НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: WEB-застосунок адміністрування роботи відділу аспірантури

Виконав студент IV курсу, групи ІП-13
(шифр групи)
Недельчев Євген Олександрович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Фіногенов О. Д. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ст. викл. Вітковська І. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент, к.т.н., доц., Мураховський С. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Недельчеву Євгену Олександровичу
(прізвище, ім'я, по батькові)

1. Тема проєкту WEB-застосунок адміністрування роботи відділу аспірантури

керівник проєкту Фіногенов Олексій Дмитрович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. №1705-с

2. Термін подання студентом проєкту «16» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області
2) Розроблення вимог до програмного забезпечення
3) Конструювання та розроблення програмного забезпечення
4) Аналіз якості та тестування програмного забезпечення.
5) Розгортання та супровід програмного забезпечення

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань _____

2) Схема бази даних _____

3) Креслення вигляду екранних форм _____

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «15» березня 2025 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	01.04.2025	
3	Постановка та формалізація задачі	12.04.2025	
4	Розробка інформаційного забезпечення	17.04.2025	
5	Алгоритмізація задачі	21.04.2025	
6	Обґрунтування вибору використаних технічних засобів	23.04.2025	
7	Розробка програмного забезпечення	11.05.2025	
8	Налагодження програми	20.05.2025	
9	Виконання графічних документів	22.05.2025	
10	Оформлення пояснювальної записки	23.05.2025	
11	Подання ДП на попередній захист	02.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	14.06.2025	

Студент _____ Євген НЕДЕЛЬЧЕВ
(підпис) (ініціали, прізвище)

Керівник _____ Олексій ФІНОГЕНОВ
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 39 таблиць, 25 рисунків та 14 джерел – загалом 71 сторінка.

Дипломний проєкт присвячений розробці вебдодатку PostgradHub, котрий використовується для автоматизації та адміністрування навчальних процесів ЗВО та призначений для підвищення ефективності внутрішніх процесів університету.

Метою проєкту є підвищення швидкості та якості роботи відділу аспірантури закладу вищої освіти завдяки автоматизації ключових процесів та візуалізації роботи.

У першому розділі було здійснено аналіз предметної області та існуючих рішень, описано бізнес процеси та сформульовано задачу.

Другий розділ присвячений аналізу вимог до програмного забезпечення, формулюванню функціональних та нефункціональних вимог до системи.

В третьому розділі розглянуто підходи до конструювання програмного забезпечення, проаналізовано різні підходи, та описано деталі вибору архітектури та реалізації системи.

Четвертий розділ присвячений аналізу якості розробленого програмного продукту та здійсненню його тестування.

В п’ятому розділі, здійснюється опис процесу розгортання та супроводу програмного забезпечення.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, ЗВО, АСПІРАНТУРА, REACTJS, NEXTJS, NESTJS, PRISMAORM, POSTGRESQL.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 39 tables, 25 figures and 14 sources – in total 71 pages.

The diploma project is dedicated to the development of the PostgradHub web application, which is used to automate and administer the educational processes of higher education institutions and is designed to improve the efficiency of internal university processes.

The goal of the development is to create a web application that will increase the speed and quality of work by automating key processes and visualizing work.

The first section analyzes the subject area and existing solutions, describes business processes, and formulates the task.

The second section is devoted to the analysis of software requirements, formulation of functional and non-functional requirements for the system.

The third section discusses approaches to software design, analyzes different approaches, and describes the details of choosing the architecture and implementing the system.

The fourth section is devoted to the analysis of the quality of the developed software product and its testing.

The fifth section describes the process of software deployment and maintenance.

KEYWORDS: WEBAPP, HEI, PHD, REACTJS, NEXTJS, NESTJS, PRISMAORM, POSTGRESQL.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1323.045440.01.91	Технічне завдання	16	
3	А4	КПІ.ІП-1323.045440.02.81	Пояснювальна записка	71	
4	А4	КПІ.ІП-1323.045440.03.12	Текст програми	25	
5	А4	КПІ.ІП-1323.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1323.045440.05.34	Керівництво користувача	13	
7	А3	КПІ.ІП-1323.045440.06.99	Графічний матеріал	3	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1323.045440.00.90		
Розроб.		Недельчев Є.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Фіногенов .					
Н.контр.		Вітковська І.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-13	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2025 р.

WEB-застосунок адміністрування роботи відділу аспірантури

Технічне завдання

КПІ.ІП-1323.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олексій ФІНОГЕНОВ

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Євген НЕДЕЛЬЧЕВ

Київ – 2025

Зміст

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: WEB-застосунок адміністрування роботи відділу аспірантури

Галузь застосування: освіта та наука.

Наведене технічне завдання поширюється на розробку вебдодатку PostgradHub, котрий використовується для автоматизації та адміністрування навчальних процесів ЗВО та призначений для підвищення ефективності внутрішніх процесів університету.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки PostgradHub є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для автоматизації та адміністрування роботи відділу аспірантури, що дозволить оптимізувати обмін інформацією та підвищити ефективність внутрішніх процесів університету.

Метою розробки є підвищення швидкості та якості роботи відділу аспірантури закладу вищої освіти завдяки автоматизації ключових процесів та візуалізації роботи.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу:

– сторінка реєстрації (рисунок 4.1);

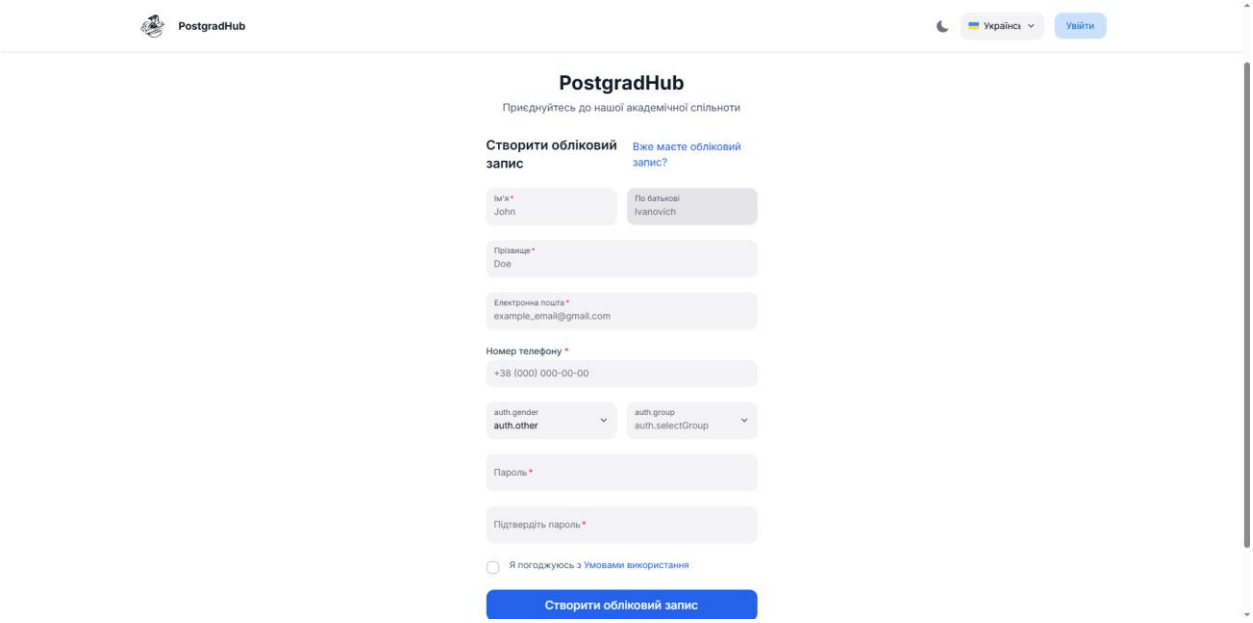


Рисунок 4.1 – Прототип сторінки реєстрації

– сторінка авторизації (рисунок 4.2);

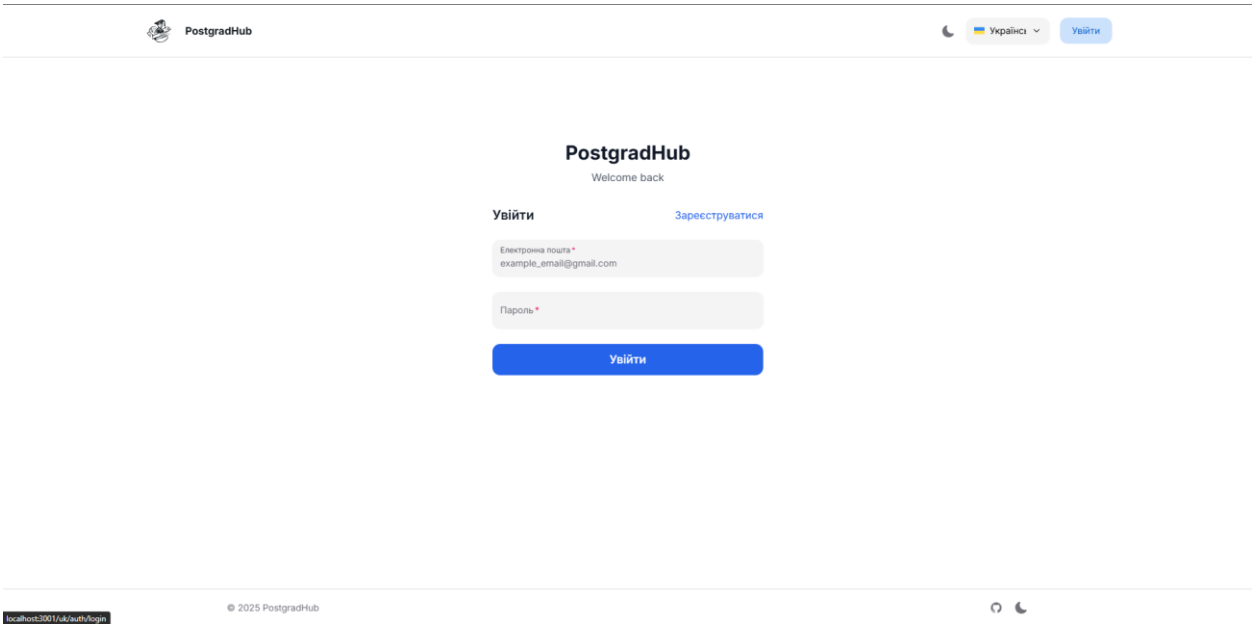


Рисунок 4.2 – Прототип сторінки авторизації

– головна сторінка (рисунок 4.3);

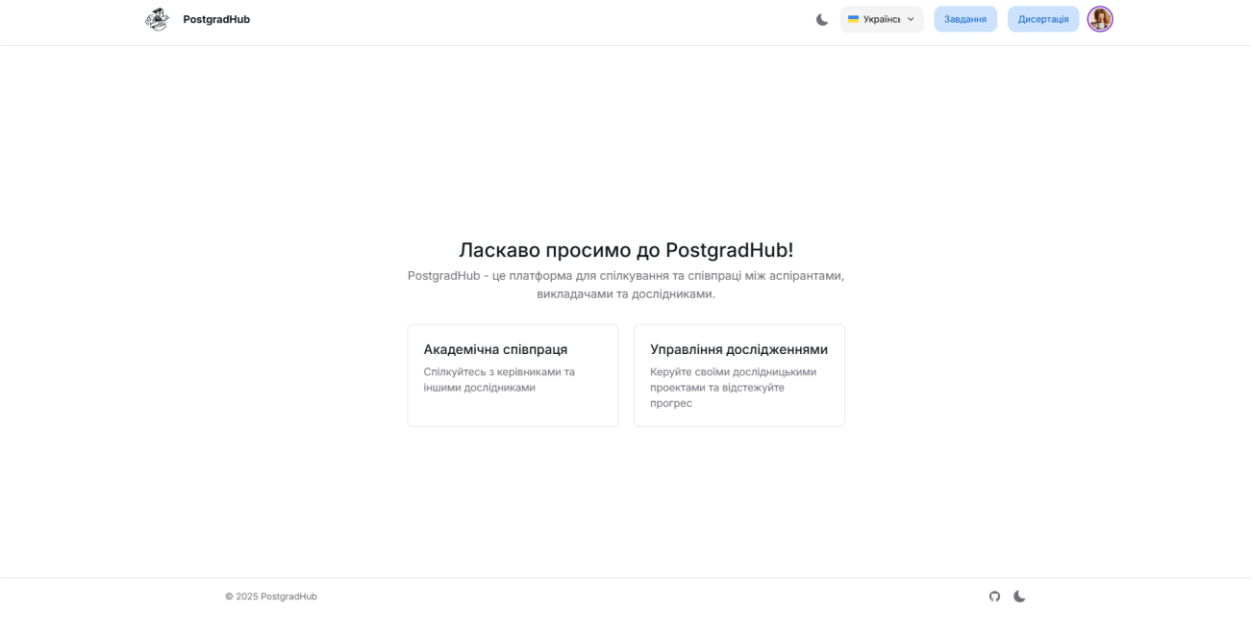


Рисунок 4.3 – Прототип головної сторінки

– сторінка профіля користувача (рисунок 4.4);

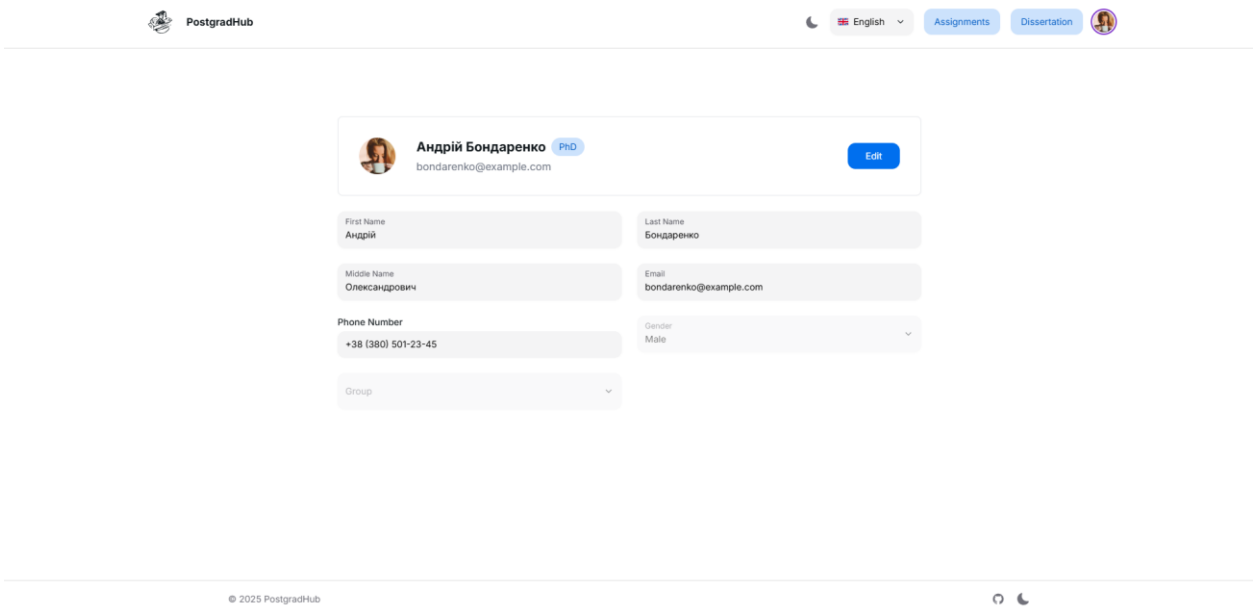


Рисунок 4.4 – Прототип профілю користувача

– сторінка управління планом дисертації (рисунок 4.5);

