерів

2.5 Висновки до розділу 2

У другому розділі було розглянуто засоби реалізації вебзастосунку Study Planner і обґрунтовано вибір технологій для клієнтської частини, серверної частини, бази даних та середовища розгортання.

Для клієнтської частини було використано React, Vite, React Router DOM, FullCalendar, Tailwind CSS, HTML, CSS і JavaScript. Такий набір засобів дозволяє створити динамічний компонентний інтерфейс, реалізувати сторінки застосунку, календарний перегляд, маршрутизацію та зручне оформлення інтерфейсу.

Для серверної частини було використано Node.js та Express.js. Додатково застосовано бібліотеки pg, bcryptjs, jsonwebtoken, dotenv і cors. Вони забезпечують роботу REST API, підключення до PostgreSQL, хешування паролів, перевірку JWT-токенів і коректну взаємодію між клієнтською та серверною частинами.

Для збереження даних було обрано PostgreSQL, оскільки застосунок працює зі структурованими персональними даними та зв’язками між користувачами, задачами, календарними подіями, нагадуваннями й налаштуваннями. PostgreSQL дозволяє забезпечити цілісність даних і надійне збереження інформації.

Для організації запуску й розгортання використано Docker Compose та Nginx. Docker Compose дозволяє запускати клієнтську частину, серверну частину й базу даних як окремі сервіси, а Nginx забезпечує роздачу клієнтської частини та проксування API-запитів.

Отже, обраний набір технологій відповідає фактичній реалізації Study Planner і дозволяє створити повноцінний клієнт-серверний вебзастосунок із динамічним інтерфейсом, REST API [17], авторизацією користувача, календарним переглядом, надійним збереженням даних і зручним середовищем запуску.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Аналіз вимог до системи

Проєктування вебзастосунку Study Planner починається з визначення вимог до системи. На цьому етапі потрібно зрозуміти, які задачі має виконувати застосунок, які дії повинен мати змогу виконувати студент і які характеристики мають бути забезпечені під час роботи системи. Для Study Planner основною ідеєю є створення зручного інструмента для особистого навчального планування, тому вимоги пов’язані із задачами, дедлайнами, календарними подіями, нагадуваннями, статистикою та персональними налаштуваннями.

Функціональні вимоги описують конкретні можливості, які повинна надавати система користувачу. У Study Planner користувачем є студент, який після реєстрації та входу в систему може створювати навчальні задачі, редагувати їх, змінювати статуси, додавати дедлайни й теги, переглядати календар, отримувати нагадування та аналізувати власний прогрес.

Функціональні вимоги наведені в таблиці 3.1.

Таблиця 3.1 – Функціональні вимоги до Study Planner

Код	**Назва вимоги**	**Опис**	**Пріоритет**
FR-01	Реєстрація користувача	Користувач може створити акаунт за допомогою імені, електронної пошти та пароля	Високий
FR-02	Вхід до системи	Користувач може авторизуватися та отримати доступ до власних даних	Високий
FR-03	Додавання завдання	Користувач може створити навчальну задачу з назвою, описом, дедлайном і тегами	Високий
FR-04	Редагування завдання	Користувач може змінити дані вже створеної задачі	Високий

Продовження таблиці 3.1

Код	**Назва вимоги**	**Опис**	**Пріоритет**
FR-05	Видалення завдання	Користувач може видалити неактуальну задачу	Середній
FR-06	Зміна статусу	Користувач може змінювати стан задачі: очікує, в процесі, виконано	Високий
FR-06	Зміна статусу	Користувач може змінювати стан задачі: очікує, в процесі, виконано	Високий
FR-07	Перегляд календаря	Користувач може переглядати задачі й події у календарному вигляді	Високий
FR-08	Отримання нагадувань	Система може зберігати нагадування, пов’язані з календарними подіями	Середній
FR-09	Пошук і фільтрація	Користувач може швидко знаходити задачі за текстом, статусом, тегом або датою	Середній
FR-10	Перегляд статистики	Система відображає кількість активних, виконаних і прострочених задач	Середній
FR-11	Персональні налаштування	Користувач може змінювати тему інтерфейсу, вигляд календаря та нагадування	Середній

Нефункціональні вимоги описують якість роботи системи. Для Study Planner важливо, щоб застосунок був зрозумілим у щоденному використанні, коректно працював у браузері, захищав персональні дані користувача та забезпечував швидку взаємодію між клієнтською і серверною частинами.

Нефункціональні вимоги наведені в таблиці 3.2.

Таблиця 3.2 – Нефункціональні вимоги до Study Planner

Код	**Назва вимоги**	**Опис**
NFR-01	Зручність інтерфейсу	Інтерфейс має бути простим для щоденного створення задач, перегляду календаря та зміни статусів
NFR-02	Захист даних	Дані користувача повинні бути доступні тільки після авторизації

Продовження таблиці 3.2

Код	**Назва вимоги**	**Опис**
NFR-03	Доступ до власних записів	Користувач має бачити лише власні задачі, події, нагадування та налаштування
NFR-04	Робота в браузері	Вебзастосунок має коректно працювати в сучасних браузерах
NFR-05	Адаптивність	Інтерфейс має залишатися зручним на різних розмірах
NFR-06	Швидкість API	Запити до серверної частини мають виконуватися без помітної затримки
NFR-07	Цілісність даних	PostgreSQL має підтримувати зв’язки між користувачами, задачами, подіями та налаштуваннями
NFR-08	Можливість розширення	Структура системи має дозволяти додавати нові функції без повної перебудови застосунку

Після визначення функціональних і нефункціональних вимог було описано основні сценарії використання Study Planner. Вони показують, як студент взаємодіє із системою під час щоденного навчального планування. Сценарії охоплюють реєстрацію, вхід до системи, створення задач, роботу з календарем, пошук, фільтрацію, перегляд статистики та зміну персональних налаштувань. Такий опис дозволяє пов’язати вимоги до системи з реальними діями користувача. Завдяки цьому можна зрозуміти, які функції є основними для роботи застосунку, а які доповнюють процес планування.

Для того, щоб наочно показати як буде проходити використання застосунку, було створено основні сценарії використання, які показані в таблиці 3.3.

Після визначення основних сценаріїв використання було сформовано технічне завдання для розроблення вебзастосунку Study Planner. Його призначення полягає в тому, щоб узагальнити вимоги до системи та визначити, який результат має бути отриманий після реалізації. Технічне завдання допомагає пов’язати потреби користувача з майбутньою структурою застосунку, базою даних, інтерфейсом і серверною логікою.

Таблиця 3.3 – Основні сценарії використання Study Planner

№	Сценарій	Мета користувача	Початкова умова	Очікуваний результат
1	Реєстрація користувача	Створити персональний акаунт	Користувач ще не має акаунту	У базі даних створено нового користувача
2	Вхід до системи	Отримати доступ до власних даних	Користувач має зареєстрований акаунт	Користувач авторизований і переходить до застосунку
3	Формування навчального плану	Організувати задачі та події за датами	Користувач увійшов у систему	Створено набір задач і календарних подій
4	Створення задачі	Додати нову навчальну задачу	Користувач авторизований	Задача збережена в базі даних
5	Редагування задачі	Уточнити назву, опис, дедлайн або теги	Задача вже існує	Дані задачі оновлено
6	Зміна статусу задачі	Позначити поточний стан виконання	Задача вже творена	Статус задачі змінено
7	Додавання календарної події	Запланувати подію з датою і часом	Користувач авторизований	Подія відображається в календарі
8	Перегляд календаря	Побачити задачі й події за датами	У системі є задачі або події	Користувач бачить навчальне навантаження в календарі
9	Пошук і фільтрація задач	Швидко знайти потрібні задачі	У списку є кілька задач	Відображаються тільки задачі, що відповідають умовам
10	Перегляд статистики	Оцінити власний прогрес	У користувача є задачі	Показано кількість активних, виконаних і прострочених задач
11	Зміна налаштувань	Налаштувати застосунок під власні потреби	Користувач авторизований	Налаштування збережено для конкретного користувача
12	Отримання нагадування	Не пропустити важливу подію	Для події створено нагадування	Система зберігає або обробляє нагадування

Згідно з технічним завданням, Study Planner має забезпечувати особисте навчальне планування студента. Користувач повинен мати змогу створити акаунт, увійти до системи, додавати навчальні задачі, встановлювати дедлайни, створювати календарні події, змінювати статуси задач,

використовувати пошук і фільтрацію, переглядати статистику виконання та зберігати персональні налаштування. Усі ці можливості мають бути об’єднані в одному вебзастосунку, щоб студенту не потрібно було окремо використовувати нотатки, календар і список справ.

Окремою вимогою є персональний характер роботи з даними. Кожен користувач повинен мати доступ тільки до власних задач, подій, нагадувань і налаштувань. Це означає, що система має перевіряти користувача перед виконанням захищених дій і зберігати дані так, щоб записи різних користувачів не змішувалися між собою.

Також технічне завдання передбачає, що застосунок має бути зручним для щоденного використання. Інтерфейс не повинен бути перевантаженим, а основні дії – створення задачі, зміна статусу, перегляд календаря або пошук – мають виконуватися швидко і зрозуміло. Це важливо, оскільки Study Planner розрахований не на складне адміністрування навчального процесу, а на персональну організацію роботи студента.

Таким чином, технічне завдання визначає Study Planner як вебзастосунок для особистого планування навчальної діяльності. Система повинна поєднувати роботу із задачами, дедлайнами, календарними подіями, статистикою та налаштуваннями, а також забезпечувати авторизований доступ до персональних даних користувача. Детальна структура реалізації, вибір архітектури та взаємодія компонентів розглядаються в наступному підрозділі.

3.2 Проєктування прецедентів використання

Прецеденти використання описують основні можливості, які отримує студент під час роботи зі Study Planner (див. рис. 3.1). Оскільки застосунок орієнтований на особисте навчальне планування, у системі передбачено одного основного актора – студента.