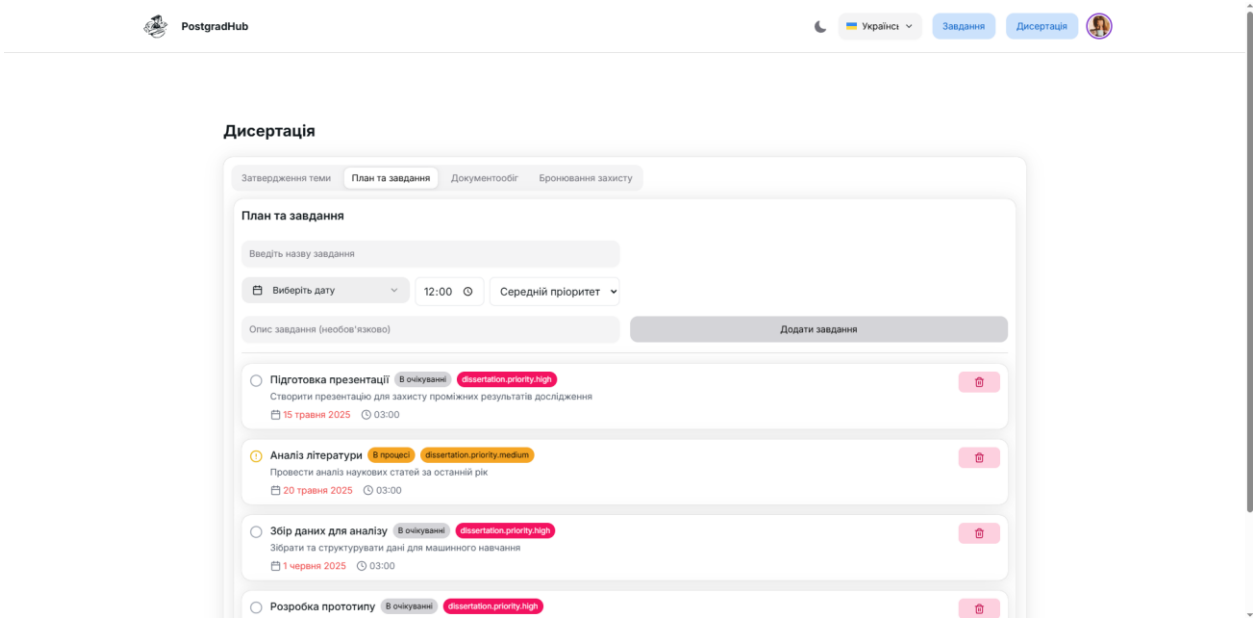


Рисунок 4.5 – Прототип сторінки управління планом дисертації

– сторінка перегляду списку завдань (рисунок 4.6);

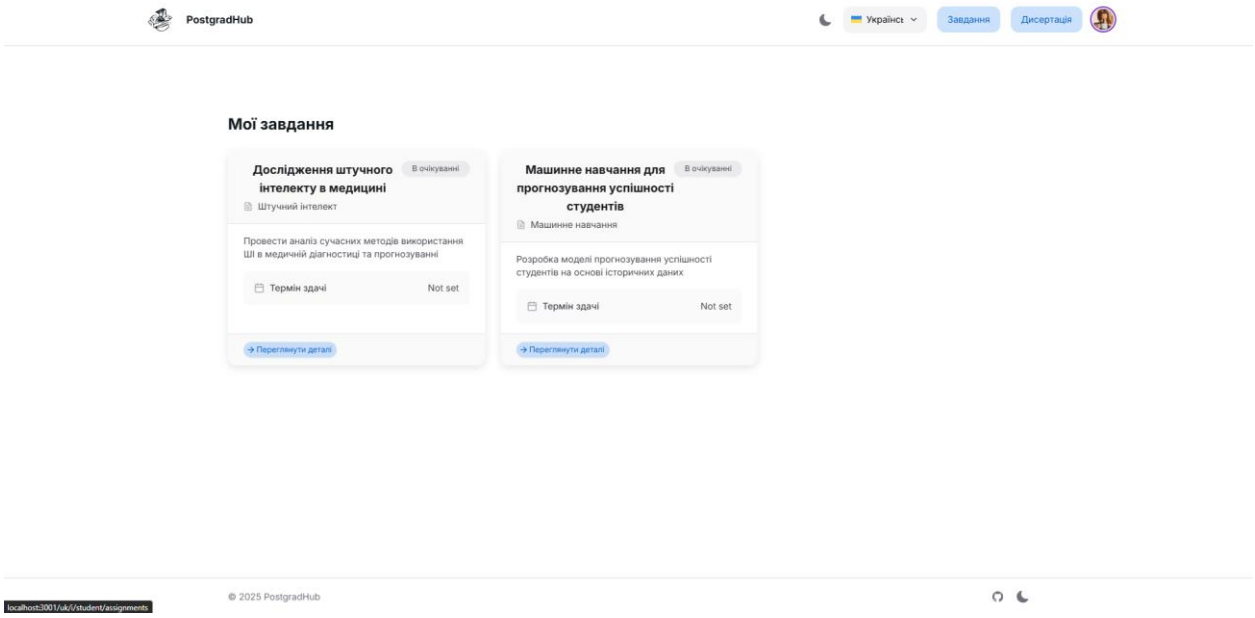


Рисунок 4.6 – Прототип сторінки зі списком завдань

– сторінка бронювання дати захисту (рисунок 4.7);

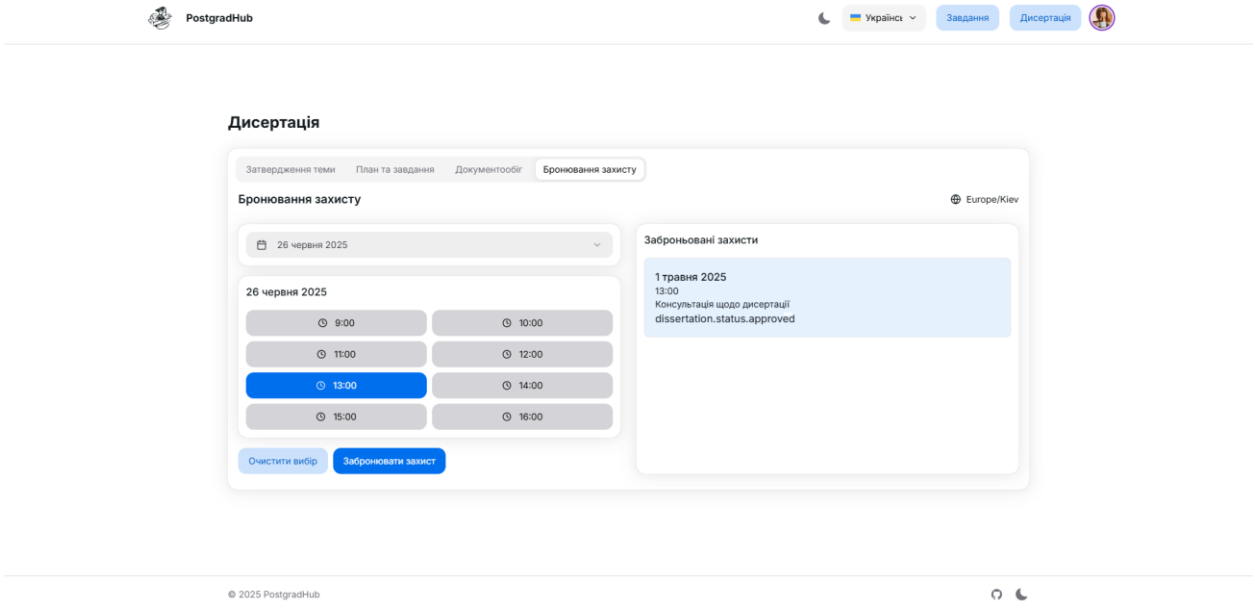


Рисунок 4.7 – Прототип сторінки бронювання дати захисту

– сторінка з аналітикою та звітами (рисунок 4.8);

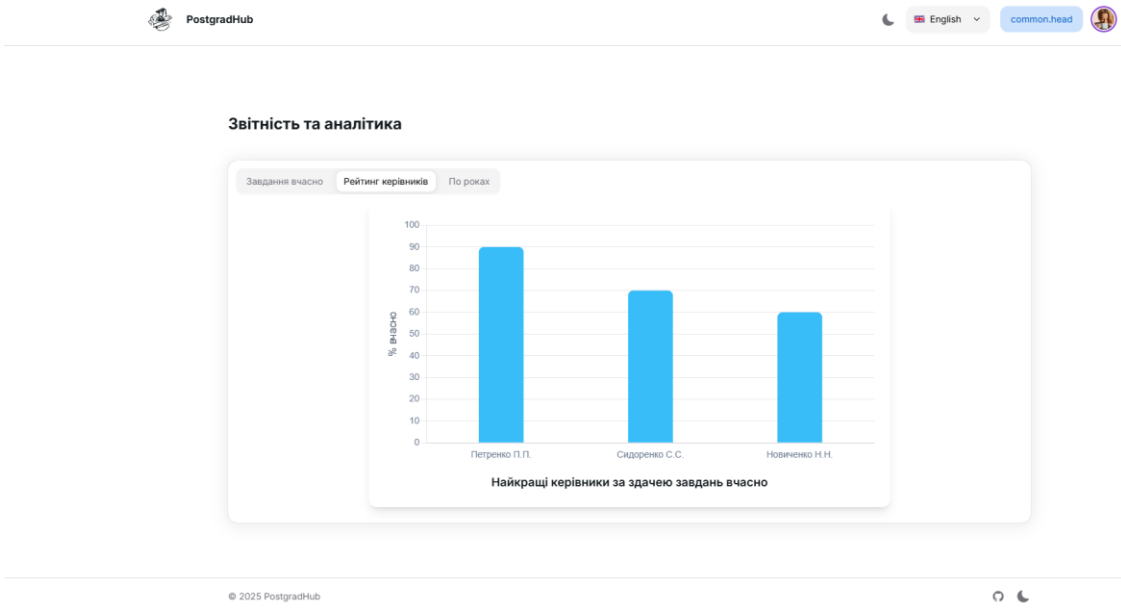


Рисунок 4.8 – Прототип сторінки звітності та аналітики

– сторінка перегляду списку студентів (рис. 4.9).

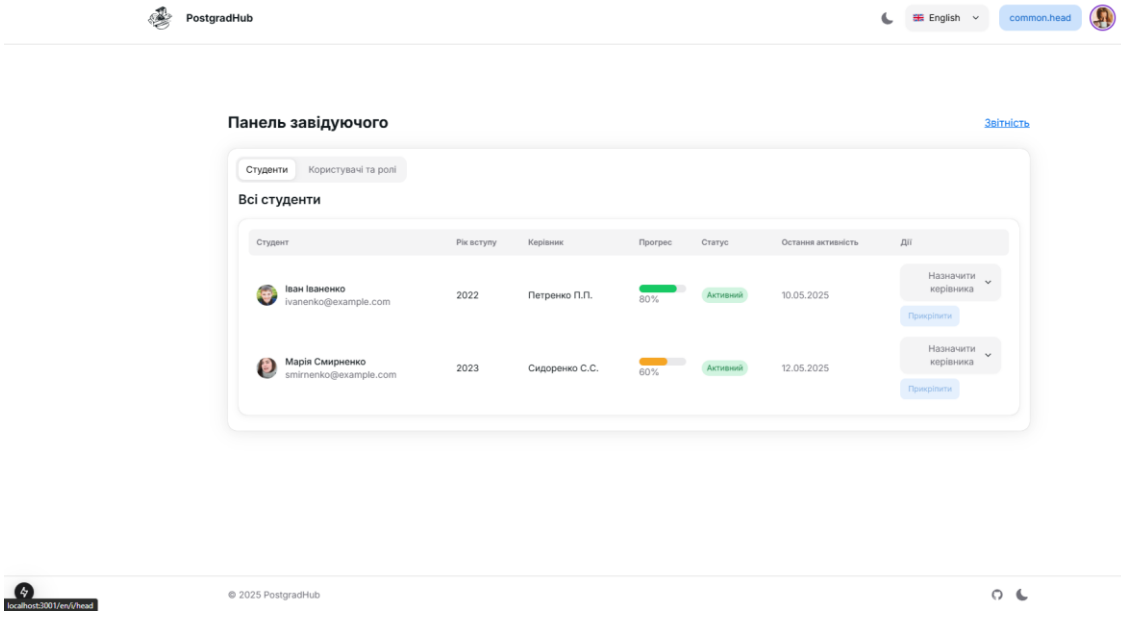


Рисунок 4.9 – Прототип сторінки зі списком студентів

4.1.2 Для неавторизованого користувача:

– стартова сторінка;
– реєстрація користувача;
– авторизація користувача.

4.1.3 Для авторизованого користувача (студента):

- перегляд головна сторінка;
- перегляд і редагування профілю;
- перегляд і редагування плану дисертації;
- перегляд та подання на перевірку завдань;
- подача теми дисертації на затвердження;
- бронювання дати захисту.

4.1.4 Для авторизованого користувача (наукового керівника):

- перегляд списку студентів, закріплених за науковим керівником;
- призначення та редагування задач, встановлення дедлайнів;
- перегляд поданих файлів, коментування, оцінювання;
- затвердження тем наукових дисертацій студентів.

4.1.5 Для авторизованого користувача (голови відділу аспірантури):

- перегляд та завантаження аналітики та звітності;
- адміністрування користувачів;
- затвердження графіків захистів.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

- тип процесору: Intel Core i3;
- об‘єм ОЗП: 4 Гб;
- підключення до мережі Інтернет зі швидкістю від 20 мегабіт;
- вільне місце на диску: 100 МБ.

Рекомендована конфігурація технічних засобів (та на якій виконувалась розробка):

- тип процесору: Intel Core i5;
- об‘єм ОЗП: 16 Гб;
- підключення до мережі Інтернет зі швидкістю від 100 мегабіт;
- вільне місце на диску: 5 Гб

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем:

- Windows 10 і новіше;
- MacOS 10.14 і новіше;
- дистрибутиви Linux Ubuntu та Mint.

4.5.1 Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються.

4.5.2 Вимоги до вихідних даних

Вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування TypeScript.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища Visual Studio Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторію на GitHub.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати готову до розгортання збірку додатку.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

- пояснювальна записка;
- технічне завдання;
- текст програми;
- програма та методика тестування;
- керівництво студента;
- керівництво наукового керівник;
- керівництво голови відділу освіти.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

- схема структурна варіантів використання;
- схема бази даних;
- креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення рекомендованої літератури	15.03.2025	
2.	Аналіз існуючих методів розв’язання задачі	01.04.2025	
3.	Постановка та формалізація задачі	12.04.2025	
4.	Розробка інформаційного забезпечення	17.04.2025	
5.	Алгоритмізація задачі	21.04.2025	
6.	Обґрунтування вибору використаних технічних засобів	23.04.2025	
7.	Розробка програмного забезпечення	11.05.2025	
8.	Налагодження програми	20.05.2025	
9.	Виконання графічних документів	22.05.2025	
10.	Оформлення пояснювальної записки	23.05.2025	
11.	Подання ДП на попередній захист	02.06.2025	
12.	Подання ДП рецензенту	10.06.2025	
13.	Подання ДП на основний захист	14.06.2025	

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **WEB-застосунок адміністрування роботи відділу аспірантури**

КПІ.ІП-1323.045440.02.81

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment, інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс.
CSV	–	Comma-Separated Values, текстовий формат табличних даних, розділених комами
ER	–	Entity-Relation diagram.
LMS	–	Learning Management Systems
JWT	–	JSON Web Token, вебтокен у форматі JSON для автентифікації
ER	–	Entity-Relation diagram.
SEO	–	Search Engine Optimization
XSS	–	Cross-Site Scripting
REST	–	Representational State Transfer, стиль архітектури веб-API
SQL	–	Structured Query Language
UI	–	User Interface
PhD	–	Доктор філософії
БД	–	База даних.
ЗВО	–	Заклад вищої освіти

ВСТУП

У сучасних умовах цифрової трансформації вищої освіти ефективне управління процесами аспірантури потребує застосування спеціалізованих інформаційних систем. Наявність різних аспектів PhD (терміни зарахування, порядок звітності, необхідність затвердження звітів науковим керівником) ускладнює ручне ведення обліку та обмін даними, що призводить до втрати часу та збільшеного ризику помилок. Розробка вебзастосунку для автоматизації та адміністрування роботи відділу аспірантури є вкрай актуальною з огляду на потребу збільшення швидкості обробки інформації та забезпечення прозорості процесів у закладах вищої освіти.

Провідні тенденції в галузі освітніх технологій спрямовані на інтеграцію Learning Management Systems (LMS) із модулями для управління дослідницькою діяльністю, впровадження єдиного інформаційного простору з аналітичними панелями (Dashboard) і можливістю обміну даними між різними підсистемами університету. Проте в українському середовищі відсутні універсальні інструменти, адаптовані під особливості нормативних вимог та внутрішніх процесів кожного ЗВО.

Стан сучасних розробок у цій тематиці характеризується поодинокими впровадженнями модулів у загальні інформаційні системи університетів, які найчастіше обмежуються базовим обліком студентів і не враховують специфіку аспірантської діяльності. Комплексного рішення, що поєднувало б аналіз робочих процесів, гнучку архітектуру бази даних та мультиролевий інтерфейс, наразі немає.

У рамках дипломного проекту планується:

– провести аналіз робочих процесів відділу аспірантури;

– розробити архітектуру системи та створити схему бази даних;

– реалізувати інтерфейси для різних ролей (аспірант, науковий керівник, керівник відділу);

– забезпечити можливість інтеграції з внутрішніми інформаційними системами ЗВО;

– сформувати модуль звітності та аналізу результатів.

Метою роботи є створення вебзастосунку, який збільшить швидкість та якість роботи відділу аспірантури шляхом автоматизації ключових процесів та візуалізації даних. Об’єктом дослідження є програмне забезпечення для адміністрування навчальних процесів ЗВО, предметом – процеси розробки, аналізу, впровадження та супроводу такого ПЗ.

Практичне значення проекту полягає в тому, що розроблене рішення може бути впроваджене в університетах та інститутах для:

– централізованого обліку даних аспірантів;

– автоматизації формування й погодження звітностей;

– підвищення прозорості та оперативності управлінських рішень у сфері підготовки PhD-спеціалістів.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

Автоматизація обліку та адміністрування аспірантських відділів є вкрай нагальною потребою, беручи до уваги багатогранність освітніх рівнів PhD, відмінність термінів зарахування та звітності, а також потребу в узгодженні звітів з науковими керівниками. Відсутність цілісного інструменту для введення й опрацювання інформації ускладнює оперативний зв'язок між аспірантами, керівниками та адміністрацією, що веде до втрати часу та вірогідних помилок в документах.

В рамках дипломного проєкту планується розробити вебдодаток, що комплексно оптимізує робочі процеси відділу аспірантури. З цією метою спершу необхідно провести глибокий аналіз предметної області: окреслити ключові бізнес-процеси, виявити слабкі місця поточних підходів та визначити спільні вимоги до системи. Наступним кроком буде вивчення існуючих рішень – як універсальних систем управління навчанням (LMS) чи систем планування ресурсів підприємства (ERP), так і спеціалізованих продуктів для обліку PhD, оцінити їх функціональність та можливість адаптації під нормативні вимоги українських вишів.

Таким чином, основними задачами даної роботи є: моделювання бізнес-процесів відділу аспірантури та формалізація функціональних і нефункціональних вимог, розробка загальної архітектури програмного забезпечення з обґрунтуванням вибору технологій, створення бази даних та інтерфейсів для різних користувачів (аспірант, науковий керівник, керівник відділу), забезпечення можливості інтеграції з внутрішніми інформаційними системами університету, а також проведення тестування, розгортання та підготовка повної супровідної документації. Кінцева мета – створити надійний та зручний інструмент для покращення швидкості та прозорості управлінських рішень у сфері підготовки PhD-спеціалістів.

1.2 Аналіз предметної області

Предмет дослідження стосується організаційних процесів та адміністрування підготовки претендентів на науковий ступінь доктора філософії (PhD) у вищих навчальних закладах України. Згідно із Законом України «Про вищу освіту» (№ 1556-VII від 01.07.2014), аспірантура – це форма організації підготовки науково-педагогічних кадрів, що реалізується шляхом систематичних наукових досліджень під керівництвом наукового керівника. Ключові бізнес-процеси в цій сфері охоплюють: реєстрацію здобувачів, формування індивідуальних планів наукової роботи, подання та погодження звітів, організацію захистів та облік результатів.

Сучасний розвиток інформаційних технологій у вищій освіті спрямований на створення єдиного інформаційного середовища закладу, що об'єднує студентські, викладацькі та адміністративні компоненти. Зокрема, моделі систематизованих інформаційних систем представлені як основа для покращення синергетичних взаємодій між внутрішніми комунікаціями та зовнішніми інтерфейсами університету. Разом з тим, для аспірантів розробляються відкриті цифрові платформи, що забезпечують доступ до наукових ресурсів, обробку результатів експериментів та створення звітів.

На даний момент у вітчизняних ЗВО впроваджуються окремі модулі в межах LMS/ERP-рішень (наприклад, базова база даних студентів у Moodle[5]), що дозволяють реєструвати здобувачів та вести основну звітність. Однак такий підхід має певні недоліки:

– розрізненість даних: інформація зберігається в різних системах без єдиної консолідації;

– обмеженість функціоналу: відсутні механізми автоматичного формування індивідуальних планів та погодження звітів із науковими керівниками;

– ручне управління процесами: ряд операцій (наприклад, призначення дат захистів) виконується вручну, що збільшує ризик помилок та затримок.

Для подолання цих проблем у сучасних дослідженнях пропонуються такі шляхи вдосконалення:

– створення єдиного сховища даних з чітко визначеною моделлю предметної області та розмежуванням прав доступу користувачів;

– автоматизація логіки бізнес-процесів через робочі процеси (workflow) з відстеженням статусів;

– розробка модульного вебінтерфейсу з адаптивним UI для різних ролей (аспірант, науковий керівник, відділ аспірантури);

– можливість інтеграції з внутрішніми системами університету та зовнішніми реєстрами.

В рамках цього дипломного проекту обрано підхід створення окремого вебзастосунку на основі сучасного технологічного стеку (Next.js[2] на фронтенді, Nest.js[3] з Prisma[4] на бекенді), що забезпечить централізоване зберігання даних та реалізує автоматизований workflow для всіх етапів підготовки аспірантів.

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації вебдодатку PostgradHub. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз відомих програмних продуктів

Для порівняння проєкту з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	PostgradHub	Moodle	ProQuest ETD Administrator	Пояснення
Мульти-ролевий доступ (аспірант, науковий керівник, адміністратор)	Так	Так (студент/ викладач/ адмін)	Так (автор/ адмін)	Moodle має базові ролі, заточені під навчальні курси, немає окремих ролей керівника відділу; ProQuest ETD підтримує тільки роль автора та адміністратора.
Узгодження звітів з науковими керівниками	Так	Частково (через форуми/ коментарі)	Ні	У PostgradHub – вбудований workflow для затвердження звітів; Moodle не має формального процесу узгодження документів, ProQuest ETD – лише подача без зворотного зв’язку.

Продовження таблиці 1.1

Функціонал	PostgradHub	Moodle	ProQuest ETD Administrator	Пояснення
Додавання опублікованих статей (назва та лінк)	Так	Ні	Ні	В PostgradHub можна зберігати публікації аспіранта; Moodle/ProQuest не мають окремого реєстру статей у контексті PhD.
Бронювання дати/часу захисту дисертації	Так	Ні	Ні	Жоден із аналогів не підтримує модуль онлайн-бронювання сесій захисту у календарі з автоматичними сповіщеннями.
План роботи на дисертацію (задачі та опис), які собі формує аспірант	Так	Ні	Ні	У Moodle немає модулю деталізації робочого плану дисертації; ProQuest фокусується на поданні фінального документа.

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

У сучасній розробці вебзастосунків для автоматизації адміністративних процесів розглядають кілька ключових фронтенд- та бекенд- рішень, а також різні підходи до роботи з базами даних.

По-перше, серед інструментів, що орієнтовані на React[1], є Create React App, Gatsby, Remix та Next.js. CRA підходить для простих SPA, але не має вбудованого серверного рендерингу, що негативно впливає на швидкість першого завантаження та SEO. Gatsby найкраще пасує для статичних сайтів із великою кількістю контенту, але складні інтерактивні адмін-панелі там будувати не дуже зручно. Remix пропонує гнучкий роутінг і високу продуктивність, проте менш розповсюджений і вимагає освоєння специфічних патернів. Next.js поєднує переваги SSR і SSG «з коробки», має файлову маршрутизацію, API-роути в тому ж репозиторії та автоматичну оптимізацію ресурсів, що робить його ідеальним вибором для університетської адмін-панелі: прискорює відображення сторінок, полегшує інтеграцію з бекендом та зменшує обсяг рутинного налаштування.

На серверній частині часто обирають Express або Koa – легкі HTTP-рушії, проте вони не визначають чіткої структури проєкту. У великих застосунках доводиться «вручну» будувати шари контролерів, сервісів і репозиторіїв, писати boilerplate для DI та валідації. NestJS натомість пропонує модульну архітектуру з вбудованим контейнером залежностей, декларативними декораторами для контролерів і провайдерів, підтримкою middleware, фільтрів винятків та готовою інтеграцією з OpenAPI (Swagger[14]). Це дозволяє одразу мати чітку структуру «controller-service-repository» і фокусуватися на бізнес-логіці, а не на організації фреймворку.

Що стосується бази даних, традиційно на Node.js використовують ORM-рішення типу TypeORM, Sequelize чи Objection.js. Проте останнім часом Prisma здобуває популярність завдяки автоматичній генерації типізованих клієнтів для запитів, прозорій системі міграцій та ясній моделі схеми. На відміну від TypeORM, який може вимагати додаткових налаштувань заради потужної типізації, Prisma дає змогу миттєво отримати автодоповнення в IDE та гарантує відповідність коду схемі БД.

У результаті поєднання Next.js на фронтенді, NestJS на бекенді та Prisma ORM для роботи з PostgreSQL[6] надає:

– швидкий старт без зайвого boilerplate-коду;

– оптимізоване рендерингом SEO-дружнє відображення інтерфейсу;

– чітку модульну архітектуру з готовими паттернами DI та обробки помилок;

– типізовану роботу з даними, що зменшує кількість помилок та прискорює розробку.

Такий стек дозволяє мінімізувати рутинні налаштування фреймворків та зосередитися на автоматизації саме PhD-процедур: реєстрація, узгодження планів, звітність та призначення захистів.

1.4 Аналіз та моделювання бізнес-процесів

Для опису бізнес процесу програмного забезпечення використовується BPMN модель (рисунки 1.1 – 1.3).

Опис послідовності реєстрації користувача (рис. 1.1):

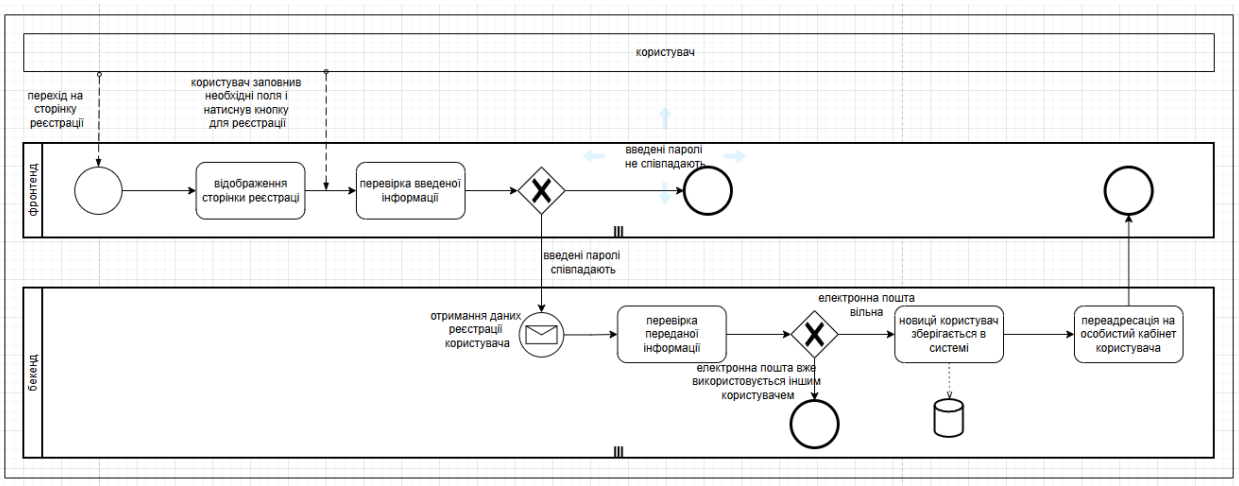


Рисунок 1.1 – BPMN модель реєстрації користувача

– користувач переходить на сторінку реєстрації користувача;

– користувач заповнює поля необхідні для реєстрації. Відбувається перевірка введеної інформації на стороні користувача. Якщо введена інформація некоректна то під некоректним полем виводиться повідомлення з

детальною інформацією про помилку. Якщо введена інформація коректна, то в такому випадку вся інформація відправляється на сервер на опрацювання;

– відбувається перевірка чи вільне ім’я користувача. Якщо електронна пошта зайнята, видається помилка про те, що пошта недоступна. Якщо введена інформація коректна, то вся інформація відправляється на сервер на опрацювання та збереження;

– користувач реєструється та зберігається в системі;

– користувач переадресовується в особистий кабінет

Опис моделі бізнес-процесу завантаження аспірантом дисертації (рис. 1.2):

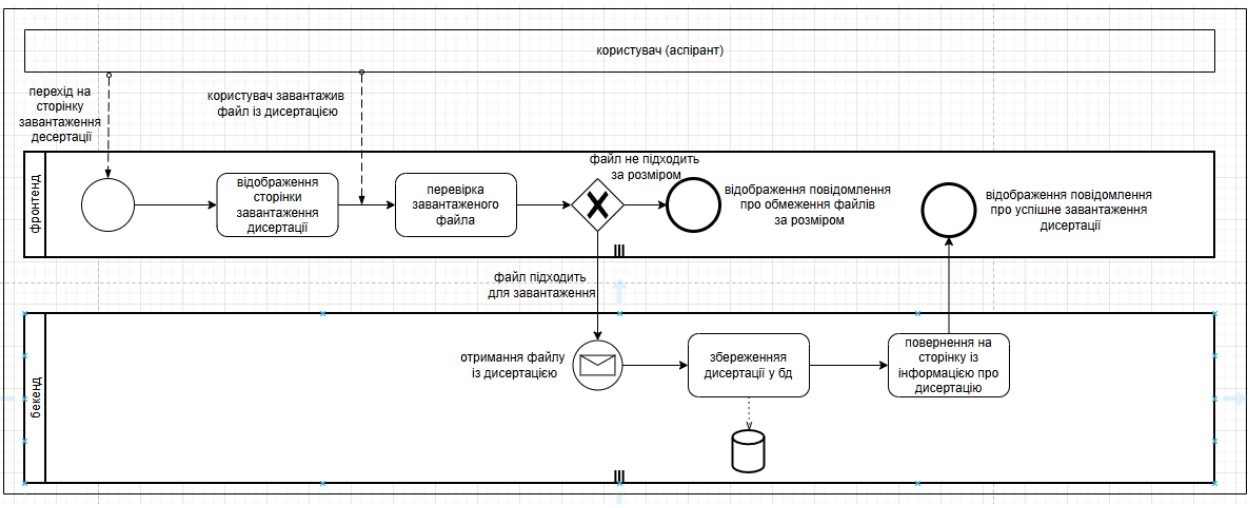


Рисунок 1.2 – BPMN модель завантаження дисертації

– користувач переходить на сторінку реєстрації завантаження дисертації;

– користувач завантажує файл із дисертацією. Відбувається перевірка завантаженого файлу на стороні користувача. Якщо файл не підходить за розміром, тоз’являється повідомлення про необхідність корегування. Якщо файл відповідає критеріям завантаження. то вся інформація відправляється на сервер на опрацювання;

– дисертація зберігається в системі;

– відображається повідомлення про успішне завантаження дисертації.

Опис моделі бізнес-процесу створення завдання для аспірантів науковим керівником (рис. 1.3):

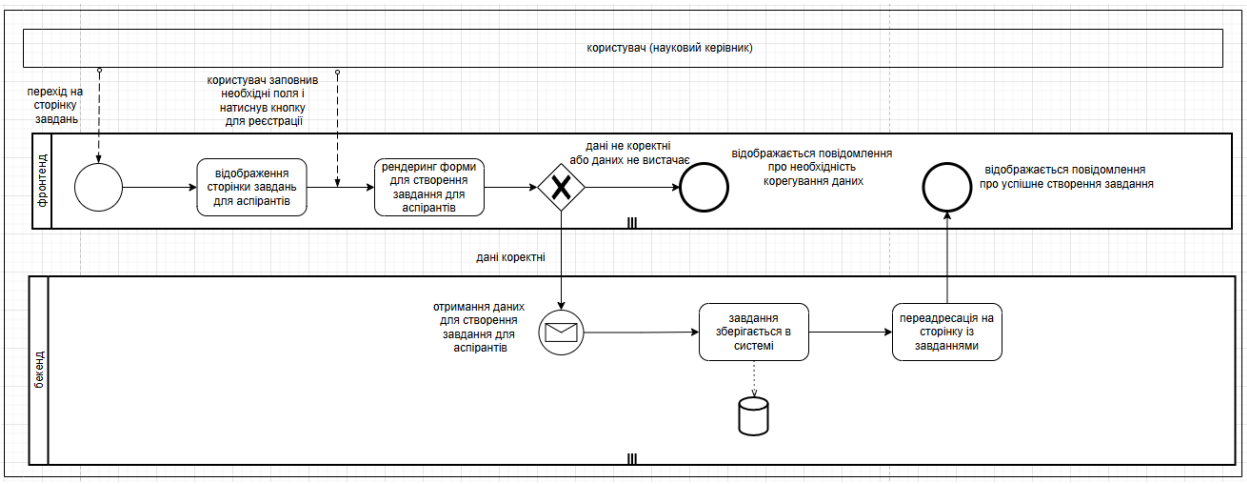


Рисунок 1.3 – BPMN модель створення завдання для аспірантів

– користувач переходить на сторінку завдань;

– користувач заповнює необхідні поля для створення завдання для аспірантів. Відбувається перевірка введеної інформації на стороні користувача. Якщо введена інформація некоректна, то під некоректним полем виводиться повідомлення з детальною інформацією про помилку. Якщо введена інформація коректна, то вся інформація відправляється на сервер на опрацювання;

– дані завдання для аспірантів зберігаються в системі;

– відображається повідомлення про успішне створення завдання і переадресація на сторінку із завданнями;

Висновки до розділу

Після здійснення передпроєктного дослідження предметної області було визначено, що автоматизація обліку та адміністрування аспірантури є

критичною складовою підвищення ефективності роботи відділів PhD у вищих навчальних закладах. Основна ідея проєкту полягає в об'єднанні всіх ключових процесів – від реєстрації здобувачів і формування індивідуальних планів до подання звітів, ведення публікацій і бронювання захистів –в одному єдиному вебдодатку з чітко розмежованими ролями та інтуїтивним інтерфейсом.