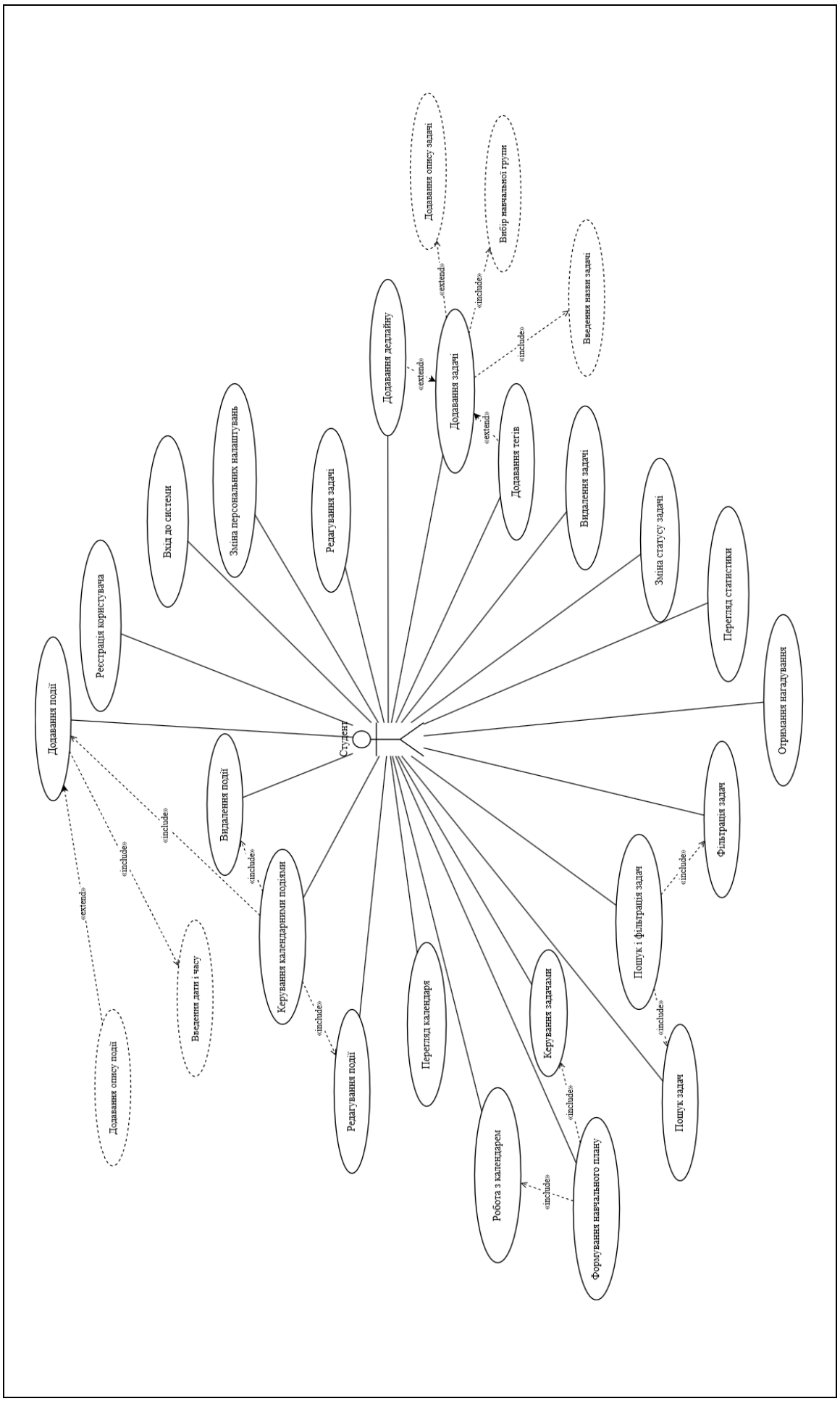


Рисунок 3.1 – Діаграма варіантів використання Study Planner

Він працює з власним акаунтом, задачами, календарем, нагадуваннями, статистикою та налаштуваннями. Таке подання допомагає зрозуміти, які дії є головними для користувача і як вони пов’язані між собою. Також прецеденти показують, що всі основні функції системи зосереджені навколо персонального планування навчальної діяльності.

Діаграма варіантів використання показує, що студент може зареєструватися, увійти до системи, сформувати навчальний план, додавати й редагувати задачі, переглядати календар, отримувати нагадування та переглядати статистику. Це продемонстровано в таблиці 3.4.

Таблиця 3.4 – Опис основних прецедентів Study Planner

Код	**Прецедент**	**Опис**
UC-01	Реєстрація користувача	Створення персонального аккаунту студента.
UC-02	Вхід до системи	Перевірка облікових даних і отримання доступу до власної інформації.
UC-03	Створення навчального плану	Формування набору задач і подій на певний період.
UC-04	Додавання завдання	Створення навчальної задачі з назвою, описом, дедлайном і тегами.
UC-05	Редагування завдання	Оновлення даних задачі після зміни навчальних умов.
UC-06	Перегляд календаря	Перегляд задач і подій за датами.
UC-07	Отримання нагадування	Збереження або обробка нагадування про подію чи дедлайн.
UC-08	Перегляд статистики	Оцінка кількості активних, виконаних і прострочених задач.

Такий набір прецедентів відповідає меті Study Planner. Система не містить ролей викладача або адміністратора, тому всі сценарії зосереджені на особистій роботі студента з власними навчальними задачами.

3.3 Проєктування моделі даних

Модель даних Study Planner побудована навколо персональних записів користувача.

Центральною сутністю є користувач, оскільки всі задачі, календарні події, нагадування та налаштування мають належати конкретному акаунту. Такий підхід дозволяє відокремити дані різних користувачів і забезпечити персональний характер роботи застосунку.

Для чіткого визначення сутностей та основних атрибутів була зроблена таблиця 3.5.

Таблиця 3.5 – Основні сутності бази даних Study Planner

Сутність	**Призначення**	**Основні атрибути**
users	Зберігання даних користувачів	id, name, email, password_hash, google_id, avatar_url, created_at
tasks	Зберігання навчальних задач	id, user_id, event_id, title, description, status,deadline, due_date, tags, created_at, updated_at
events	Зберігання календарних подій	id, user_id, title, type, subject, description, start_time, end_time, created_at
reminders	Зберігання нагадувань про події	id, user_id, event_id, remind_at, is_sent, created_at
user_settings	Зберігання персональних налаштувань	user_id, theme, start_view, notifications, updated_at

Для задач у базі даних передбачено поле tags типу TEXT[]. Це означає, що теги зберігаються безпосередньо в записі задачі. Для персонального планувальника такий варіант є достатнім, бо теги використовуються для пошуку, фільтрації та швидкого структурування задач одного користувача. Якщо систему надалі розширювати до складнішої багатокористувацької моделі, теги можна винести в окремі таблиці.

Поле event_id у таблиці tasks дозволяє пов’язати задачу з календарною подією. Це потрібно для того, щоб задачі з дедлайнами могли відображатися в календарі разом з іншими подіями користувача. Завдяки цьому календар показує не лише окремі записи, а й навчальне навантаження за датами.

Наочно діаграма бази даних продемонстрована на рисунку 3.2. ER-діаграма показує зв’язки між таблицями.

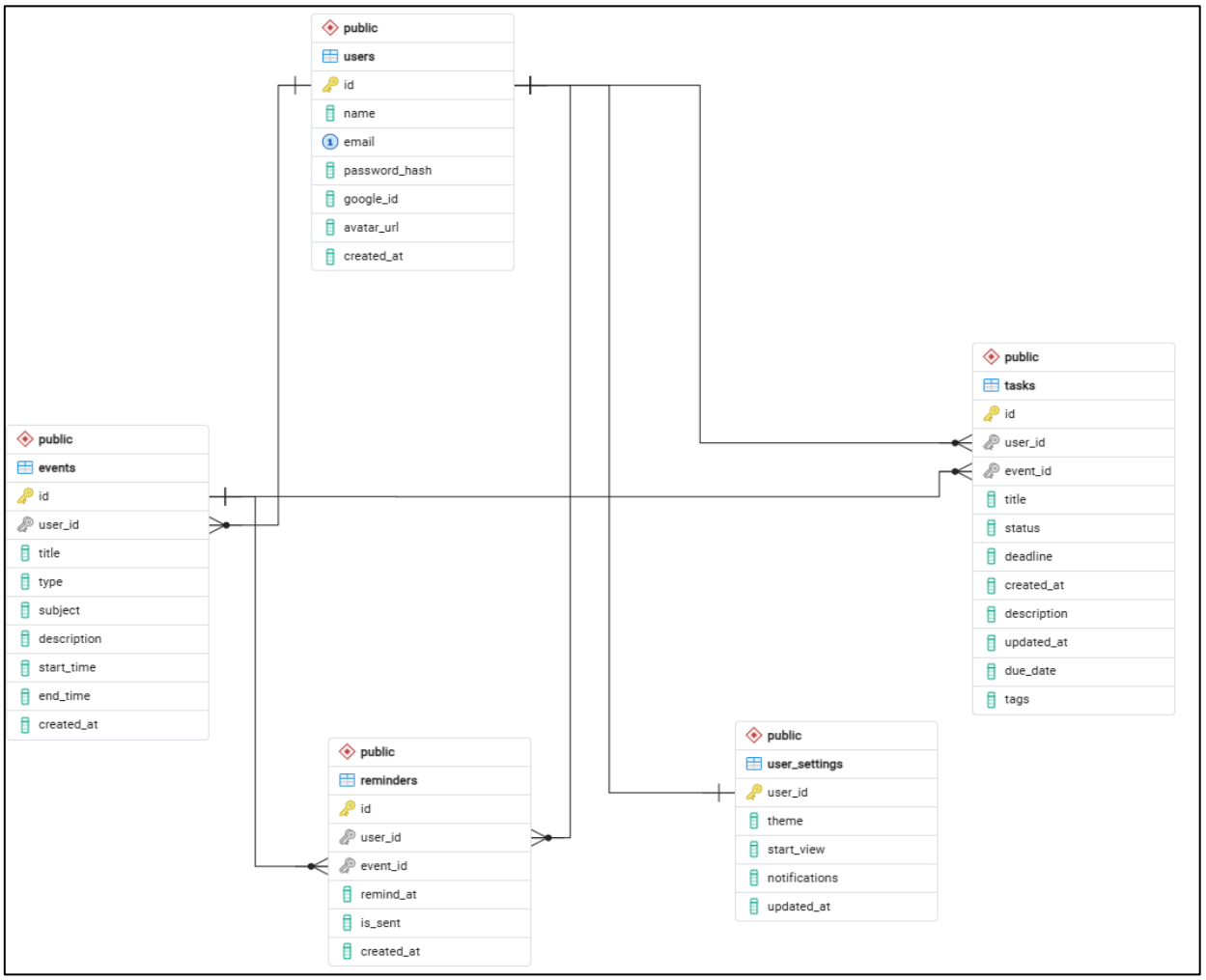


Рисунок 3.2 – ER-діаграма бази даних Study Planner

Один користувач може мати багато задач, подій і нагадувань, але кожен такий запис належить тільки одному користувачу. Налаштування мають зв’язок один до одного з користувачем, оскільки для кожного акаунту зберігається один набір персональних параметрів. Така структура дозволяє відокремити дані різних користувачів і не змішувати їх між собою. Крім того, зв’язки між таблицями допомагають підтримувати цілісність даних під час

створення, редагування або видалення записів. Ключові атрибути бази даних продемонстровано в таблиці 3.6.