Аналіз наявних рішень засвідчив, що більшість LMS/ERP-систем (наприклад, Moodle, SAP SLcM, Oracle PeopleSoft Campus Solutions) та спеціалізованих продуктів для дисертацій (ProQuest ETD, Symplectic Elements) покривають лише окремі аспекти: вони можуть реєструвати здобувачів або зберігати фінальні документи, але не забезпечують комплексного workflow із автоматичним узгодженням звітів, деталізацією планів роботи, мультимедійним обліком публікацій і онлайн-бронюванням захистів. Головні недоліки таких систем – розрізненість даних, відсутність єдиного сховища, ручне призначення дат і обмежена інтеграція з локальними інформаційними системами університету.

У результаті було встановлено, що розробка спеціалізованого вебзастосунку PostgradHub є оптимальним рішенням для повноцінної автоматизації PhD-процесів. Цей сервіс забезпечить здобувачам можливість формувати й редагувати індивідуальний план роботи (завдання та їх опис), додавати інформацію про опубліковані статті (назва та лінк) і бронювати дату та час захисту через інтегрований календар з автоматичними сповіщеннями. Наукові керівники отримають інструменти для погодження звітів, моніторингу прогресу та взаємодії з аспірантами в рамках єдиного інтерфейсу. Адміністрація відділу зможе керувати заявками на створення чи редагування організацій, налаштовувати динамічні й статичні черги на послуги та інтегрувати PostgradHub із внутрішніми SIS/ERP-системами університету.

Таким чином, створення PostgradHub сприятиме підвищенню швидкості та прозорості управлінських рішень у сфері підготовки PhD-спеціалістів, зменшить кількість помилок, забезпечить єдину консолідацію даних і зміцнить конкурентні позиції вищих навчальних закладів.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Діаграму варіантів використання наведено в графічному матеріалі, креслення 1.

В таблицях 2.1-2.15 наведено опис варіантів використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться відповідні дані: пошта користувача, пароль в системі, та його повтор для підтвердження, ім’я та прізвище, а також чек-бокс для підтвердження умов сервісу. Після заповнення даних користувача натискає кнопку реєстрації. Після цього з’являється повідомлення про успішну реєстрацію, і користувач перенаправляється на головну сторінку додатку.
Extension	Якщо дані некоректні – кнопка неактивна, поля підсвічуються.
Post-Condition	Користувач створений, перехід на головну сторінку додатку.

Таблиця 2.2 – Варіант використання UC-02

Use case name	Вхід користувача
Use case ID	UC-02

Продовження таблиці 2.2

Goals	Аутентифікація користувача
Actors	Користувач
Trigger	Користувач бажає увійти в систему
Pre-conditions	Користувач зареєстрований
Flow of Events	Користувач переходить на сторінку входу, вводить e-mail і пароль, натискає «Увійти». У разі успіху – перехід на головну сторінку.
Extension	Якщо дані невірні – повідомлення про помилку.
Post-Condition	Користувач аутентифікований.

Таблиця 2.3 – Варіант використання UC-03

Use case name	Перегляд та редагування профілю
Use case ID	UC-03
Goals	Перегляд та зміна особистих даних
Actors	Користувач
Trigger	Користувач обирає «Профіль»
Pre-conditions	Користувач увійшов у систему
Flow of Events	Перехід на сторінку профілю, перегляд інформації, редагування та збереження змін.
Extension	Некоректні дані – підсвічування полів.
Post-Condition	Дані профілю оновлені.

Таблиця 2.4 – Варіант використання UC-04

Use case name	Вибір теми дисертації (для студента)
Use case ID	UC-04
Goals	Вибір або зміна теми дисертації
Actors	Студент
Trigger	Студент переходить у розділ «Дисертація»

Продовження таблиці 2.4

Pre-conditions	Студент авторизований
Flow of Events	Перегляд доступних тем, вибір теми, підтвердження вибору.
Extension	-
Post-Condition	Тема закріплена за студентом.

Таблиця 2.5 – Варіант використання UC-05

Use case name	Додавання публікацій до дисертації
Use case ID	UC-05
Goals	Додавання наукових публікацій
Actors	Студент
Trigger	Студент бажає додати публікацію
Pre-conditions	Студент авторизований
Flow of Events	Перехід у розділ «Публікації», введення даних, завантаження файлу, збереження.
Extension	Некоректний файл – повідомлення про помилку.
Post-Condition	Публікація додана.

Таблиця 2.6 – Варіант використання UC-06

Use case name	Перегляд та виконання завдань
Use case ID	UC-06
Goals	Виконання навчальних завдань
Actors	Студент
Trigger	Студент переходить у розділ «Завдання»
Pre-conditions	Студент авторизований
Flow of Events	Перегляд списку завдань, виконання, завантаження файлів, відмітка про виконання.

Продовження таблиці 2.6

Extension	Некоректний файл – повідомлення про помилку.
Post-Condition	Завдання надіслано на перевірку.

Таблиця 2.7 – Варіант використання UC-07

Use case name	Перевірка завдань студентів (для керівника)
Use case ID	UC-07
Goals	Перевірка та оцінювання завдань
Actors	Керівник
Trigger	Керівник переходить у розділ «Завдання студентів»
Pre-conditions	Керівник авторизований
Flow of Events	Перегляд списку студентів, вибір завдання, перевірка, виставлення оцінки.
Extension	-
Post-Condition	Завдання перевірено, оцінка виставлена.

Таблиця 2.8 – Варіант використання UC-08

Use case name	Перегляд звітів (для адміністратора/керівника)
Use case ID	UC-08
Goals	Аналіз звітності студентів
Actors	Керівник, Голова відділу аспірантури
Trigger	Перехід у розділ «Звіти»
Pre-conditions	Користувач має відповідні права
Flow of Events	Перегляд списку звітів, фільтрація, експорт даних.
Extension	-
Post-Condition	Звіт переглянуто/експортовано.

Таблиця 2.9 – Варіант використання UC-09

Use case name	Зміна мови інтерфейсу
Use case ID	UC-09
Goals	Перемикання мови додатку
Actors	Користувач
Trigger	Користувач бажає змінити мову
Pre-conditions	-
Flow of Events	Користувач обирає мову у меню, інтерфейс оновлюється відповідно до вибраної мови.
Extension	Якщо мова недоступна – повідомлення про помилку.
Post-Condition	Інтерфейс відображається обраною мовою.

Таблиця 2.10 – Варіант використання UC-10

Use case name	Керування темою інтерфейсу
Use case ID	UC-10
Goals	Зміна теми інтерфейсу між темною і світлою
Actors	Користувач
Trigger	Користувач бажає змінити тему інтерфейсу
Pre-conditions	-
Flow of Events	Користувач натискає на іконку перемикання теми в навігаційній панелі. Система змінює тему інтерфейсу на протилежну (світла/темна).
Extension	-
Post-Condition	Інтерфейс відображається в обраній темі.

Таблиця 2.11 – Варіант використання UC-11

Use case name	Керування темою інтерфейсу
Use case ID	UC-11

Продовження таблиці 2.11

Goals	Створення та управління планом роботи над дисертацією
Actors	Студент
Trigger	Студент переходить у розділ «План дисертації»
Pre-conditions	Студент авторизований
Flow of Events	Студент створює нові завдання, встановлює дедлайни, пріоритети та статуси. Система відображає прогрес виконання плану.
Extension	Некоректні дати – повідомлення про помилку
Post-Condition	План дисертації оновлено

Таблиця 2.12 – Варіант використання UC-12

Use case name	Керування студентами (для керівника)
Use case ID	UC-12
Goals	Перегляд та управління списком студентів
Actors	Керівник
Trigger	Керівник переходить у розділ «Студенти»
Pre-conditions	Керівник авторизований
Flow of Events	Перегляд списку студентів, їх прогресу, статусу та останньої активності. Можливість фільтрації та пошуку.
Extension	-
Post-Condition	-

Таблиця 2.13 – Варіант використання UC-13

Use case name	Аналітика та звітність (для голови відділу)
Use case ID	UC-13
Goals	Перегляд аналітичних даних та генерація звітів
Actors	Голова відділу аспірантури
Trigger	Перехід у розділ «Звіти»

Продовження таблиці 2.13

Pre-conditions	Користувач має права голови відділу
Flow of Events	Перегляд графіків та діаграм, що відображають статистику по студентах, керівниках та прогресу робіт. Можливість експорту даних.
Extension	-
Post-Condition	-

Таблиця 2.14 – Варіант використання UC-14

Use case name	Керування завданнями (для керівника)
Use case ID	UC-14
Goals	Створення та управління завданнями для студентів
Actors	Керівник
Trigger	Керівник переходить у розділ «Завдання»
Pre-conditions	Керівник авторизований
Flow of Events	Створення нових завдання, встановлення термінів виконання, призначення студентів, перегляд статусу виконання.
Extension	Некоректні дані – підсвічування полів
Post-Condition	Завдання створено/оновлено

Таблиця 2.15 – Варіант використання UC-15

Use case name	Керування користувачами (для адміністратора)
Use case ID	UC-15
Goals	Управління користувачами системи
Actors	Адміністратор
Trigger	Перехід у розділ «Користувачі»
Pre-conditions	Користувач має права адміністратора

Продовження таблиці 2.15

Flow of Events	Перегляд списку користувачів, зміна їх ролей, статусів та інших параметрів. Можливість блокування/розблокування облікових записів.
Extension	Некоректні дані – повідомлення про помилку
Post-Condition	Дані користувачів оновлено

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.16 наведено загальну модель вимог, а в таблиці 2.17 наведений опис функціональних вимог до програмного забезпечення.

Таблиця 2.16 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Перегляд сторінки привітання.	FR-1	Низький	Низький
2	Система авторизації (реєстрація / вхід / вихід)	FR-2	Високий	Високий
2.1	Реєстрація користувача	FR-3	Високий	Високий
2.2	Авторизація користувача.	FR-4	Високий	Високий
2.3	Вихід із акаунту.	FR-5	Середній	Низький
3	Перегляд та редагування профілю	FR-6	Середній	Низький
4	Керування інтерфейсом (мова / тема)	FR-7	Середній	Низький
5	План дисертації (створення / моніторинг)	FR-8	Високий	Середній
6	Управління науковими публікаціями	FR-9	Середній	Середній

Продовження таблиці 2.16

7	Завдання дисертації (виконання / перевірка)	FR-10	Високий	Середній
8	Бронювання дати захисту	FR-11	Високий	Середній
9	Завантаження/перевірка дисертаційних файлів	FR-12	Високий	Середній
10	Аналітика та генерація звітів	FR-13	Середній	Середній
11	Керування студентами (для керівника)	FR-14	Середній	Середній
12	Керування завданнями (для керівника)	FR-15	Високий	Середній
13	Адміністрування користувачів (для адміна)	FR-16	Високий	Високий
14	Пошук та фільтрація даних	FR-17	Середній	Низький
15	Експорт даних (CSV)	FR-18	Середній	Низький

Таблиця 2.17 – Перелік функціональних вимог

Назва	Опис
FR-1	Перегляд сторінки привітання. Неавторизований користувач бачить вітальну сторінку з інформацією про PostgradHub та кнопками «Увійти» / «Зареєструватися».
FR-2	Система авторизації. Платформа забезпечує повний цикл аутентифікації: реєстрація, вхід, керування токенами, вихід.
FR-3	Реєстрація користувача Гість заповнює форму, та реєструється, після чого отримує роль «Студент» за замовчуванням.

Продовження таблиці 2.17

FR-4	Авторизація користувача Перевірка облікових даних і надання JWT-токена з відповідними правами доступу.
FR-5	Вихід із акаунту Вебклієнт видаляє токен, сервер позначає сесію як неактивну.
FR-6	Редагування профілю Користувач може змінювати ПІБ, фото, контакти, пароль; система перевіряє валідність даних.
FR-7	Керування мовою/темою Підтримка перемикання між українською та англійською, світлою та темною темою, з миттєвим застосуванням.
FR-8	План дисертації Студент створює задачі з дедлайнами; система відстежує прогрес, відображає діаграми.
FR-9	Публікації Завантаження та перевірка публікацій
FR-10	Завдання дисертації Студент завантажує завдання; керівник перевіряє, залишає коментар, виставляє статус.
FR-11	Бронювання захисту Інтегрований календар із вільними слотами; студент пропонує дату, керівник підтверджує.
FR-12	Завантаження дисертації Завантаження файлів дисертації
FR-13	Аналітика та звіти Адміністрація формує дашборди, експортує звіти у CSV.

Продовження таблиці 2.17

FR-14	Керування студентами Керівник переглядає список аспірантів, їхній прогрес, статистику публікацій.
FR-15	Керування завданнями Керівник створює/редагує задачі, призначає студентів, встановлює дедлайни, пріоритети.
FR-16	Адміністрування користувачів Адміністратор змінює ролі, блокує/розблоковує облікові записи.
FR-17	Пошук та фільтрація Повнотекстовий пошук по дисертаціях, публікаціях; фільтри за датою, статусом, керівником.
FR-18	Експорт даних Можливість експортувати таблиці та звіти в CSV

На основі вищенаведених таблиць було побудовано матрицю трасування вимог, яку можна побачити на рисунку 2.1.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18
UC-1	X	X	X															
UC-2		X		X	X													
UC-3						X												
UC-4								X			X	X						
UC-5									X									
UC-6								X		X								
UC-7										X					X			
UC-8													X				X	X
UC-9							X											
UC-10							X											
UC-11								X		X								
UC-12														X	X		X	
UC-13													X					X
UC-14										X					X			
UC-15																X		

Рисунок 2.1 – Матриця трасування вимог

Матриця трасування вимог ілюструє зв’язок між функціональними вимогами (FR) та відповідними випадками використання (UC). Вона дозволяє простежити, які вимоги покриваються якими сценаріями системи. Це дає змогу переконатися, що кожна вимога реалізується хоча б одним випадком використання, і навпаки – що кожен сценарій має конкретне функціональне обґрунтування. Завдяки цій матриці видно, що ключові модулі, такі як авторизація, керування профілем, завданням та дисертацією, мають покриття у випадках використання, а також стає зрозуміло, які елементи системи мають найвищу інтеграцію й важливість, оскільки зустрічаються в кількох UC.

2.3 Розроблення нефункціональних вимог

У розроблюваному вебзастосунку мають бути реалізовані такі нефункціональні вимоги:

– безпека – система повинна бути надійно захищена від несанкціонованого доступу та звичайних вразливостей, таких як SQL-ін’єкції[11] та XSS-атаки[10];

– доступність – система має гарантувати стабільну роботу та бути доступною для користувачів більшість часу;

– сумісність – система повинна правильно функціонувати у різних браузерах без порушення основної функціональності;

– тестованість – реалізація функцій має дозволяти їх перевірку на відповідність визначеним функціональним вимогам;

– зручність використання – інтерфейс має бути простим, логічним та зрозумілим, щоб користувач чітко розумів результат кожної своєї дії.

2.4 Аналіз економічних показників програмного забезпечення

Для обчислення економічних показників застосуємо методику Use Case Points (UCP), враховуючи Technical Complexity Factor (TCF).

Розпочнемо з опису акторів. У нашому сценарії є чотири актори: Неавторизований користувач, Студент, Науковий керівник, Голова Відділу Аспірантури. Оцінимо їх як Complex за складністю, адже вони взаємодіють з інтерфейсом. Отже, кожен користувач отримає по 3 бали складності, сумарно – 12 балів.

Здійснимо класифікацію складності варіантів використання, згідно з таблицею 2.18, що представлена вище, у цьому розділі.

Таблиця 2.18 – Оцінка складності варіантів використання

Варіант використання	Складність	Бал
UC-1	Average	10
UC-2	Simple	5
UC-3	Average	10
UC-4	Average	10
UC-5	Average	10

Продовження таблиці 2.18

Варіант використання	Складність	Бал
UC-6	Complex	15
UC-7	Simple	5
UC-8	Average	10
UC-9	Simple	5
UC-10	Simple	5
UC-11	Average	10
UC-12	Simple	5
UC-13	Complex	15
UC-14	Average	10
UC-15	Simple	10

Маємо 135 балів.

$$UUCP = UUCW + UAW = 135 + 12 = 147$$

В таблиці 2.19 наведено технічні коефіцієнти факторів.

Таблиця 2.19 – Технічний коефіцієнт TCF

Фактор	Вага	Оцінка	Σ
T1: Розподіленість системи	2	3	6
T2: Продуктивність	1	3	3
T3: Складність UI	1	4	4
T4: Піддатність змінам	1	3	3
T5: Безпека	1	3	3
T6: Навчанная користувача	1	2	2
T7: Обслуговуваність	1	3	3

В сумі маємо 24.

$$TCF = 0.6 + 0.01 \times 40.5 = 1.005$$

$$UCP = UUCP \times TCF = 147 \times 1.005 \approx 147.735$$

За продуктивність одного розробника було взято 24 год / UCP. Розраховано очікуваний час розробки:

UCP / productivity = 147.735 / 24 ≈ 6.15 (людино/місяців)

Було припущено, що середня заробітня плата розробника скаладає 10$ за годину (1680$ на місяць), тоді:

1680$ * 6.15 = 10300$

Отже, запропонований проєкт оцінюється у 8280$, а за умови що він розроблявся однією людиною, то час на розробку дорівнює 5.52 місяці, що дуже близько до правди.

2.5 Постановка завдання на розробку програмного забезпечення

Розробка програмного забезпечення передбачає створення спеціалізованого вебзастосунку, призначеного для автоматизації та ефективного адміністрування процесів, пов’язаних з підготовкою аспірантів у закладах вищої освіти. Основна мета полягає у впровадженні зручного інструменту, який забезпечуватиме взаємодію між основними учасниками процесу – аспірантами, науковими керівниками та працівниками відділу аспірантури. Система має забезпечити централізований облік, моніторинг та аналіз даних, що стосуються навчальної та наукової діяльності здобувачів ступеня PhD.

Розроблюване програмне забезпечення має розв'язати ряд важливих задач, серед яких: автоматизація рутинних адміністративних дій, таких як реєстрація користувачів, створення індивідуальних планів, подання та узгодження звітів, бронювання дати захисту дисертації; підвищення швидкості, достовірності та прозорості обробки освітньої інформації; впровадження гнучкої системи ролей з відповідним розмежуванням прав доступу; а також забезпечення адаптивного інтерфейсу з підтримкою декількох мов.

Для реалізації зазначених цілей буде застосовуватись сучасний технологічний стек, що включає Next.js для фронтенду, NestJS для серверної логіки та PostgreSQL як систему управління базами даних. Передбачається створення архітектури на основі клієнт-серверної моделі з чітким поділом обов’язків між

складовими частинами. Система повинна підтримувати масштабовану модульну структуру, що забезпечить можливість її подальшого розширення та адаптації до потреб конкретного ЗВО.

У підсумку, результатом роботи має стати впровадження вебзастосунку PostgradHub – мінімально життєздатного продукту (MVP), який дасть змогу значно покращити адміністративне управління підготовкою аспірантів, скоротити часові та трудові витрати на обробку інформації, а також забезпечити єдине інформаційне середовище для всіх учасників процесу аспірантури.

Висновки до розділу

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення. Було здійснено аналіз варіантів застосування системи, в котрому визначено основні сценарії взаємодії користувачів з вебзастосунком: реєстрація, аутентифікація, редагування профілю, вибір теми дисертації, додавання публікацій, виконання завдань, подання звітів, бронювання захистів та інше. Для кожної функції було визначено відповідного актора, умови виконання, логіку поведінки та кінцеві наслідки.

На основі аналізу варіантів застосування сформульовано функціональні потреби до програмного забезпечення. Визначено основні модулі та їх функціонал, враховуючи систему авторизації, керування профілем, роботу з індивідуальними планами, публікаціями, звітами, аналітикою, адміністративною панеллю тощо. Усі потреби були структуровані, оцінені за пріоритетами та ризиками, що дозволяє планувати подальший процес розробки більш обґрунтовано.

Також визначено основні нефункціональні вимоги до системи, зокрема щодо безпеки, доступності, сумісності, тестованості та зручності інтерфейсу. Ці аспекти є надзвичайно важливими для надійної та безперебійної роботи вебзастосунку в умовах реального використання.

На завершення розділу було узагальнено постановку завдання на розробку програмного забезпечення, де окреслено основну мету, цілі та задачі

проєкту. Всі зібрані та систематизовані потреби лягли в основу технічного завдання, що стане відправною точкою для наступного етапу – безпосередньої розробки програмного продукту.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Для розробки вебзастосунку адміністрування роботи відділу аспірантури було обрано клієнт-серверну архітектуру з окремими фронтенд та бекенд складниками. Такий підхід забезпечує розділення обов'язків між компонентами, дозволяючи досягти високої гнучкості та масштабованості системи.

Інтерфейсна частина (фронтенд) відповідає за взаємодію з користувачем та показ даних, тоді як серверна логіка (бекенд) реалізує бізнес-правила, обробляє запити та працює з базою даних. Клієнт та сервер функціонують як автономні модулі, сполучаючись між собою через API за допомогою стандартних протоколів.

Клієнт-серверна архітектура дає змогу незалежно масштабувати фронтенд та бекенд згідно з поточними навантаженнями. Таке розділення обов’язків також спрощує підтримку та розвиток системи: зміни в одній частині не впливають безпосередньо на іншу.

Завдяки поділу на два незалежні частини, система легко інтегрується з новими службами та може бути пристосована під різні сценарії застосування. Це робить клієнт-серверну архітектуру оптимальним вибором для вебзастосунку адміністрування роботи відділу аспірантури.

Нижче представлено докладний опис архітектури вебдодатку за допомогою діаграм C4 (рис. 3.1 – 3.4).

На рисунку 3.1 зображено загальний огляд системи у вигляді контекстної діаграми, де користувачі взаємодіють з додатком.

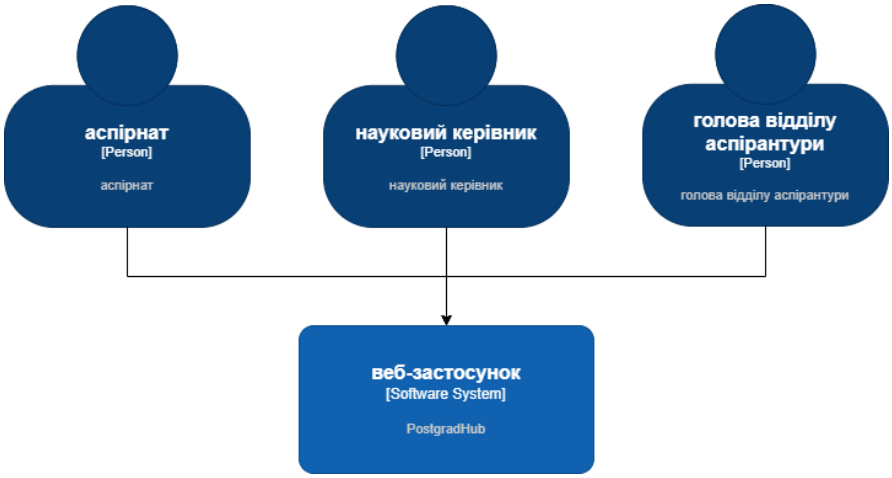


Рисунок 3.1 – Перший рівень діаграми C4

На рисунку 3.2 представлено діаграму контейнерів системи.

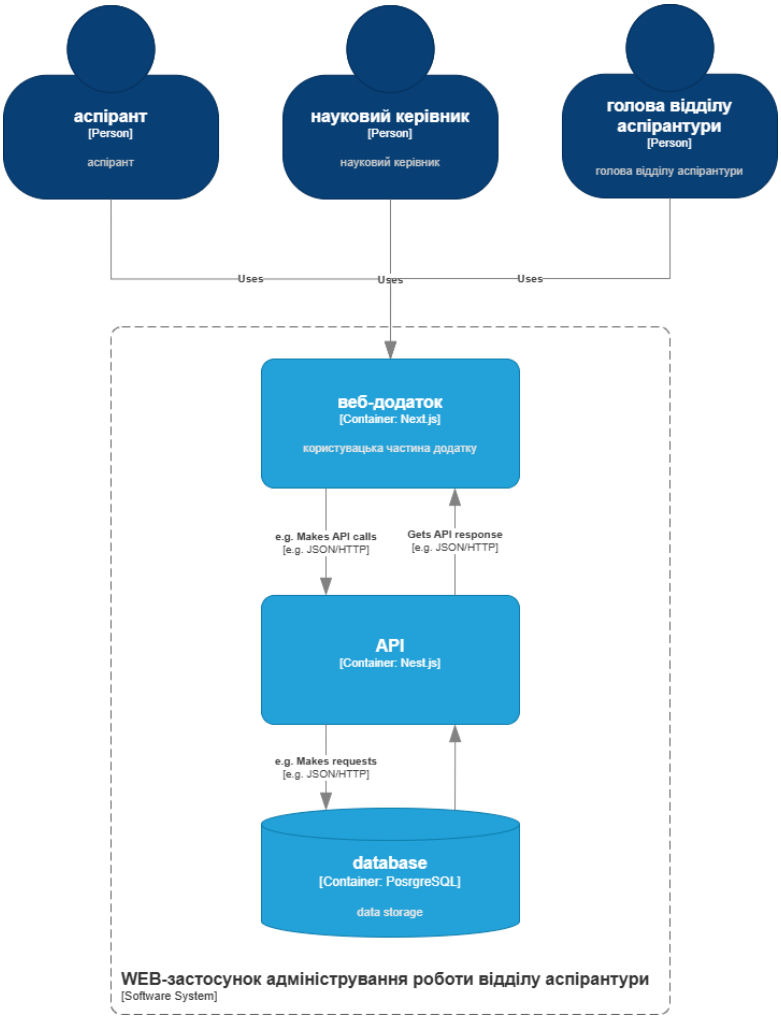


Рисунок 3.2 – Другий рівень діаграми C4

Деталізовану діаграму контейнера бекенду можна спостерігати на рисунку 3.3.

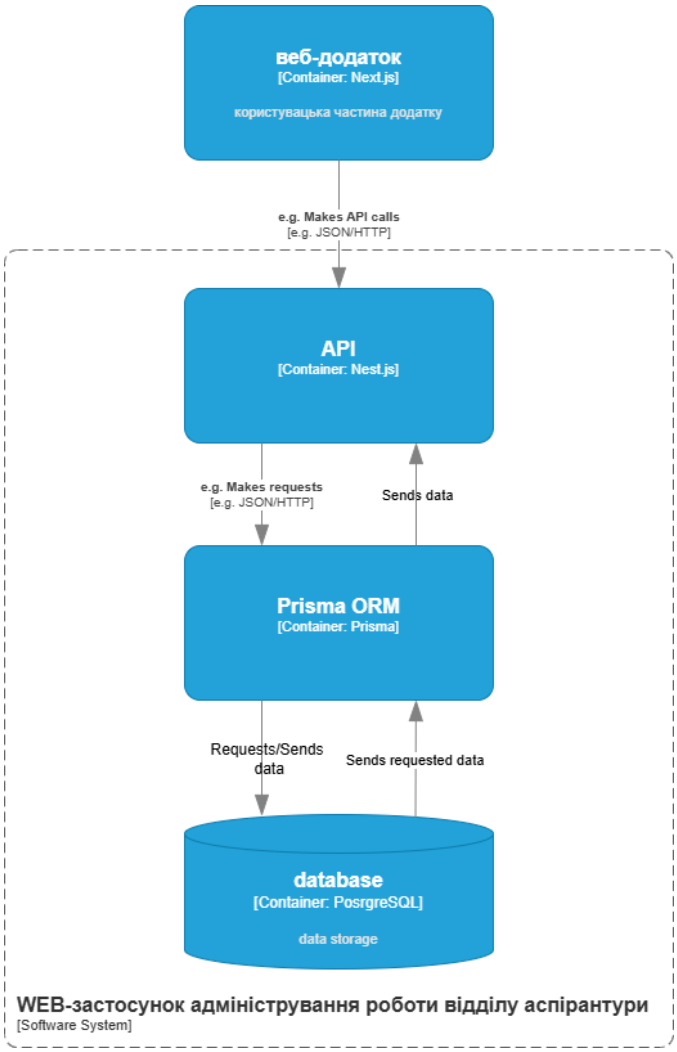


Рисунок 3.3 – Третій рівень діаграми C4

На четвертому рівні діаграми C4 відображено UML діаграму класів (рисунок 3.4).

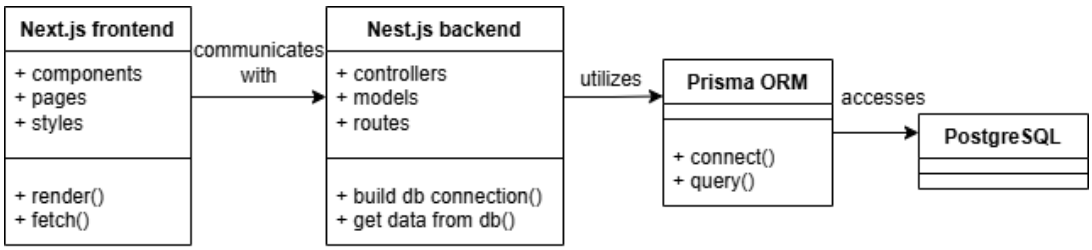


Рисунок 3.4 – Четвертий рівень діаграми C4

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

У галузі розробки вебзастосунків задля автоматизації адміністративних процесів існує низка поширених фронтенд- та бекенд-рішень, а також підходів до роботи з базами даних.

Серед React-орієнтованих інструментів виокремлюють Create React App, Gatsby, Remix та Next.js. CRA підходить для простих SPA, але без серверного рендерингу перше завантаження сповільнюється, а SEO страждає. Gatsby ефективний для сайтів із великою статичною інфраструктурою, проте створювати складні адмін-панелі на його основне дуже зручно. Remix забезпечує потужний роутінг і високу продуктивність, але рідше використовується й потребує вивчення власних шаблонів. Next.js «з коробки» підтримує SSR і SSG, пропонує файлову маршрутизацію й API-роути в одному проєкті та автоматично оптимізує ресурси, що робить його найкращим вибором для університетської адмін-панелі – сторінки завантажуються швидше, інтеграція з бекендом спрощується, а рутинні налаштування зводяться до мінімуму.

На сервері часто застосовують легкі HTTP-рушії Express або Koa, але вони не задають стандартної архітектури: у великих проєктах доводиться самостійно розбивати код на контролери, сервіси й репозиторії, додавати boilerplate для DI та валідації. Альтернативою є NestJS із модульною структурою, вбудованим контейнером залежностей, декларативними декораторами, підтримкою middleware, фільтрами винятків і готовою інтеграцією з OpenAPI (Swagger). Це дає чітку схему «controller-service-repository» та дозволяє зосередитися на бізнес-логіці замість налаштування фреймворку.

Щодо доступу до баз даних, традиційно в Node.js використовують ORM-бібліотеки типу TypeORM, Sequelize або Objection.js. Однак Prisma набирає популярність завдяки автогенерації типізованих клієнтів, прозорій системі міграцій і зрозумілій схемі даних. На відміну від TypeORM, котрий іноді потребує додаткових налаштувань для повноцінної типізації, Prisma забезпечує миттєве автодоповнення в IDE й гарантує відповідність запитів схемі бази даних.

Для зберігання даних у цьому проєкті обрано PostgreSQL – надійну реляційну СУБД з підтримкою ACID-транзакцій, розширеною роботою з JSON-типами та потужними механізмами індексації. PostgreSQL дозволяє ефективно обробляти складні запити, зберігати структуровані дані та гарантує цілісність інформації. У порівнянні з MySQL він пропонує розширені засоби роботи з геоданими й аналітичними запитами, а з MongoDB – гнучкіші можливості транзакцій та чіткішу схему даних, що критично важливо для точності обробки PhD-процесів. Інтеграція Prisma з PostgreSQL забезпечує зручний клієнтський API та надійну систему міграцій, що прискорює розробку й спрощує підтримку бази даних.