Таблиця 3.6 – Ключові атрибути сутностей

Таблиця	**Поле**	**Призначення**
users	id	Унікальний ідентифікатор користувача
users	email	Електронна пошта для входу до системи
users	password_hash	Хеш пароля користувача
tasks	user_id	Зв’язок задачі з користувачем
tasks	status	Поточний стан виконання задачі
tasks	due_date / deadline	Дата виконання або дедлайн задачі
tasks	tags	Масив тегів для фільтрації та пошуку
events	start_time, end_time	Час початку та завершення календарної події
reminders	remind_at	Час нагадування про подію
user_settings	theme, start_view, notifications	Персональні параметри роботи із застосунком

Крім ER-діаграми, було побудовано діаграму класів (див. рис. 3.3). Вона описує ті самі сутності з погляду об’єктної моделі та показує, які дані використовуються в логіці застосунку.

На діаграмі відображено основні класи, що відповідають користувачу, задачі, календарній події, нагадуванню та персональним налаштуванням. Це допомагає краще зрозуміти, як структура бази даних пов’язується з програмною реалізацією системи.

Крім того, діаграма класів показує не тільки атрибути сутностей, а й логічні зв’язки між ними.

Діаграма класів допомагає пов’язати структуру бази даних із програмною логікою. Наприклад, задача має назву, опис, статус, дедлайн і теги, а користувач пов’язаний із власними задачами, подіями та налаштуваннями.

Такий опис полегшує подальшу реалізацію моделей даних і контролерів серверної частини.

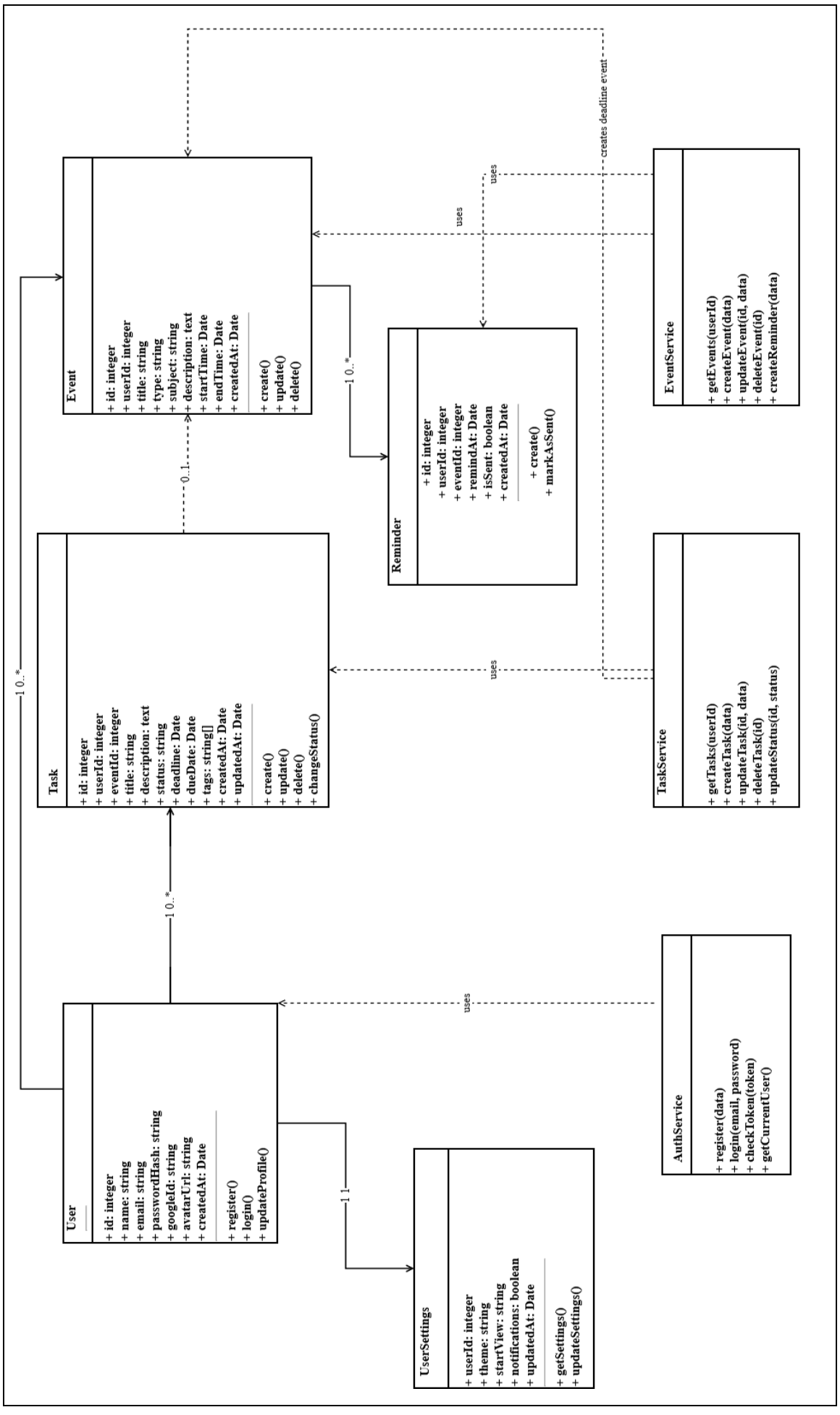


Рисунок 3.3 – Діаграма класів Study Planner

3.4 Проєктування поведінки системи

Поведінка Study Planner описує, як саме користувач взаємодіє із застосунком після входу в систему. Основний сценарій пов’язаний зі створенням і подальшим виконанням навчальних задач. Студент додає задачу, встановлює дедлайн, за потреби додає теги, переглядає задачу у списку або календарі, редагує її та змінює статус під час виконання.

Для того, щоб показати як працювати із задачею, було зроблено діаграму діяльності, яка показана на рисунку 3.4.

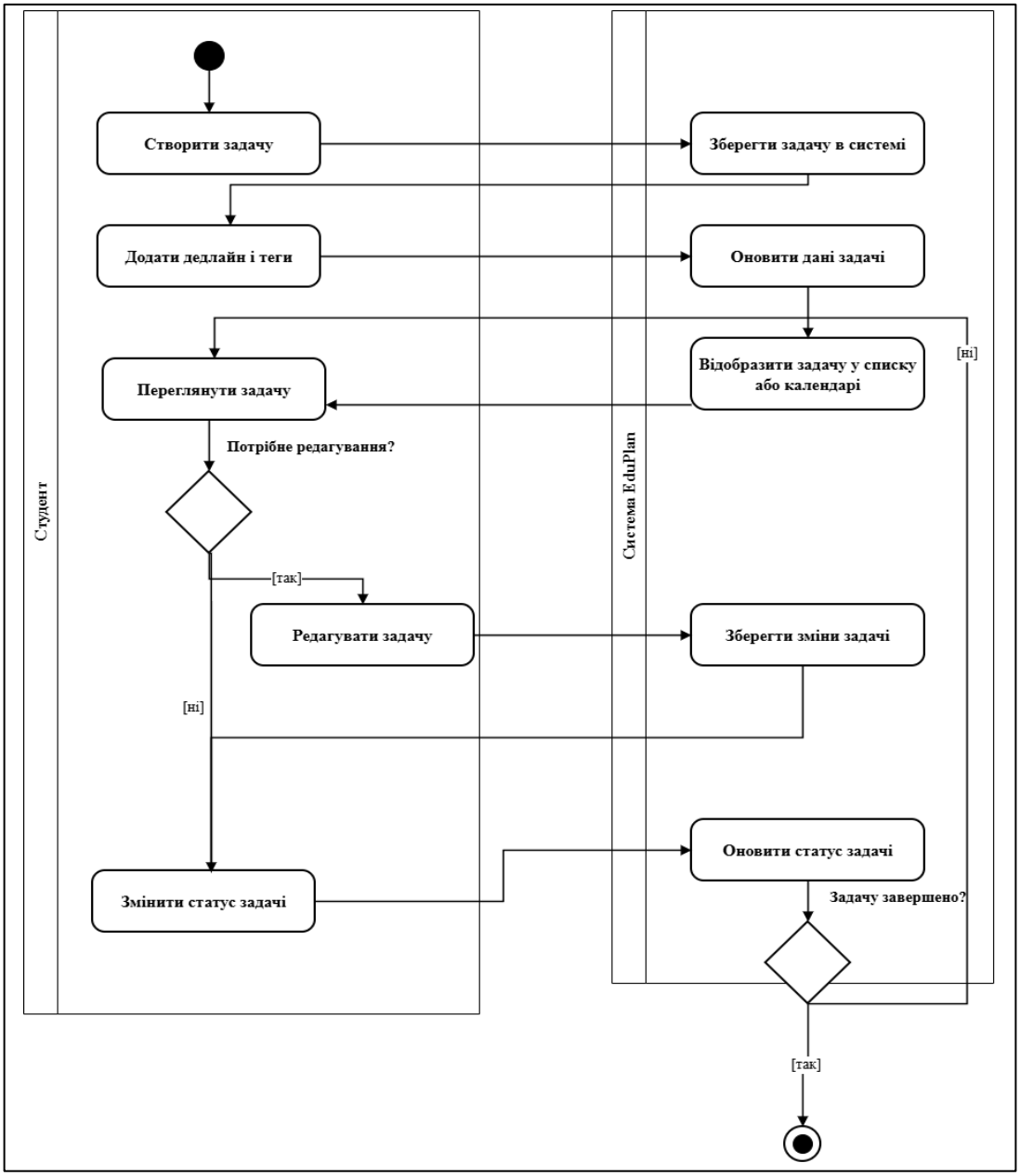


Рисунок 3.4 – Діаграма діяльності роботи із задачею

Діаграма діяльності показує життєвий цикл задачі в системі. Після створення задача не залишається статичним записом: її можна редагувати, переносити, уточнювати тегами та переводити в інший статус. Після завершення задача враховується у статистиці виконання.