3.3 Конструювання програмного забезпечення

Далі наведено, чому для PostgradHub обрано стек Next.js + NestJS + Prisma + PostgreSQL, доповнений Docker-контейнеризацією[9]. Вибрані інструменти задовольняють вимоги до масштабованості, швидкості розробки й довгострокової підтримки.

3.3.1 Опис структури бази даних

В якості системи управління базами даних використовується PostgreSQL. База даних серверу призначена для зберігання користувачів, а також даних про їх академічну діяльність: індивідуальні плани та завдання, звіти аспірантів, заявки на захист дисертацій, зведені звіти для керівників та інформацію про групову приналежність. Модель бази даних наведена в графічному матеріалі, креслення 2.

Опис таблиць бази даних наведено у таблицях 3.1-3.8.

Таблиця 3.1 – Опис таблиці User

Назва поля	Тип даних	Опис
id	UUID	Унікальний ідентифікатор користувача

Продовження таблиці 3.1

Назва поля	Тип даних	Опис
group_id	UUID	Посилання на групу, до якої належить користувач
email	VARCHAR(255)	Адреса електронної пошти
password	VARCHAR(255)	Хеш пароля користувача
firstName	VARCHAR(100)	Ім’я користувача
second_name	VARCHAR(100)	По-батькові користувача
last_name	VARCHAR(100)	Прізвище користувача
createdAt	TIMESTAMPTZ	Дата й час створення запису
updatedAt	TIMESTAMPTZ	Дата й час останнього оновлення
deletedAt	TIMESTAMPTZ	Дата й час видалення
gender	VARCHAR(50)	Стать користувача
phone	VARCHAR(30)	Номер телефону користувача
role	VARCHAR(30)	Роль користувача

Таблиця 3.2 – Опис таблиці Group

Назва поля	Тип даних	Опис
id	UUID	Унікальний ідентифікатор групи
name	VARCHAR(100)	Назва групи
createdAt	TIMESTAMPTZ	Дата й час створення запису
updatedAt	TIMESTAMPTZ	Дата й час останнього оновлення

Таблиця 3.3 – Опис таблиці Assignment

Назва поля	Тип даних	Опис
id	UUID	Унікальний ідентифікатор завдання
user_id	UUID	Посилання на користувача (аспіранта)
title	VARCHAR(255)	Коротка назва завдання
subject	VARCHAR(255)	Назва дисципліни
description	TEXT	Детальний опис завдання
due_date	DATE	Крайній термін виконання
priority	VARCHAR(30)	Пріоритетність завдання
status	VARCHAR(30)	Статус завдання
requirements	TEXT[]	Вимоги до завдання
attachments	TEXT[]	Вкладення до завдання
author_id	TEXT	Посилання на користувача (керівника або адміна), що створював завдання
createdAt	TIMESTAMPTZ	Дата й час створення запису
updatedAt	TIMESTAMPTZ	Дата й час останнього оновлення
deletedAt	TIMESTAMPTZ	Дата й час видалення
isDeleted	BOOLEAN	Видалено завдання чи ні

Таблиця 3.4 – Опис таблиці Comment

Назва поля	Тип даних	Опис
id	UUID	Унікальний ідентифікатор коментаря

Продовження таблиці 3.4

Назва поля	Тип даних	Опис
text	TEXT	Текст коментаря
author_id	TEXT	Посилання на користувача, що залишив коментар
assignment_id	TEXT	Унікальний ідентифікатор завдання
createdAt	TIMESTAMPTZ	Дата й час створення коментаря
updatedAt	TIMESTAMPTZ	Дата й час останнього оновлення
deletedAt	TIMESTAMPTZ	Дата й час видалення
isDeleted	BOOLEAN	Видалено коментар чи ні

Таблиця 3.5 – Опис таблиці UserAssignment

Назва поля	Тип даних	Опис
id	UUID	Унікальний ідентифікатор звіту
user_id	TEXT	Посилання на студента, якому створено завдання
assignment_id	TEXT	Посилання на користувача, що опублікував завдання
assigned_at	VARCHAR(50)	Кому призначено завдання
due_date	TIMESTAMPTZ	Дата й час створення запису

Таблиця 3.6 – Опис таблиці Booking

Назва поля	Тип даних	Опис
id	UUID	Унікальний ідентифікатор запису
start_date	TIMESTAMP	Дата й час початку запису
end_date	TIMESTAMP	Дата й час завершення запису
status	VARCHAR(30)	Статус запису
description	TEXT	Опис запису
user_id	TEXT	Посилання на студента, що створив запис
approved_by	TEXT	Посилання на користувача (керівника або адміністратора), що підтвердив запис
createdAt	TIMESTAMPTZ	Дата й час створення запису
updatedAt	TIMESTAMPTZ	Дата й час останнього оновлення
deletedAt	TIMESTAMPTZ	Дата й час видалення
isDeleted	BOOLEAN	Видалено запис чи ні

Таблиця 3.7 – Опис таблиці Task

Назва поля	Тип даних	Опис
id	UUID	Унікальний ідентифікатор задачі
name	TEXT	Назва задачі
description	TEXT	Опис задачі
deadline	TIMESTAMP	Дедлайн задачі

Продовження таблиці 3.7

Назва поля	Тип даних	Опис
status	VARCHAR(30)	Статус задачі
priority	VARCHAR(30)	Пріоритетність задачі
user_id	TEXT	Посилання на студента
created_by_id	TEXT	Посилання на користувача (керівника або адміністратора), яким була створена задача
createdAt	TIMESTAMPTZ	Дата й час створення задачі
updatedAt	TIMESTAMPTZ	Дата й час останнього оновлення
deletedAt	TIMESTAMPTZ	Дата й час видалення
isDeleted	BOOLEAN	Видалено задачу чи ні

Таблиця 3.8 – Опис таблиці Document

Назва поля	Тип даних	Опис
id	UUID	Унікальний ідентифікатор документу
type	VARCHAR(30)	Тип документу
name	TEXT	Назва докусенту
status	VARCHAR(30)	Статус документу
file_url	TEXT	Посилання на документ
createdAt	TIMESTAMPTZ	Дата й час створення задачі
updatedAt	TIMESTAMPTZ	Дата й час останнього оновлення

Продовження таблиці 3.8

Назва поля	Тип даних	Опис
deletedAt	TIMESTAMPTZ	Дата й час видалення
isDeleted	BOOLEAN	Видалено задачу чи ні

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.9.

Таблиця 3.9 – Опис утиліт

№ п/п	Назва утиліти	Опис утиліти
1	Postman[7]	Програмне забезпечення необхідне для тестування REST запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.
2	Visual Studio Code[8]	Основне середовище для фронтенд-розробки вебзастосунку з використанням Next.js та TypeScript.
3	React	JavaScript-бібліотека, яка використовується для побудови компонентного інтерфейсу користувача. Забезпечує реактивність і повторне використання UI.
4	Next.js	Фреймворк для React, що забезпечує серверний рендеринг, маршрутизацію, оптимізацію продуктивності, підтримку API-маршрутів «з коробки».
5	NestJS	Серверний фреймворк на Node.js, побудований на базі Express. Використовується для реалізації REST API, бізнес-логіки та взаємодії з БД.
6	Prisma Studio	Інтерфейс для взаємодії з базою даних під час розробки, перегляду, тестування та редагування записів.

Продовження таблиці 3.9

7	PostgreSQL	Система керування базами даних, у якій зберігаються всі дані користувачів, завдань, публікацій, звітів та іншої навчальної інформації.
8	Docker	Платформа контейнеризації, що використовувалася для ізоляції середовища розробки та спрощення розгортання застосунку.
9	Swagger (OpenAPI)	Інструмент для документування та інтерактивного тестування REST API, автоматично інтегрований у NestJS.
10	ESLint + Prettier	Інструменти для перевірки синтаксису та автоматичного форматування коду відповідно до єдиного стилю.
11	Figma	Онлайн-сервіс для розробки UX/UI-дизайну вебзастосунку. Використовувався для створення макетів сторінок та погодження інтерфейсів.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Безпека відомостей є надзвичайно важливим аспектом у розробці вебзастосунку, що обробляє особисту, академічну та адміністративну інформацію здобувачів наукового ступеня. У процесі розробки програмного забезпечення були враховані сучасні потреби до захисту даних, відповідно до принципів «security by design» та стандартів OWASP.

На рівні аутентифікації застосунок використовує механізм JWT (JSON Web Token) для захищеного керування сесіями користувачів. Паролі зберігаються у хешованому вигляді з використанням криптографічного алгоритму

bcrypt. Захист від SQL-ін’єкцій забезпечується за рахунок використання ORM Prisma, яка автоматично параметризує запити до бази даних PostgreSQL.

Усі критично важливі запити до API захищено middleware-шарами автентифікації та авторизації, які перевіряють права доступу в залежності від ролі користувача (аспірант, керівник, адміністратор).

На етапі розробки проводилось ручне тестування вразливостей з використанням Postman та інтегрованих інструментів браузера. Інтерфейс адміністратора з чутливими функціями доступний лише для авторизованих сесій, що проходять перевірку на валідність та термін дії токенів.

Отже, у вебзастосунку реалізовано базові та розширені механізми забезпечення конфіденційності, цілісності та доступності відомостей, що гарантує безпечну експлуатацію системи в умовах реального використання.

Висновки до розділу

У третьому розділі було виконано детальний аналіз архітектури та втілення програмного забезпечення, призначеного для автоматизації праці відділу аспірантури. Обґрунтовано вибір клієнт-серверної архітектури, яка забезпечує розділення обов’язків між фронтендом і бекендом, сприяє гнучкості розгортання та полегшує масштабування системи.

Для реалізації інтерфейсної частини було використано бібліотеку React та фреймворк Next.js, що забезпечують високу продуктивність . Серверна логіка втілена на базі NestJS – фреймворку, котрий забезпечує модульну структуру, підтримку контролерів, сервісів, репозиторіїв, а також інтеграцію з Swagger для документування API. Як систему управління базами даних використано PostgreSQL, а для роботи з нею – ORM Prisma, що гарантує типізовану взаємодію та зручність розробки.

Розглянуто структуру бази даних, включаючи опис основних таблиць, що відповідають за зберігання необхідної інформації про користувачів, завдання, звіти, публікації та заявки на захист дисертацій.

Окрему увагу приділено безпеці даних: впроваджено аутентифікацію через JWT, шифрування паролів, захист від SQL-ін’єкції, контроль доступу за ролями, валідацію даних та централізовану обробку помилок.

Також представлено набір утиліт та інструментів, що застосовувалися під час розробки, включно з Postman, Docker, Prisma Studio, NestJs, Next.js, системами контролю версій та інтерфейсами для дизайну.

Отже, на основі сформованих вимог було втілено технічне ядро системи PostgradHub, що закладає фундамент для ефективного управління процесами аспірантури, забезпечує надійність, масштабованість і безпечну роботу програмного забезпечення.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Якість програмного забезпечення є одним із ключових показників його надійності, зручності у використанні та придатності до подальшого супроводу й масштабування. Для об’єктивної оцінки якості розробленого веб-додатку було використано інструменти автоматизованого аналізу, зокрема SonarCloud[13] та Lighthouse[12]. Вони дозволяють перевірити кодову базу за низкою технічних і користувацьких критеріїв, охоплюючи як внутрішні аспекти (структура коду та наявність помилок), так і зовнішні (продуктивність, доступність, дотримання рекомендацій та стандартів). На рисунках 4.1 та 4.2 наведено звіти SonarCloud та LightHouse відповідно.

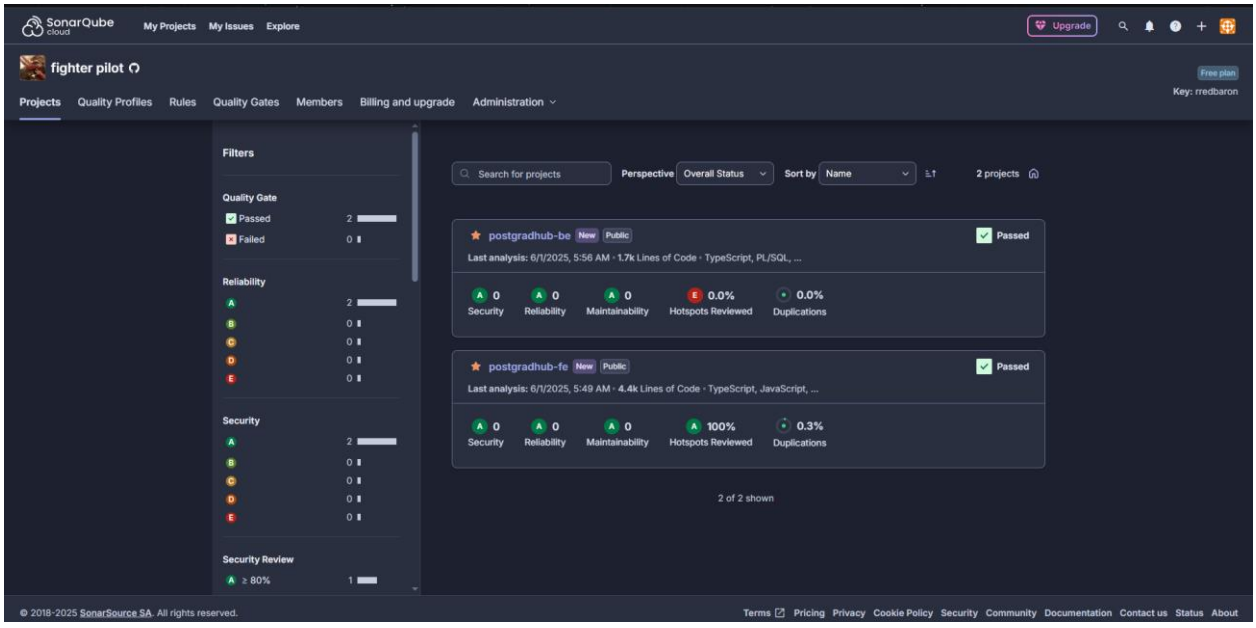


Рисунок 4.1 – Звіт SonarCloud

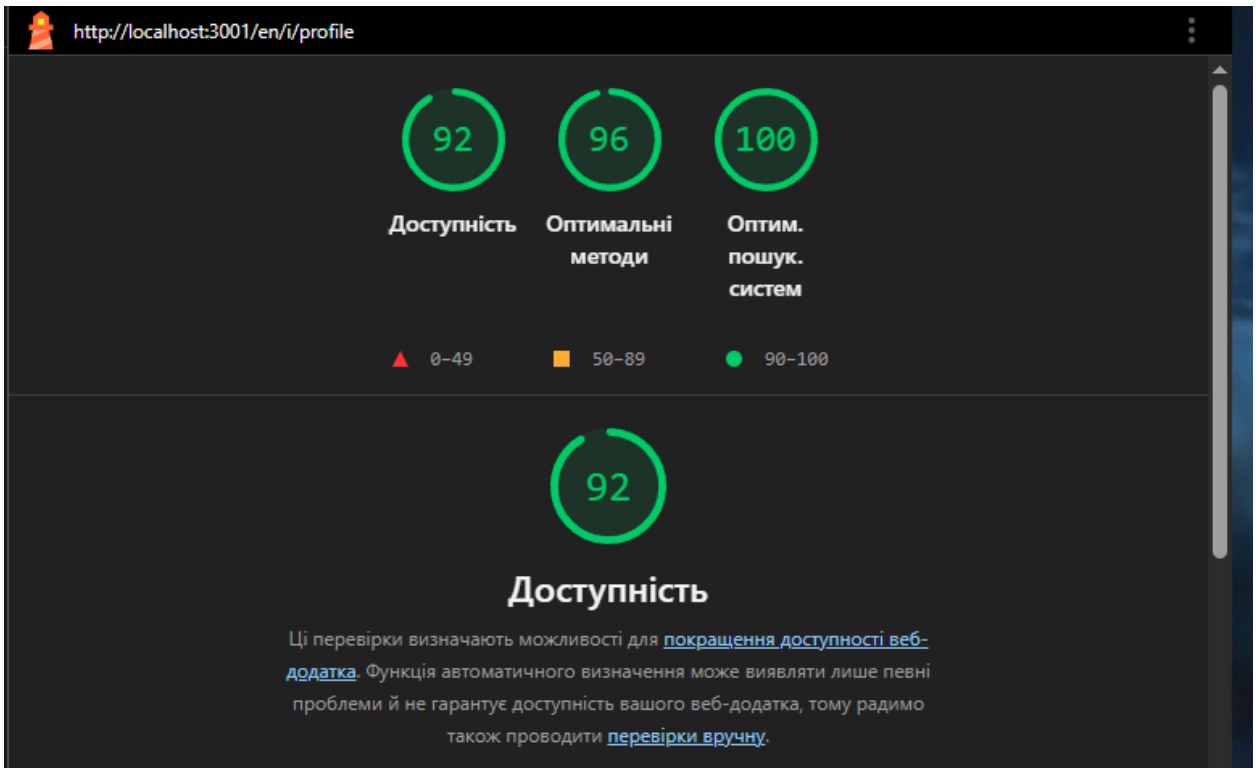


Рисунок 4.2 – Звіт LightHouse з продуктивності

Аналіз у SonarCloud показав, що як фронтенд-, так і бекенд-частини проєкту успішно пройшли перевірку якості зі статусом «Passed», без виявлених критичних помилок, вразливостей або серйозних недоліків. Рівень підтримуваності коду отримав найвищу оцінку – A. У фронтенді виявлено лише 3 некритичні code smells, тоді як у бекенді їх 16, проте всі вони не впливають на функціонал. Важливим результатом є повне покриття потенційно ризикованих ділянок коду (security hotspots) – 100%, що підтверджує високий рівень безпеки. Крім того, дублювання коду мінімальне: у фронтенді воно становить лише 0,3%, а в бекенді взагалі відсутнє (0%), що свідчить про якісну структуру та читабельність коду. Додатковий аудит за допомогою Lighthouse підтвердив високий рівень оптимізації веб-додатку: сторінка профілю отримала 92 бали за доступність, 96 – за відповідність найкращим практикам і максимальні 100 балів за SEO-оптимізацію. Ці результати демонструють, що додаток не лише технічно якісний, але й зручний для користувачів, безпечний та готовий до ефективної роботи.

4.2 Опис процесів тестування

Тестування виконується згідно з документом «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.11.

Таблиця 4.1 – Тест 1.1 Реєстрація користувача

Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні дані	Електронна пошта, ім’я, прізвище, номер телефону, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка до цього не була зареєстрована в системі, ім’я, прізвище, валідний номер телефону, пароль від 6 символів. Після цього натискається кнопка підтвердження реєстрації.
Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на головну сторінку додатку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Авторизація користувача

Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні дані	Електронна пошта та пароль зареєстрованого користувача
Опис проведення тесту	У відповідні поля вводяться зареєстровані в системі еклектронна пошта та пароль.
Очікуваний результат	Авторизація проходить успішно, користувач перенаправляється на головну сторінку додатку.

Продовження таблиці 4.2

Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Зміна мови інтерфейсу

Початковий стан системи	Користувач знаходиться на головній сторінці додатку
Вхідні дані	Дві доступні для вибори мови інтерфейсу – українська та англійська.
Опис проведення тесту	В навігаційній панелі користувач обирає з випадаючого списку бажану мову інтерфейсу.
Очікуваний результат	Мову інтерфейсу змінено.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 2.1 Створення завдання (для керівника)

Початковий стан системи	Користувач авторизований в додатку із роллю «Науковий керівник» та знаходиться на сторінці керування завданнями.
Вхідні дані	Назва завдання, строки його подання, опис завдання, список вимог, перелік прикріплених файлів (за необхідністю).
Опис проведення тесту	Користувач заповнює форму створення завдання валідними даними та натискає на кнопку «Створити».
Очікуваний результат	Завдання створено.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 2.2 Призначення студентів до завдання

Початковий стан системи	Користувач авторизований в додатку із роллю «Науковий керівник» та знаходиться на сторінці керування завданнями.
Вхідні дані	Створене завдання та список доступних для призначення студентів.
Опис проведення тесту	Користувач обирає студентів для закріплення зі спадаючого меню.
Очікуваний результат	Студента успішно прикріплено до завдання.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 3.1 Завантаження пропозиції теми дисертації

Початковий стан системи	Користувач авторизований в додатку із роллю «Студент» та знаходиться на сторінці «Дисертація».
Вхідні дані	Файл з пропозицією теми.
Опис проведення тесту	Користувач завантажує файл із запропонованою темою.
Очікуваний результат	Студента успішно прикріплено до завдання.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 3.2 Додавання нового завдання до плану дисертації

Початковий стан системи	Користувач авторизований в додатку із роллю «Студент» та знаходиться на сторінці «Дисертація» на вкладці плану.
Вхідні дані	Назва завдання, термін виконання, опис, пріоритет.
Опис проведення тесту	Користувач вводить коректні дані та натискає на кнопку «Створити».

Продовження таблиці 4.7

Очікуваний результат	Завдання успішно створено та з’явилося в списку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 3.3 Додавання публікації

Початковий стан системи	Користувач авторизований в додатку із роллю «Студент» та знаходиться на сторінці додавання публікацій.
Вхідні дані	Назва публікації та файл з нею.
Опис проведення тесту	Користувач завантажує файл із публікацією та вказує назву, натискає кнопку «Опублікувати».
Очікуваний результат	Публікацію успішно додано.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 4.1 Бронювання дати захисту

Початковий стан системи	Користувач авторизований в додатку із роллю «Студент» та знаходиться на сторінці календарю захистів.
Вхідні дані	Дата та час захисту, які не пересікаються з іншими.
Опис проведення тесту	Користувач обирає вільні дату та час в календарі.
Очікуваний результат	Дата та час захисту успішно заброньовані.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 5.1 Призначення керівника для студента (для голови відділу аспірантури)

Початковий стан системи	Користувач авторизований в додатку із роллю «Голова відділу аспірантури» та знаходиться на сторінці управління студентами.
Вхідні дані	Обрані студент та керівник для нього.
Опис проведення тесту	Користувач обирає студента та керівника для нього, натискає на «Підтвердити».
Очікуваний результат	Керівника закріплено за студентом успішно.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 6.1 Перегляд списку студентів (для керівника)

Початковий стан системи	Користувач авторизований в додатку із роллю «Науковий керівник» та знаходиться на сторінці управління студентами.
Вхідні дані	-
Опис проведення тесту	Користувач переходить на сторінку зі списком студентів та має змогу переглянути список.
Очікуваний результат	Список студентів з їхнім прогресом відображається коректно
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

Розглянемо в якості контрольного прикладу авторизацію студента в системі, перегляд та здачу завдань на перевірку, подання теми дисертації на

затвердження, створення власних нотаток щодо наукової роботи та бронювання дати захисту.

Почнемо з авторизації в системі. Користувач ввів коректні дані та натиснув на кнопку «Sign in» (рис. 4.3).

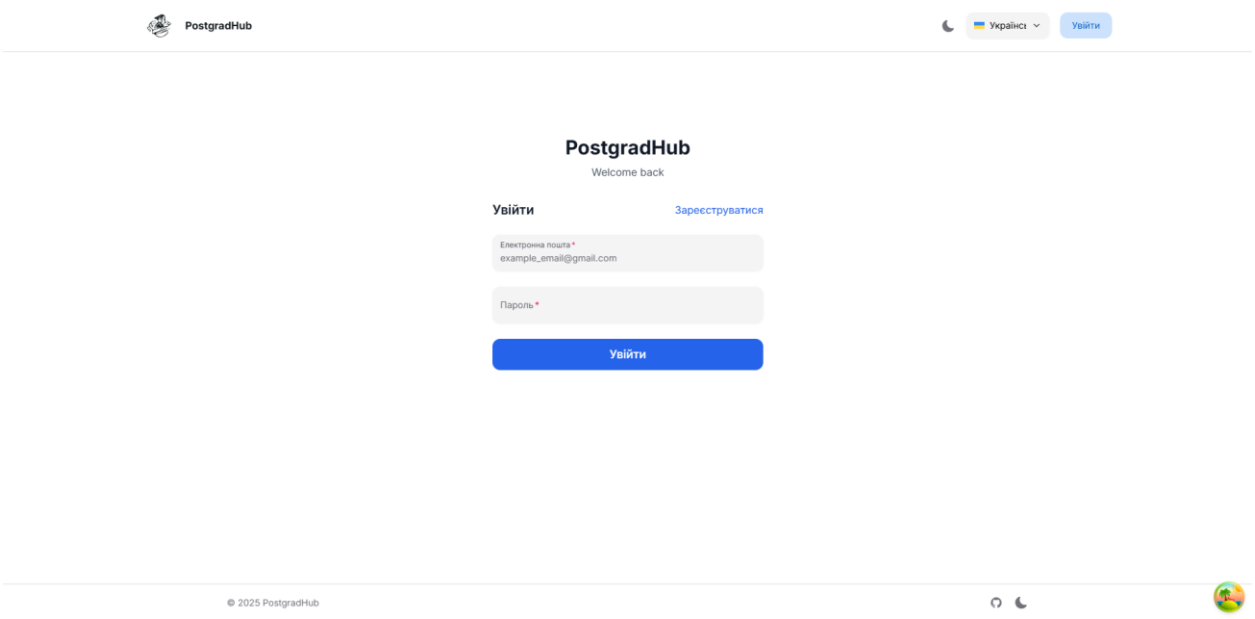


Рисунок 4.3 – Форма авторизації

Після цього користувач переходить на головну сторінку застосунку (рис. 4.4).

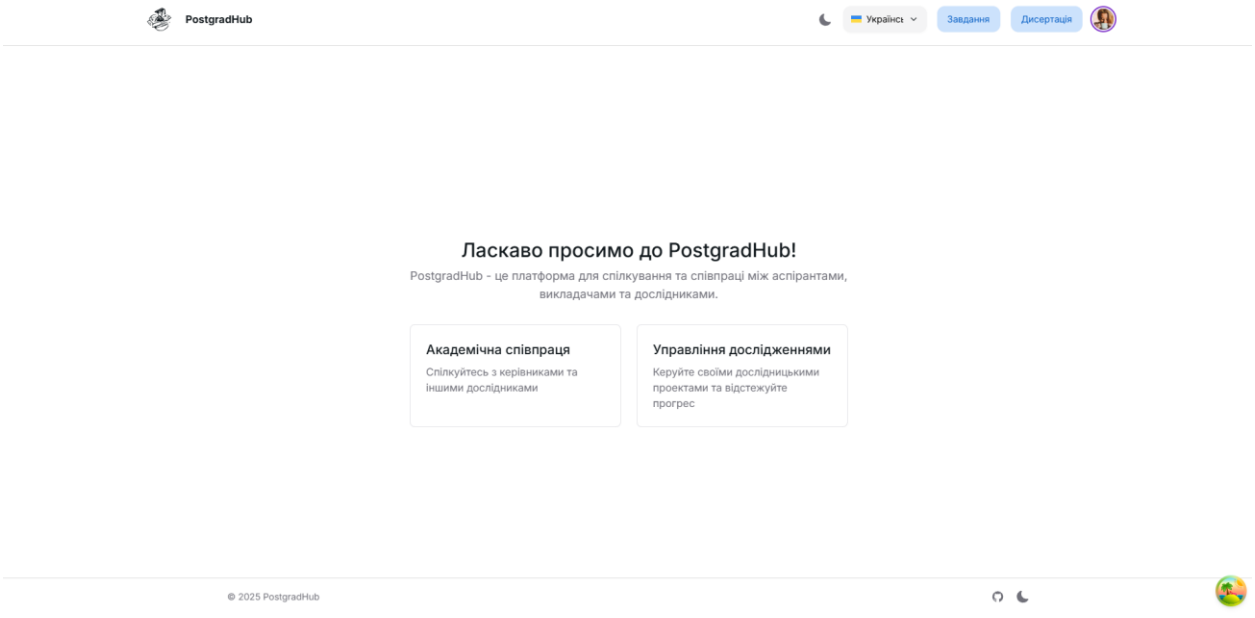


Рисунок 4.4 – Головна сторінка застосунку

Користувач натискає на кнопку «Assignments» і переходить на сторінку перегляду своїх завдань (рис. 4.5).

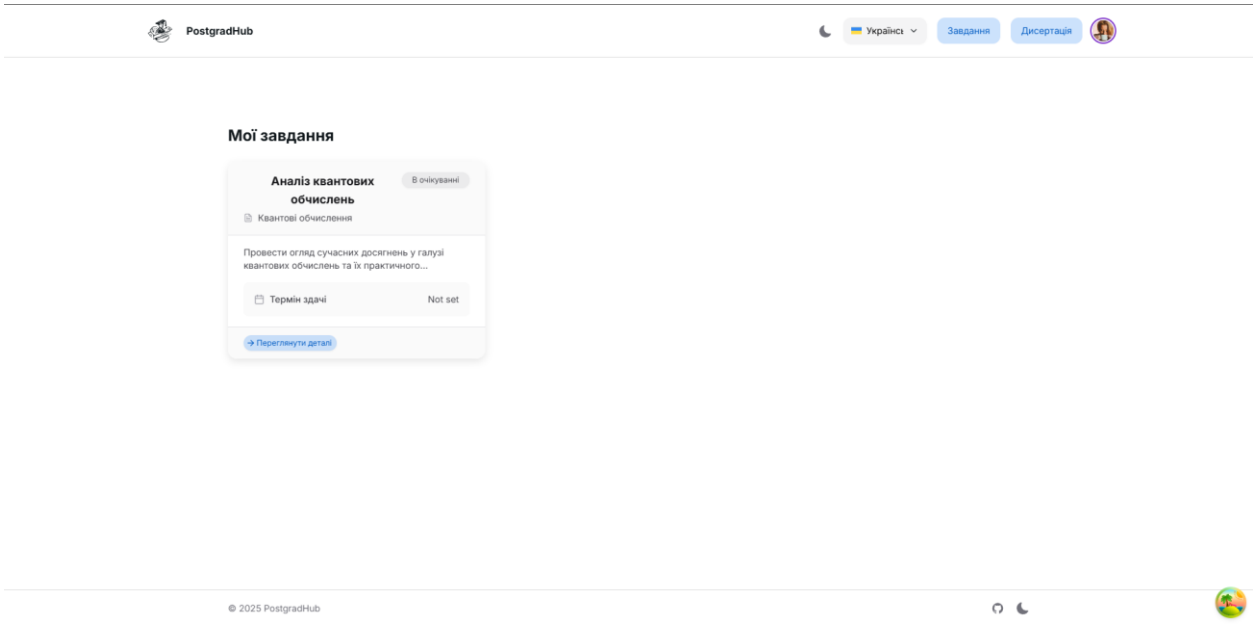


Рисунок 4.5 – Сторінка перегляду списку завдань

Користувач обирає перше завдання зі списку та переходить на сторінку детального опису завдання (рис. 4.6).

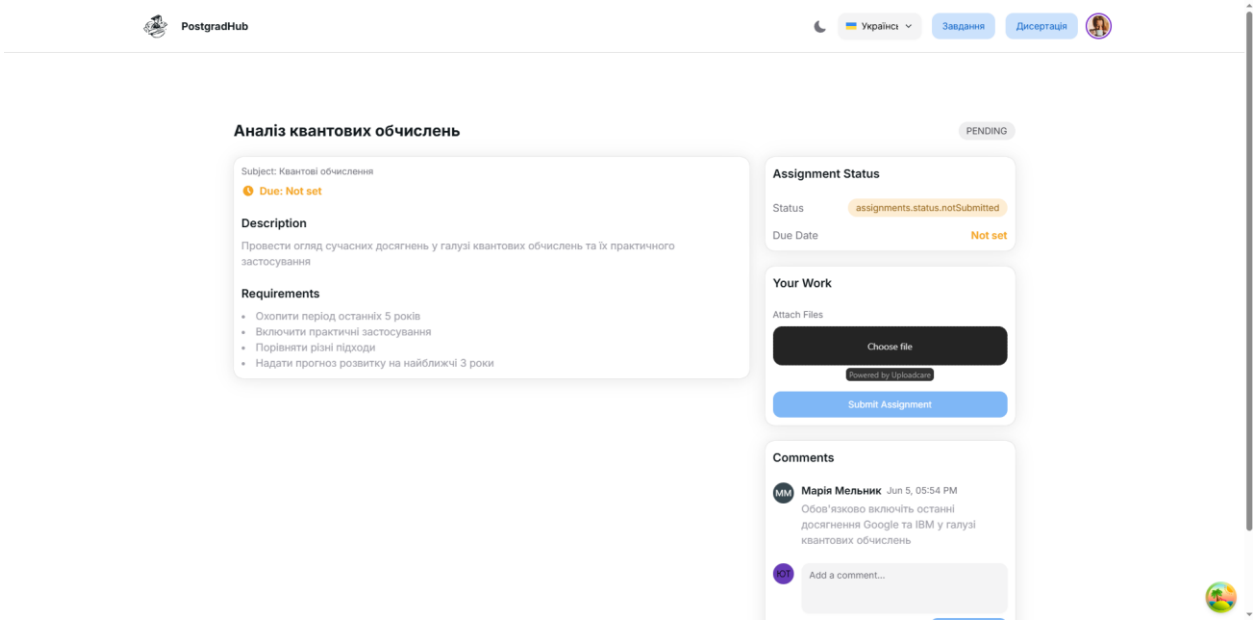


Рисунок 4.6 – Сторінка детального перегляду завдання

Користувач завантажує файл із виконаним завданням і натискає на кнопку «Submit Assignment» (рис. 4.7). Завдання здано.