Окремо було описано послідовність створення задачі. Цей сценарій є важливим, оскільки він демонструє взаємодію між користувачем, клієнтською частиною, сервером і базою даних. Процес показано на рисунку 3.5.

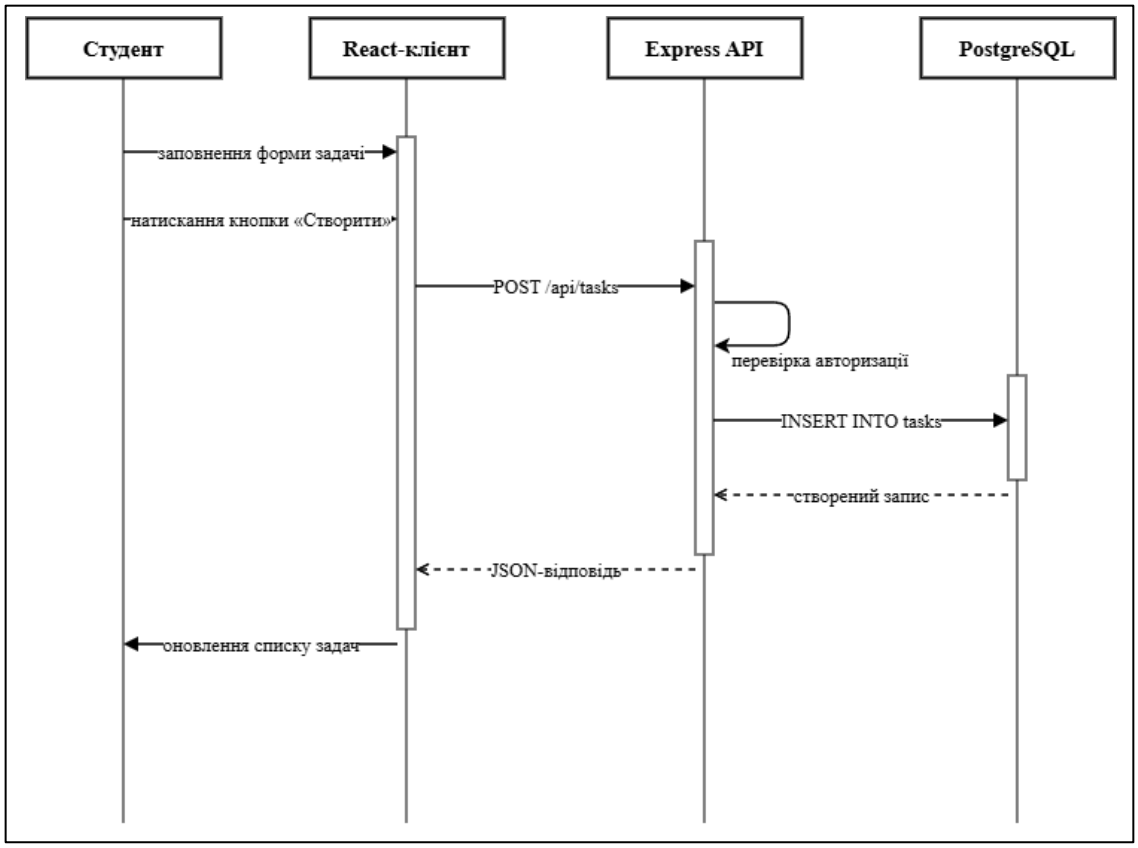


Рисунок 3.5 – Діаграма послідовності створення задачі

З діаграми видно, що користувач заповнює форму в React-інтерфейсі, після чого клієнтська частина надсилає POST-запит на сервер. Сервер перевіряє JWT-токен, визначає користувача, зберігає задачу в PostgreSQL і повертає відповідь у форматі JSON. Після цього інтерфейс оновлює список задач без перезавантаження сторінки.

Для задач також важливо описати стани, у яких вони можуть перебувати. У Study Planner використано три основні стани: pending, in_progress і

completed. В інтерфейсі вони відповідають задачам, які очікують виконання, перебувають у роботі або вже виконані (див. рис. 3.6).

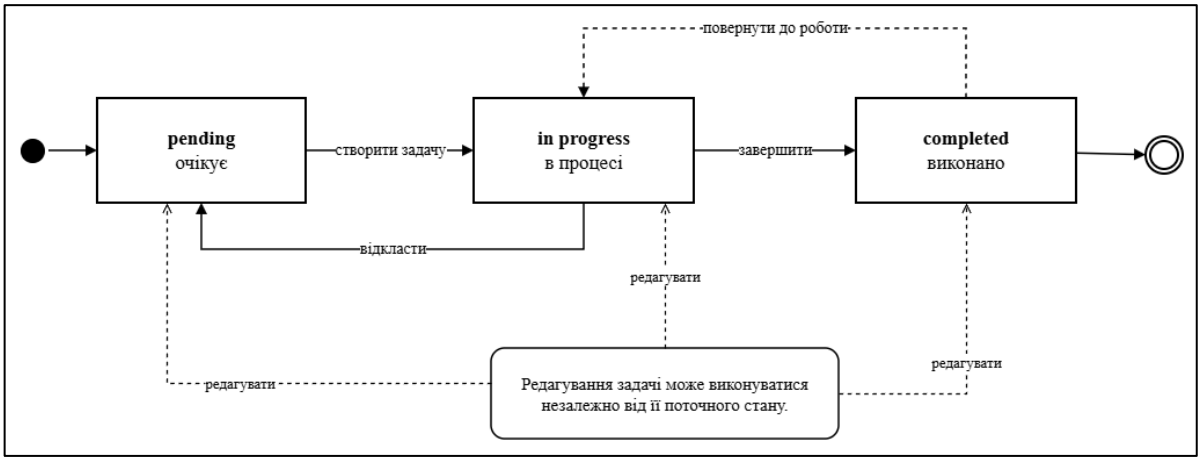


Рисунок 3.6 – Діаграма станів задачі Study Planner

Діаграма станів показує, що користувач може перевести задачу з очікування в роботу, а потім позначити її як виконану. Якщо потрібно, задачу можна відредагувати або повернути до іншого стану. Саме зміна статусу використовується під час формування статистики.

Механізм планування навчальної діяльності поєднує задачі, календарні події та нагадування. Якщо задача має дедлайн, вона може бути відображена в календарі.

Події зберігаються з датою та часом, а нагадування пов’язуються з конкретною подією. Статистика формується на основі статусів задач і дат виконання: система може показати кількість активних, виконаних і прострочених задач.

3.5 Проєктування загальної структури та компонентів системи

Після визначення вимог до Study Planner було спроєктовано загальну структуру системи. Для цього вебзастосунку обрано клієнт-серверну архітектуру, оскільки вона дозволяє розділити інтерфейс користувача,

серверну логіку та збереження даних. Такий підхід є зручним для застосунку, у якому студент працює через браузер, а його задачі, календарні події, нагадування та налаштування зберігаються в базі даних.

Загальна структура Study Planner складається з трьох основних частин: клієнтської частини, серверної частини та бази даних. Основні компоненти вебзастосунку наведено у таблиці 3.7.

Таблиця 3.7 – Основні компоненти вебзастосунку Study Planner

Компонент	**Призначення**	**Технологія**
Клієнтська частина	Відображення інтерфейсу, форм, задач, календаря, статистики та налаштувань	React
API-сервіси клієнта	Надсилання HTTP-запитів до серверу та обробка відповідей	Axios, REST API
Серверна частина	Обробка запитів, авторизація, перевірка даних, робота з базою даних	Node.js, Express.js
Middleware авторизації	Перевірка JWT-токена та визначення поточного користувача	JSON Web Token
База даних	Збереження користувачів, задач, подій, нагадувань і налаштувань	PostgreSQL

Клієнтська частина відповідає за відображення сторінок, форм, списків задач, календаря, статистики та налаштувань. Серверна частина приймає запити від клієнта, перевіряє авторизацію користувача, обробляє дані та взаємодіє з базою даних. PostgreSQL використовується для збереження користувачів, задач, календарних подій, нагадувань і персональних налаштувань.

Для наочного представлення структури системи було побудовано діаграму компонентів вебзастосунку Study Planner. Вона показує, з яких основних частин складається система та як між ними відбувається обмін даними. Продемонстровано на рисунку 3.7.

На діаграмі видно, що клієнтська частина не звертається до бази даних напряму. Усі запити проходять через REST API та серверну частину. Це дозволяє перевіряти авторизацію, контролювати правильність вхідних даних і не відкривати базу даних для прямої взаємодії з браузером користувача.

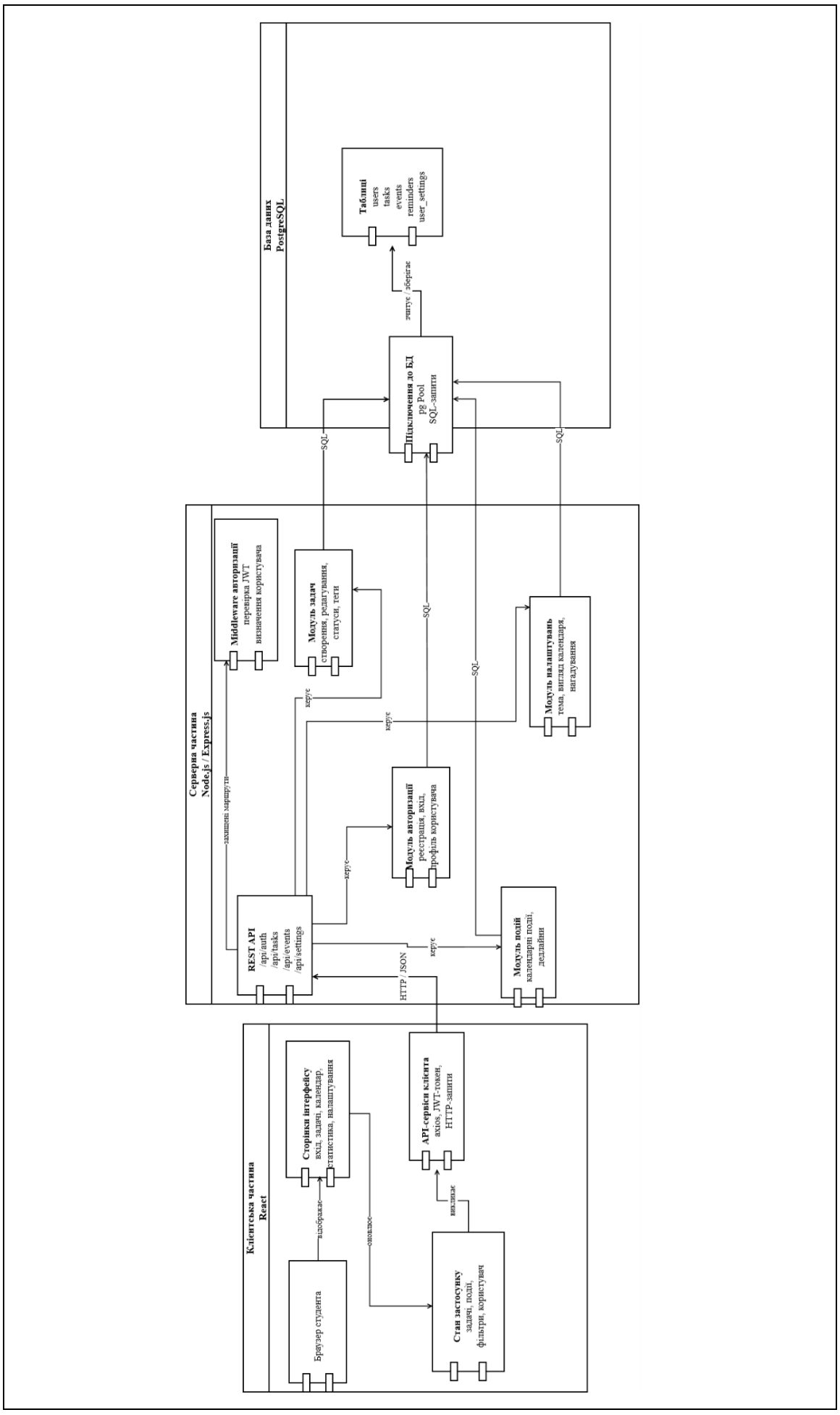


Рисунок 3.7 – Діаграма компонентів вебзастосунку Study Planner

Клієнтська частина реалізована на React. Вона містить сторінку входу та реєстрації, головну панель, сторінку задач, календар, сторінку статистики та сторінку налаштувань. Через ці сторінки студент може створювати задачі, задавати дедлайни, змінювати статуси, додавати календарні події, переглядати статистику та змінювати персональні параметри застосунку.

Серверна частина реалізована на Node.js та Express.js. Вона містить маршрути для авторизації, роботи із задачами, календарними подіями та налаштуваннями. Сервер також відповідає за перевірку JWT-токена, завдяки чому користувач отримує доступ тільки до власних даних. Це важливо, оскільки Study Planner працює з персональною навчальною інформацією студента.

Взаємодія між клієнтською та серверною частинами виконується через REST API. Клієнтська частина надсилає HTTP-запити до сервера, а сервер повертає відповідь у форматі JSON. Наприклад, під час створення задачі React-клієнт надсилає POST-запит на маршрут /api/tasks.

Сервер перевіряє авторизацію, зберігає задачу в PostgreSQL і повертає створений запис назад до клієнта. Такий підхід дозволяє розділити відображення інтерфейсу та обробку даних.

3.6 Проєктування користувацького інтерфейсу

Користувацький інтерфейс Study Planner має бути простим і зрозумілим, тому його структура побудована навколо основних дій студента.

Після входу користувач переходить до головної панелі, з якої може відкрити список задач, календар, статистику або налаштування.

Навігація між сторінками має бути очевидною, щоб студент міг швидко перейти до потрібної функції.

Основні сторінки інтерфейсу розробленої інформаційної системи наведено у таблиці 3.8.

Таблиця 3.8 – Основні сторінки інтерфейсу Study Planner

Сторінка	**Призначення**
Сторінка входу та реєстрації	Введення електронної пошти, пароля та створення нового акаунту.
Головна панель	Короткий огляд найближчих задач, дедлайнів, подій і поточного прогресу.
Сторінка задач	Створення, редагування, видалення задач, зміна статусів і робота з тегами.
Календар	Відображення задач із дедлайнами та окремих календарних подій за датами.
Сторінка статистики	Перегляд кількості активних, виконаних і прострочених задач.
Сторінка налаштувань	Зміна теми інтерфейсу, вигляду календаря та параметрів нагадувань.

Адаптивність інтерфейсу важлива для практичного використання, оскільки студент може відкривати застосунок не лише на комп’ютері, а й на ноутбуці або планшеті. Тому елементи інтерфейсу повинні коректно змінювати розміри, а основні дії мають залишатися доступними без зайвих переходів.

Зручність користування забезпечується тим, що кожна сторінка відповідає конкретній групі дій. Задачі редагуються на сторінці задач, події переглядаються в календарі, статистика винесена окремо, а персональні параметри розміщені в налаштуваннях. Завдяки цьому користувач не витрачає час на пошук потрібної функції.

3.7 Висновки до розділу 3

У третьому розділі було виконано проєктування вебзастосунку Study Planner. Спочатку було сформульовано функціональні та нефункціональні вимоги, визначено основні сценарії використання і сформовано технічне завдання. Це дозволило чітко описати, які можливості повинна мати система та які обмеження потрібно врахувати під час реалізації.

Після цього було спроєктовано загальну структуру системи, модель даних, поведінку основних процесів, прецеденти використання та користувацький інтерфейс. Для пояснення архітектури й логіки роботи застосунку використано діаграму компонентів, діаграму розгортання, ER-діаграму, діаграму класів, діаграму діяльності, діаграму послідовності, діаграму станів і діаграму варіантів використання.

Отже, у цьому розділі було сформовано проєктну основу Study Planner. Вона показує, як система має працювати на рівні користувацьких сценаріїв, програмних компонентів, бази даних і взаємодії між частинами застосунку. На основі цього проєктування можна переходити до опису реалізації та тестування.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ STUDY PLANNER

4.1 Середовище розробки та використані технології

Після завершення проєктування було виконано практичну реалізацію вебзастосунку Study Planner. Для розроблення використано середовище Visual Studio Code, оскільки воно зручне для роботи з JavaScript-проєктами, має підтримку розширень, термінал і засоби навігації по файлах. Запуск і перевірка застосунку виконувалися локально, а для контейнеризованого запуску передбачено Docker Compose.

Клієнтська частина реалізована мовами HTML, CSS і JavaScript із використанням бібліотеки React. Для збирання проєкту використано Vite, для переходів між сторінками – React Router DOM, для календарного перегляду – FullCalendar, а для HTTP-запитів до серверної частини – Axios [18]. Серверна частина реалізована на Node.js та Express.js. Для роботи з базою даних PostgreSQL використано бібліотеку pg, для авторизації – jsonwebtoken, а для хешування паролів – bcryptjs [19]. Приклад обраних засобів наведено в таблиці 4.1.

Таблиця 4.1 – Середовище розробки та використані технології

Засіб	**Призначення**
IDE JET BRAIN	Редактор коду для роботи з клієнтською та серверною частинами.
JavaScript	Основна мова програмування клієнтської та серверної частини.
HTML, CSS	Розмітка та стилізація сторінок вебзастосунку.
React	Побудова динамічного інтерфейсу користувача.
React Router DOM	Організація переходів між сторінками застосунку.

Продовження таблиці 4.1

Засіб	**Призначення**
FullCalendar	Відображення задач і подій у календарному вигляді.
Tailwind CSS	Оформлення інтерфейсу та адаптивних елементів.
Vite	Запуск і збирання клієнтської частини.
Node.js, Express.js	Реалізація серверної частини та REST API.
Axios	Надсилання HTTP-запитів із клієнтської частини.

4.2 Реалізація серверної частини (Backend)

Серверна частина Study Planner реалізована як Express-застосунок. Вона приймає запити від клієнтської частини, перевіряє дані, виконує авторизацію користувача та звертається до PostgreSQL. У серверній частині виділено маршрути для авторизації, задач, календарних подій і налаштувань.

REST API побудовано за ресурсним принципом. Окремі маршрути відповідають за реєстрацію та вхід користувача, отримання задач, створення задачі, редагування, видалення, зміну статусу, роботу з подіями та оновлення налаштувань.

Захищені маршрути використовують middleware авторизації, який перевіряє JWT-токен і визначає поточного користувача. Основні маршрути наведено в таблиці 4.2, фрагмент коду – на рисунку 4.1.

Таблиця 4.2 – Основні маршрути REST API Study Planner

Метод	**Маршрут**	**Призначення**	**Результат**
GET	/api/health	Перевірка роботи серверної частини	Статус сервера
POST	/api/auth/ register	Реєстрація нового користувача	Створений акаунт

Продовження таблиці 4.2

Метод	**Маршрут**	**Призначення**	**Результат**
POST	/api/auth/login	Вхід користувача	JWT-токен
GET	/api/tasks	Отримання задач користувача	Список задач
POST	/api/tasks	Створення задачі	Створена задача
PUT	/api/tasks/:id	Редагування задачі	Оновлена задача
DELETE	/api/tasks/:id	Видалення задачі	Повідомлення про видалення
PATCH	/api/tasks/:id/ status	Зміна статусу задачі	Задача з новим статусом
GET	/api/events	Отримання календарних подій	Список подій
POST	/api/events	Створення календарної події	Створена подія
PUT	/api/events/:id	Редагування календарної події	Оновлена подія
DELETE	/api/events/:id	Видалення календарної події	Повідомлення про видалення
GET	/api/settings	Отримання налаштувань	Параметри користувача
PUT/ PATCH	/api/settings	Оновлення налаштувань	Оновлені параметри

```
import { Router } from 'express'
import {
    getTasks,
    createTask,
    updateTask,
    deleteTask,
    updateTaskStatus,
} from '../controllers/taskController.js'
import authMiddleware from '../middleware/authMiddleware.js'

const router = Router()

router.use(authMiddleware)

router.get('/', getTasks)
router.post('/', createTask)
router.put('/:id', updateTask)
router.delete('/:id', deleteTask)
router.patch('/:id/status', updateTaskStatus)

export default router
```

Рисунок 4.1 – Маршрути для роботи із задачами

У наведеному фрагменті видно, що перед усіма маршрутами задач використовується authMiddleware. Це означає, що отримати, створити або змінити задачу може тільки авторизований користувач.

На рисунку 4.2 наведено приклад коду для створення задачі.

```
export async function createTask(req, res) {
    const { title, description, due_date, tags } = req.body

    if (!title?.trim()) {
        return res.status(400).json({
            message: 'Введіть назву завдання',
        })
    }

    const sql = `INSERT INTO tasks
        (user_id, title, description, status, due_date, tags)
        VALUES ($1, $2, $3, 'pending', $4, $5)
        RETURNING *`

    const values = [
        req.user.id,
        title.trim(),
        description || '',
        due_date || null,
        tags || [],
    ]

    const { rows } = await pool.query(sql, values)
    res.status(201).json(rows[0])
}
```

Рисунок 4.2 – Скорочений приклад створення задачі на сервері

Цей фрагмент показує спрощену логіку створення задачі. Сервер отримує дані з тіла запиту, перевіряє назву задачі, прив'язує запис до поточного користувача через req.user.id і зберігає задачу в таблиці tasks. У повній реалізації додатково обробляються дедлайни та зв'язок задачі з календарною подією.

4.3 Реалізація клієнтської частини (Frontend)

Клієнтська частина Study Planner реалізована як React-застосунок. Вона відповідає за сторінки інтерфейсу, форми введення, списки задач, календарний перегляд, статистику та налаштування. Для переходів між сторінками використовується маршрутизація, а для обміну даними із сервером створено окремі API-сервіси.

Після відкриття застосунку користувач переходить на сторінку входу або реєстрації. Зовнішній вигляд цієї сторінки позначено на рисунку 4.3. Після успішної авторизації користувач переходить до головної панелі, яку позначено на рисунку 4.4.

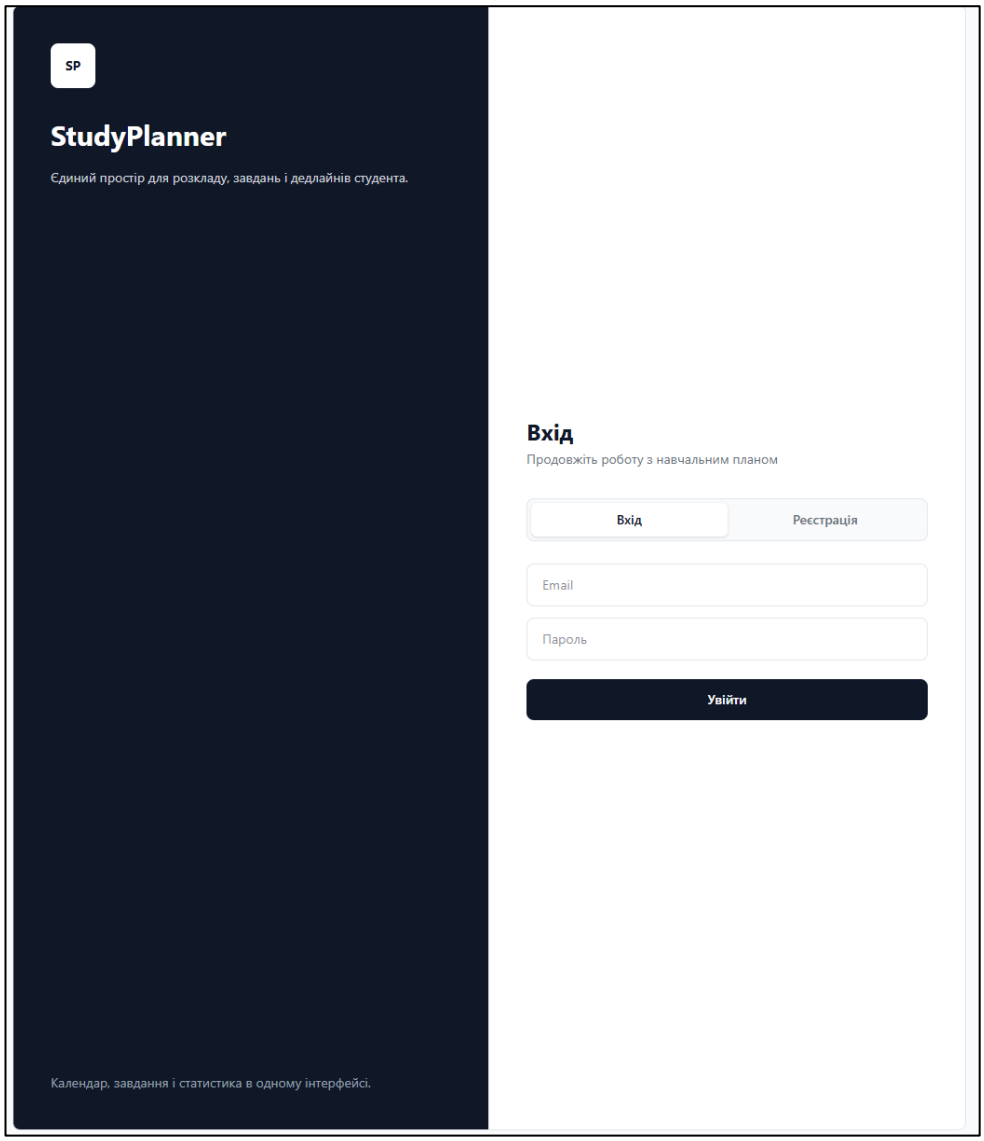


Рисунок 4.3 – Сторінка входу та реєстрації Study Planner

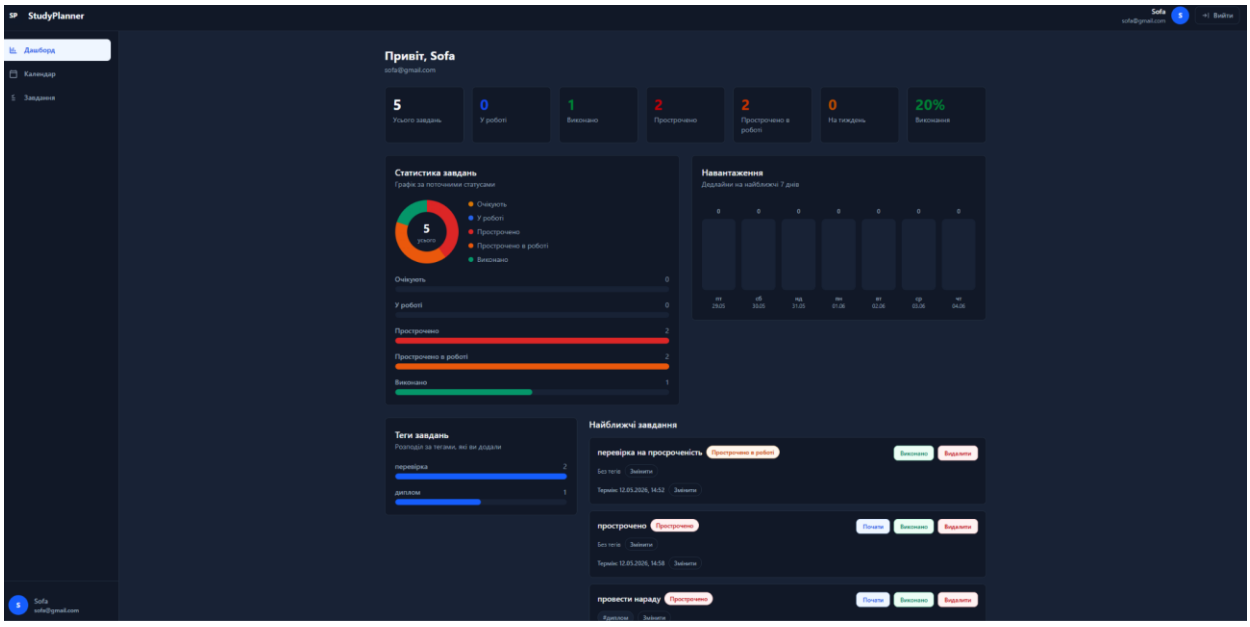


Рисунок 4.4 – Головна панель Study Planner

Основна робота із задачами виконується на сторінці задач. Користувач може створювати нові записи, редагувати назву, опис, дедлайн і теги, а також змінювати статус виконання. Вигляд цієї сторінки позначено на рисунку 4.5.

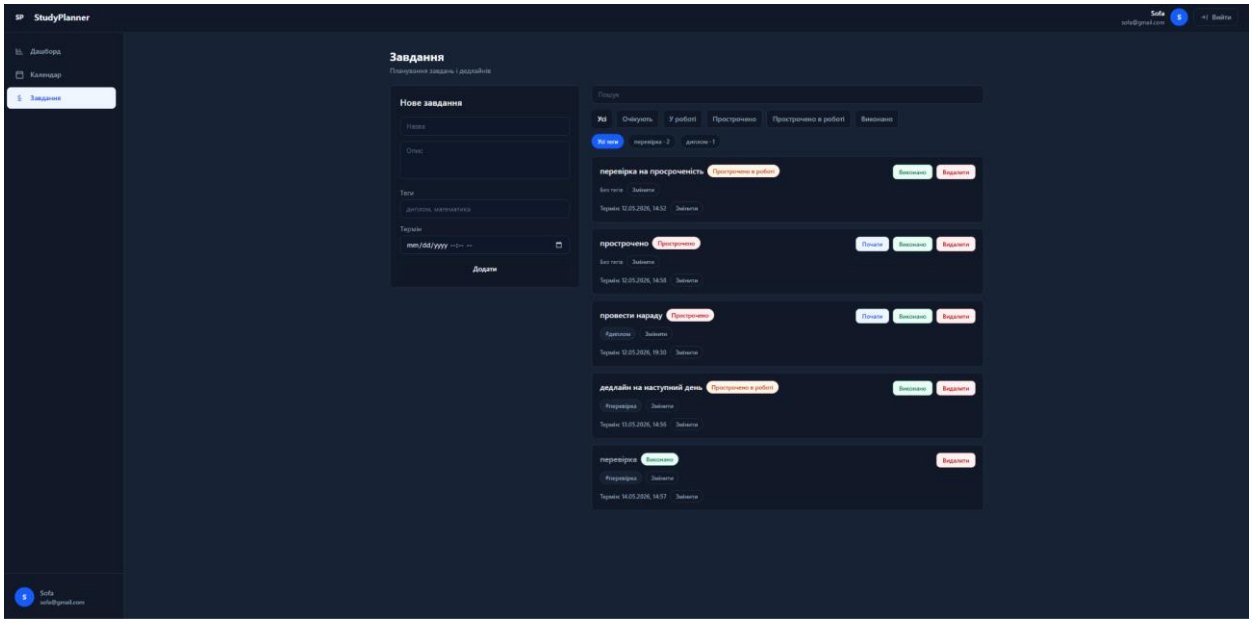


Рисунок 4.5 – Сторінка задач Study Planner

Календарний перегляд дає змогу бачити задачі та події за датами. Він потрібний для оцінки навчального навантаження протягом тижня або місяця. Вигляд календаря позначено на рисунку 4.6.

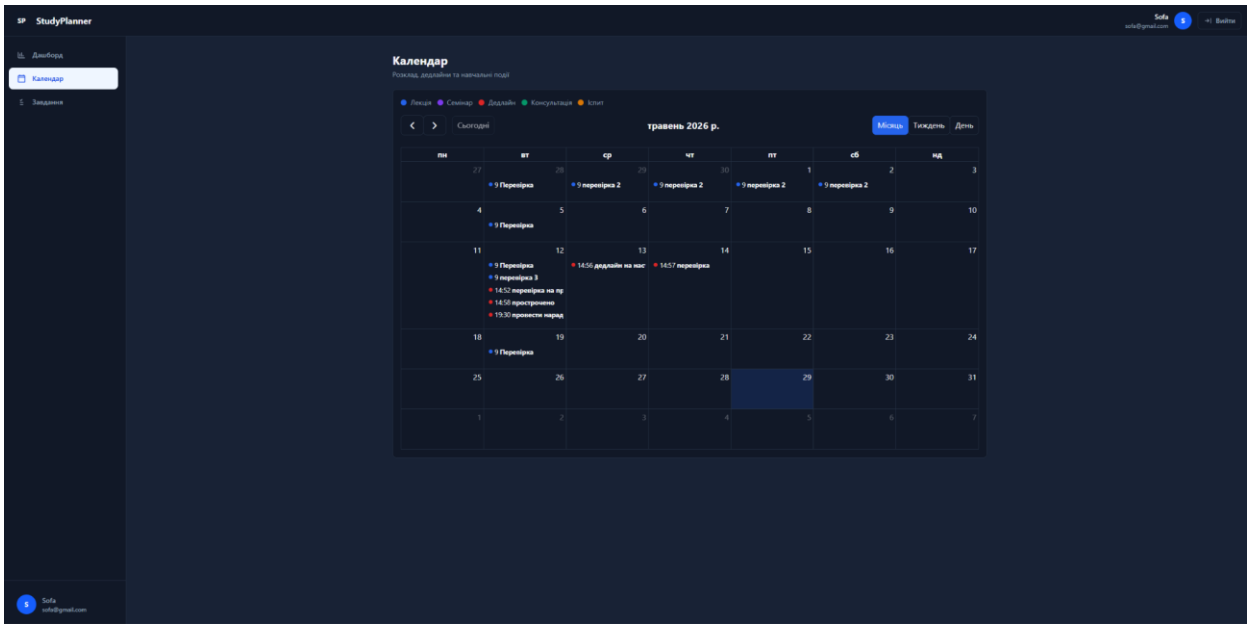


Рисунок 4.6 – Календарний перегляд Study Planner

Окрему сторінку передбачено для налаштувань. На сторінці налаштувань може змінити персональні параметри роботи застосунку. Цей екран позначено на рисунку 4.7. На рисунку 4.8 наведено фрагмент коду для роботи із задачами.

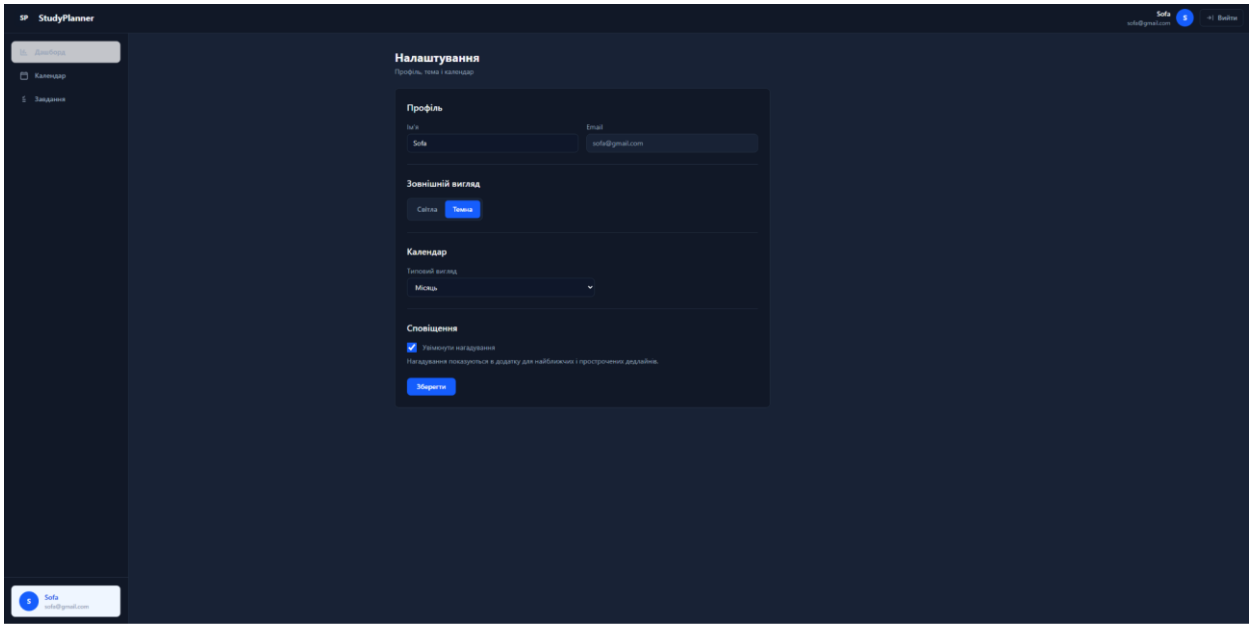


Рисунок 4.7 – Сторінка налаштувань Study Planner

```
import api from './api.js'

export const getTasks = () => api.get('/tasks')

export const createTask = (data) => api.post('/tasks', data)

export const updateTask = (id, data) => api.put(`/tasks/${id}`, data)

export const deleteTask = (id) => api.delete(`/tasks/${id}`)

export const updateStatus = (id, status) =>
    api.patch(`/tasks/${id}/status`, { status })
```

Рисунок 4.8 – API-сервіс клієнта для роботи із задачами

API-сервіс клієнта приховує деталі HTTP-запитів від компонентів інтерфейсу.

Компонент сторінки задач викликає потрібну функцію, а сервіс надсилає запит на сервер і повертає результат. Це робить код клієнтської частини простішим для підтримки.

4.4 Реалізація основного функціоналу системи

Основний функціонал Study Planner пов’язаний із навчальними задачами, календарними подіями, нагадуваннями, фільтрацією та статистикою. Під час реалізації ці можливості були розділені між клієнтською та серверною частинами. Інтерфейс приймає дії користувача, а серверна частина зберігає й повертає дані. Такий поділ дозволяє зробити застосунок більш структурованим і спростити підтримку окремих модулів. Кожна функція реалізована так, щоб користувач міг швидко виконати основні дії без зайвих переходів між сторінками.

Основні функції вебзастосунку представлено у таблиці 4.3.

Таблиця 4.3 – Реалізація основного функціоналу Study Planner

Функція	**Реалізація**
Створення задач	Користувач заповнює форму, після чого задача зберігається в PostgreSQL і з’являється у списку.
Редагування задач	Користувач змінює назву, опис, дедлайн або теги, а сервер оновлює відповідний запис.
Видалення задач	Неактуальна задача видаляється з бази даних за умови, що вона належить поточному користувачу.
Планування навчального часу	Задачі з дедлайнами та календарні події відображаються в календарі.
Нагадування	Для подій можуть зберігатися записи з часом нагадування.
Фільтрація та пошук	Користувач може відбирати задачі за статусом, тегом, датою або текстом.
Статистика	Система показує кількість активних, виконаних і прострочених задач.

Окрему увагу приділено перевірці належності даних користувачу. Під час запитів до задач, подій або налаштувань сервер використовує ідентифікатор авторизованого користувача. Завдяки цьому студент не може отримати або змінити чужі записи.

4.5 Тестування програмного забезпечення

Тестування проводилося для перевірки того, чи відповідає реалізований вебзастосунок вимогам, сформульованим під час проєктування. Перевірялися основні сценарії роботи користувача, коректність API, захист персональних даних і поведінка інтерфейсу під час звичайних та помилкових дій.

Для Study Planner найбільше значення має функціональне та інтеграційне тестування. Функціональне тестування перевіряє окремі можливості: реєстрацію, вхід, створення задачі, фільтрацію, календар і статистику. Інтеграційне тестування показує, чи правильно взаємодіють React-клієнт, Express API та PostgreSQL. Результати тестування наведено у таблиці 4.4.

Таблиця 4.4 – Результати тестування Study Planner

№	Сценарій тестування	Вхідні дані	Очікуваний результат	Статус
1	Реєстрація з коректними даними	Ім’я, email, пароль	Створено нового користувача	Пройдено
2	Реєстрація з порожніми полями	Порожня форма	Система показує помилку	Пройдено
3	Вхід з правильними даними	Коректний email і пароль	Користувач переходить до застосунку	Пройдено
4	Вхід з неправильним паролем	Коректний email, неправильний пароль	Доступ не надається	Пройдено
5	Створення задачі	Назва, опис, дедлайн, теги	Задача збережена і показана у списку	Пройдено
6	Редагування задачі	Нові дані задачі	Зміни збережено в базі даних	Пройдено
7	Видалення задачі	Ідентифікатор задачі	Задача зникає зі списку	Пройдено
8	Зміна статусу задачі	Статус in_progress або completed	Статус оновлено	Пройдено
9	Додавання календарної події	Назва, дата, час	Подія відображається в календарі	Пройдено

Продовження таблиці 4.4

№	Сценарій тестування	Вхідні дані	Очікуваний результат	Статус
10	Пошук задачі	Текст запиту	Відображаються відповідні задачі	Пройдено
11	Фільтрація за статусом	Обраний статус	Показано задачі з потрібним статусом	Пройдено
12	Перегляд статистики	Наявні задачі	Показано кількість задач за станами	Пройдено
13	Доступ без авторизації	Запит без JWT-токена	Сервер повертає помилку доступу	Пройдено

4.6 Опис результатів тестування

Результати тестування показали, що основні функції Study Planner працюють коректно. Користувач може зареєструватися, увійти до системи, створити задачу, змінити її статус, додати календарну подію, використати пошук і фільтрацію та переглянути статистику.

Під час перевірки було виявлено кілька типових помилок. Наприклад, задачі могли некоректно відображатися в календарі через різний формат дат у клієнтській та серверній частинах. Після приведення дат до єдиного формату календар почав правильно показувати дедлайни.

Також було перевірено оновлення списку задач після зміни статусу. Якщо інтерфейс не оновлював список одразу, користувач міг бачити застарілі дані. Для цього після операцій створення, редагування або зміни статусу виконується оновлення стану застосунку.

Окремо перевірявся доступ до захищених маршрутів. Запити без токена не повинні повертати персональні дані користувача. Після перевірки middleware авторизації було підтверджено, що сервер не повертає задачі, події або налаштування без коректної авторизації. Виявлені помилки та способи виправлення показані в таблиці 4.5.

Таблиця 4.5 – Виявлені помилки та способи виправлення

Проблема	**Причина**	**Виправлення**
Некоректне відображення задач у календарі	Використання різних форматів дати	Дати приведено до єдиного формату
Фільтр не оновлював список після зміни статусу	Стан клієнтської частини не оновлювався одразу	Після зміни даних виконується оновлення списку
Статус задачі не зберігався після перезавантаження	Оновлення виконувалося тільки на клієнті	Зміну статусу перенесено на серверний PATCH-запит
Запит без авторизації міг звертатися до захищених даних	Недостатня перевірка маршруту	Додано middleware авторизації для захищених маршрутів

4.7 Інструкція користувача

Для запуску системи локально потрібно встановити залежності клієнтської та серверної частин, підготувати базу даних PostgreSQL і запустити сервер. Якщо використовується Docker Compose, основні сервіси можуть запускатися разом: база даних, серверна частина та клієнтська частина.

Після запуску користувач відкриває застосунок у браузері. Спочатку потрібно зареєструватися або увійти до вже створеного акаунту. Після входу відкривається головна панель, на якій можна побачити коротку інформацію про задачі, дедлайни й поточний стан виконання.

Щоб створити завдання, користувач переходить на сторінку задач, натискає кнопку додавання та заповнює форму. У формі вказуються назва, опис, дедлайн і теги. Після збереження задача з’являється у списку та, за наявності дедлайну, може відображатися в календарі.

Для роботи з календарем користувач переходить на відповідну сторінку. Там можна переглядати навчальні задачі й події за датами. Якщо потрібно знайти конкретну задачу, можна скористатися пошуком або фільтрацією за статусом, тегом чи датою.

Статистика доступна на окремій сторінці. Вона допомагає оцінити, скільки задач уже виконано, скільки залишилося активними та які задачі прострочені. У налаштуваннях користувач може змінити тему інтерфейсу, вигляд календаря та параметри нагадувань.

4.8 Висновки до розділу 4

У четвертому розділі було описано практичну реалізацію та тестування вебзастосунку Study Planner. Було розглянуто середовище розробки, використані технології, реалізацію серверної та клієнтської частин, основні маршрути REST API, роботу з базою даних, авторизацію та основний функціонал системи.

Тестування підтвердило працездатність основних сценаріїв використання: реєстрації, входу, створення й редагування задач, зміни статусу, роботи з календарем, пошуку, фільтрації, статистики та налаштувань. Особливу увагу було приділено перевірці авторизації та доступу тільки до власних даних користувача.

Отже, реалізований вебзастосунок відповідає поставленим вимогам і може використовуватися студентом для особистого планування навчальної діяльності. Study Planner поєднує задачі, дедлайни, календарні події, нагадування, статистику та персональні налаштування в одному середовищі.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено вебзастосунок Study Planner для планування навчальних задач і календарних подій студента. Основна ідея роботи полягала в тому, щоб створити зручний інструмент для особистої організації навчання, де студент може зберігати задачі, дедлайни, події, теги, статуси, нагадування, статистику та персональні налаштування в одному середовищі.

Під час аналізу предметної області було визначено проблему неструктурованого навчального планування. Студенти часто використовують одразу кілька різних інструментів: нотатки, месенджери, календарі, файли або списки справ. Через це важко швидко оцінити навчальне навантаження, побачити найближчі дедлайни та контролювати виконання задач. Аналіз існуючих сервісів показав, що Google Calendar, Todoist, Notion, Trello та Microsoft To Do мають корисні можливості, але не завжди зручно поєднують задачі, дедлайни, календар і статистику саме для щоденного навчального планування студента.

Для реалізації Study Planner було обґрунтовано вибір технологій. Клієнтську частину створено з використанням React, оскільки ця бібліотека добре підходить для побудови динамічного інтерфейсу, роботи зі станом, формами, календарем і переходами між сторінками. Серверну частину реалізовано на Node.js та Express.js, що дозволило створити REST API для обробки запитів, авторизації користувача та взаємодії з базою даних. Для збереження даних використано PostgreSQL, оскільки система працює зі структурованими даними та зв’язками між користувачами, задачами, подіями, нагадуваннями й налаштуваннями.

У процесі проєктування було визначено функціональні та нефункціональні вимоги до системи, описано основні сценарії використання, спроєктовано загальну архітектуру, модель даних, поведінку системи та

прецеденти використання. Study Planner побудовано за клієнт-серверною архітектурою: користувач працює з React-інтерфейсом, клієнтська частина надсилає запити до Express API, сервер перевіряє авторизацію та звертається до PostgreSQL. Такий підхід дозволяє розділити інтерфейс, бізнес-логіку та збереження даних.

Практична реалізація охопила основні сторінки вебзастосунку: сторінку входу та реєстрації, головну панель, сторінку задач, календар, сторінку статистики та сторінку налаштувань. Користувач може створювати задачі, редагувати їх, видаляти, змінювати статус, додавати дедлайни й теги, переглядати задачі в календарі, користуватися пошуком і фільтрацією, а також переглядати статистику виконання.

Окрему увагу було приділено авторизації та персональному доступу до даних. Кожен користувач після входу в систему отримує доступ тільки до власних задач, календарних подій, нагадувань і налаштувань. Для цього в серверній частині використовується перевірка JWT-токена, а записи в базі даних пов’язані з ідентифікатором конкретного користувача.

Тестування підтвердило працездатність основних функцій Study Planner. Було перевірено реєстрацію, вхід до системи, створення, редагування й видалення задач, зміну статусу, роботу календаря, пошук, фільтрацію, статистику та доступ до захищених маршрутів. Під час тестування були виявлені типові помилки, зокрема проблеми з форматом дат, оновленням списку після зміни статусу та перевіркою доступу. Після виправлення цих недоліків основні сценарії роботи виконуються коректно.

Практичне значення розробленого вебзастосунку полягає в тому, що Study Planner може використовуватися студентом для щоденної організації навчальної діяльності. Застосунок допомагає не тримати задачі в різних місцях, а зберігати їх у єдиній системі, бачити дедлайни, планувати події та оцінювати власний прогрес.

Отже, поставлену мету дипломної роботи досягнуто. Було розроблено клієнт-серверний вебзастосунок Study Planner, який поєднує планування

задач, календарні події, авторизацію, пошук, фільтрацію, статистику та персональні налаштування. Реалізований продукт відповідає визначеним вимогам і може бути використаний як основа для подальшого розвитку, наприклад розширення системи нагадувань або додавання нових способів аналізу навчального навантаження.

ПЕРЕЛІК ПОСИЛАНЬ

1. Google Calendar Help. *Google*. URL: https://support.google.com/calendar/ (дата звернення: 04.02.2026).

2. Todoist: A To-Do List to Organize Your Work & Life. *Todoist*. URL: https://www.todoist.com/ (дата звернення: 04.02.2026).

3. Notion: Your Connected Workspace. *Notion*. URL: https://www.notion.com/ (дата звернення: 05.02.2026).

4. Trello: Manage Your Team’s Projects From Anywhere. *Trello*. URL: https://trello.com/ (дата звернення: 05.02.2026).

5. Microsoft To Do: Lists, Tasks & Reminders. *Microsoft*. URL: https://www.microsoft.com/en-us/microsoft-365/microsoft-to-do-list-app (дата звернення: 06.02.2026).

6. HTML: HyperText Markup Language. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 12.02.2026).

7. CSS: Cascading Style Sheets. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 12.02.2026).

8. JavaScript. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 13.02.2026).

9. React Documentation. *React*. URL: https://react.dev/ (дата звернення: 18.02.2026).

10. Angular Documentation. *Angular*. URL: https://angular.dev/ (дата звернення: 19.02.2026).

11. Vue.js Documentation. *Vue.js*. URL: https://vuejs.org/ (дата звернення: 19.02.2026).

12. Node.js Documentation. *Node.js*. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 25.02.2026).

13. Express.js Documentation. *Express.js*. URL: https://expressjs.com/ (дата звернення: 25.02.2026).

14. PostgreSQL Documentation. *PostgreSQL*. URL: https://www.postgresql.org/docs/ (дата звернення: 02.03.2026).

15. Docker Compose Documentation. *Docker*. URL: https://docs.docker.com/compose/ (дата звернення: 07.03.2026).

16. NGINX Documentation. *NGINX*. URL: https://nginx.org/en/docs/ (дата звернення: 08.03.2026).

17. REST. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/REST (дата звернення: 14.03.2026).

18. Axios Documentation. *Axios*. URL: https://axios-http.com/docs/intro (дата звернення: 18.03.2026).

19. Bcryptjs. *npm*. URL: https://www.npmjs.com/package/bcryptjs (дата звернення: 21.03.2026).