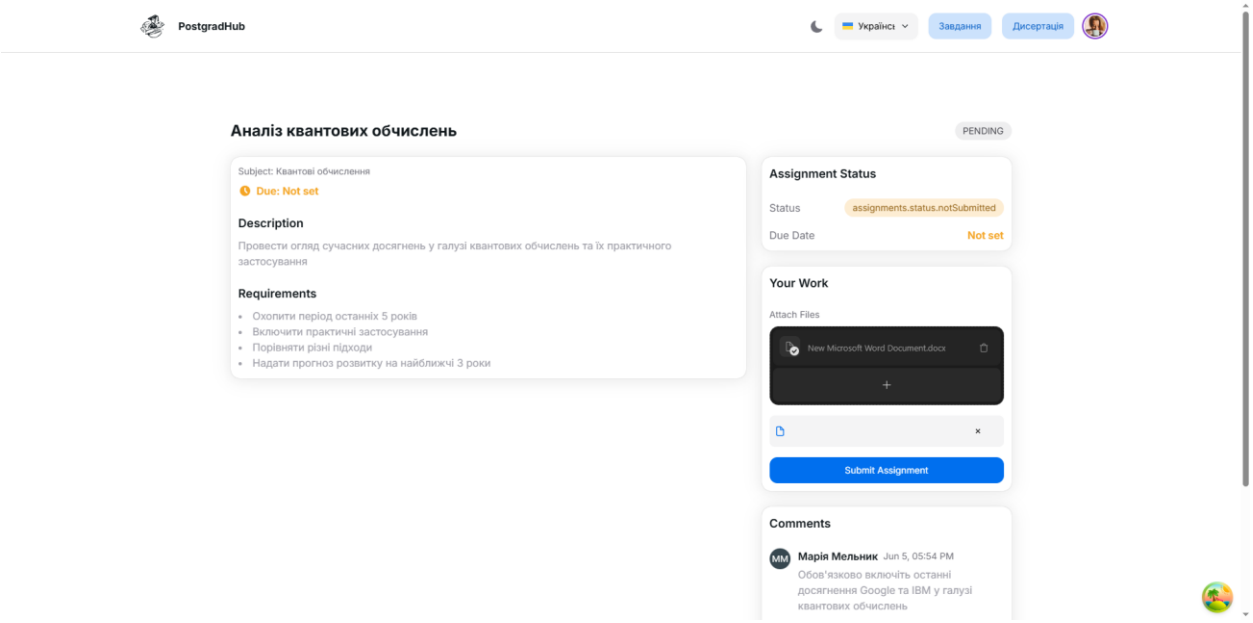


Рисунок 4.7 – Файл завантажено, завдання здано на перевірку

Користувач натискає на кнопку «Dissertation» та потрапляє на сторінку з централізованим керуванням дисертацією. Користувач завантажує файл із темою роботи (рис. 4.8).

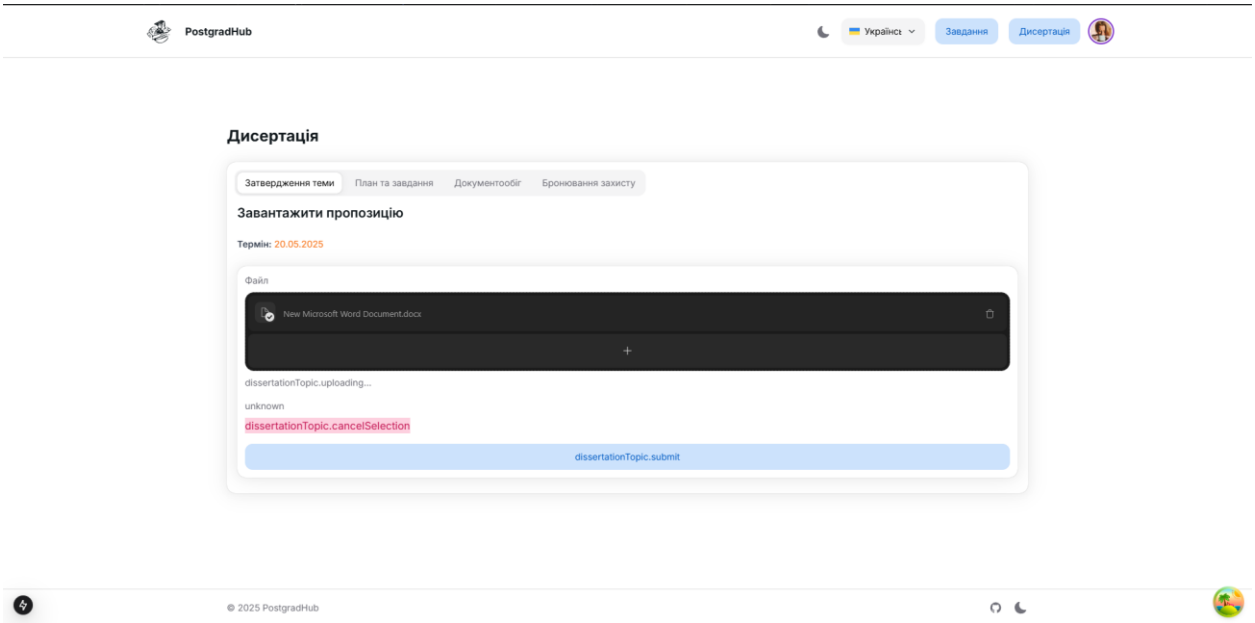


Рисунок 4.8 – Сторінка завантаження файлу з темою роботи

Користувач переходить на вкладку «Plan & Tasks» та переглядає створені ним раніше завдання, доступні лише йому (рис. 4.9).

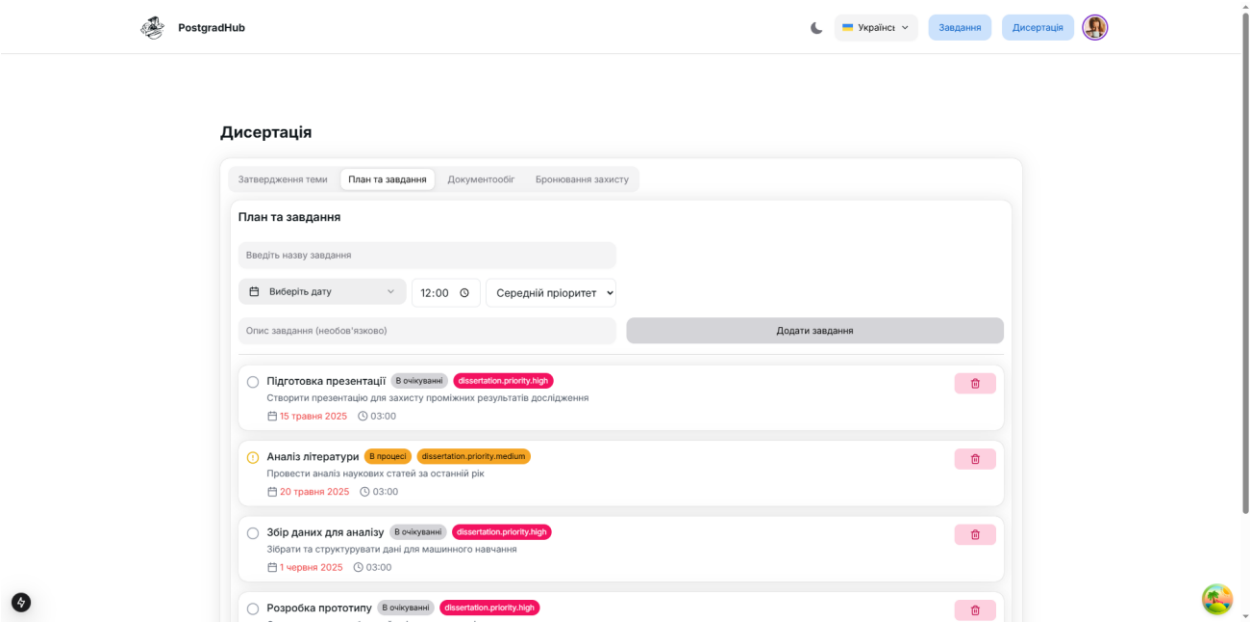


Рисунок 4.9 – Сторінка створення та перегляду власних завдань студента

Користувач переходить на вкладку «Document Flow» та завантажує призначені керівником документи (рис. 4.10).

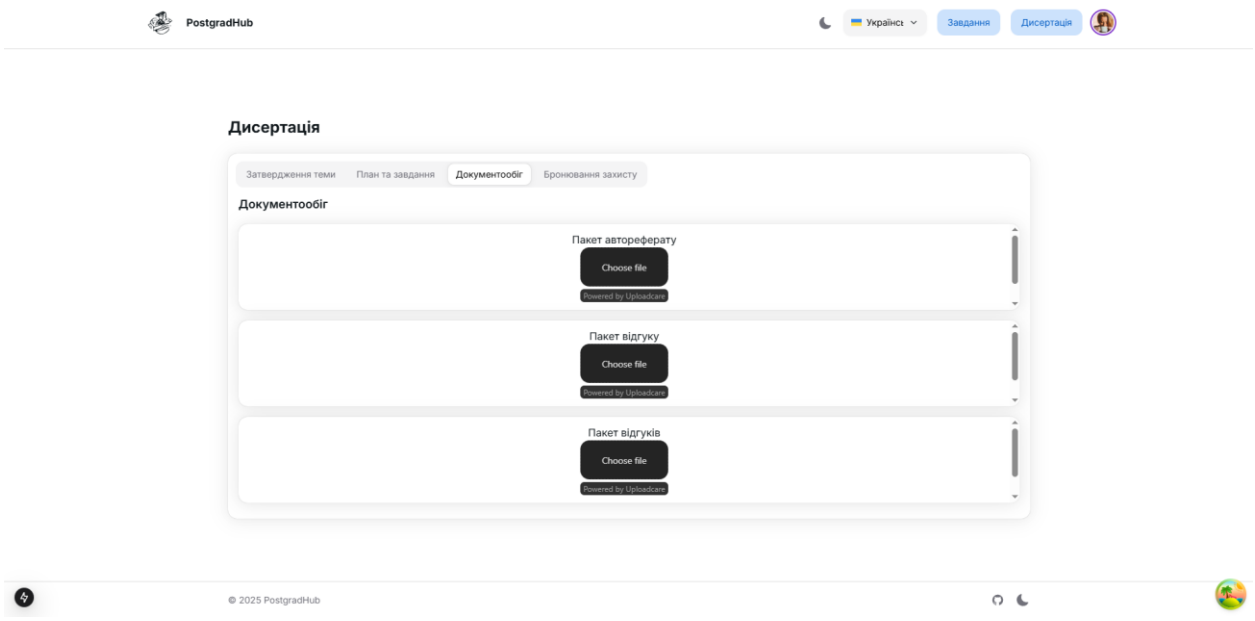


Рисунок 4.10 – Сторінка завантаження документів

Користувач переходить на вкладку «Defense Booking» та бронює дату захисту роботи (рис. 4.11).

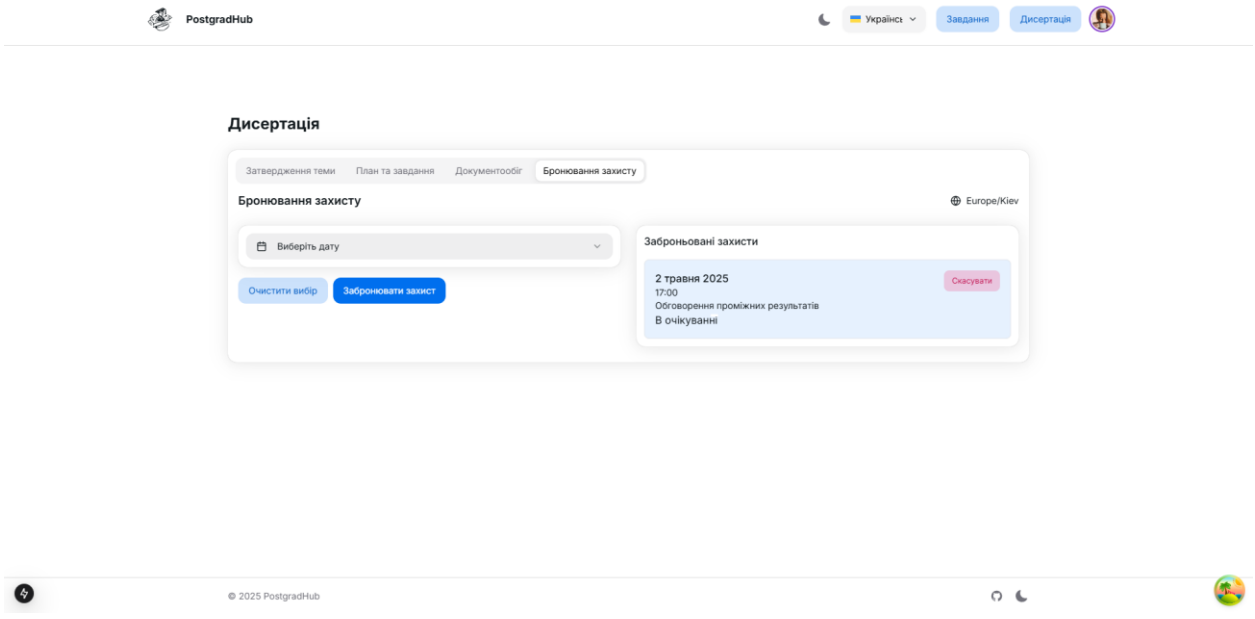


Рисунок 4.11 – Сторінка бронювання дати захисту роботи

Після виконання цих дій контрольний приклад завершено успішно.

Висновки до розділу

У результаті проведеного аналізу якості та тестування додатку можна зробити висновок, що розроблений вебдодаток задовольняє вимоги до надійності, безпеки та зручності використання. Автоматизований аналіз SonarCloud не виявив критичних помилок, уразливостей чи дублікацій коду - рівень підтримуваності оцінений найвищою «A», а покриття security hotspots становить 100%. Інструмент Lighthouse підтвердив, що фронтенд оптимізовано: сторінка продемонстрували високі бали доступності (92), відповідності найкращим практикам (96) та SEO (100). Це свідчить про читабельний та безпечний код, готовий до подальшого розвитку та масштабування.

Мануальне тестування показало, що всі ключові функціональні сценарії (реєстрація, авторизація, зміна мови, створення та призначення завдань, робота зі сторінками дисертації, бронювання захисту тощо) виконуються коректно й відповідають очікуваним результатам.

Контрольний приклад демонструє послідовне виконання основних операцій користувача-студента: від авторизації та перегляду завдань до подання файлів, формування плану дисертації та бронювання дати захисту. Кожен із кроків відображено на відповідних сторінках додатку.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для локального розгортання застосунку на машині мають бути встановлені NodeJS, Docker та Docker compose, Git. Спочатку необхідно клонувати репозиторій за допомогою команди git clone <URL_репозиторію>. Після цього перейдіть у папку проекту, де знаходяться обидві частини застосунку.

В кореневій директорії backend частини додатку необхідно створити .env файл з вмістом, наведеним на рисунку 5.1.

Рисунок 5.1 – Вміст файлу середовища

Після цього запустимо докер-образ за допомогою команди docker-compose up -d. В результаті її виконання збирається образ NestJS додатку, запускається контейнер PostgreSQL, виконуються міграції Prisma. Docker-compose наведено на рисунку 5.2.

```yaml
services:
  postgradhub-be:
    image: node:latest
    container_name: postgradhub-be-app
    restart: unless-stopped
    depends_on:
      - postgradhub-be-postgres
    working_dir: /app
    ports:
      - '3000:3000'
      - '9229:9229'
    env_file:
      - .env
    volumes:
      - .:/app
    command: sh -c "npm install && npm run migration && npm run start:dev"

  postgradhub-be-postgres:
    image: postgres:latest
    container_name: postgradhub-be-postgres
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    volumes:
      - postgradhub-be-pgdata:/var/lib/postgresql/data
    ports:
      - '5432:5432'

volumes:
  postgradhub-be-pgdata:
```

Рисунок 5.2 – Вміст докер-файлу

Якщо попередні кроки були виконані, то відкривши в браузері посилання http://localhost:3000/api можна побачити swagger-документацію (рис. 5.3).

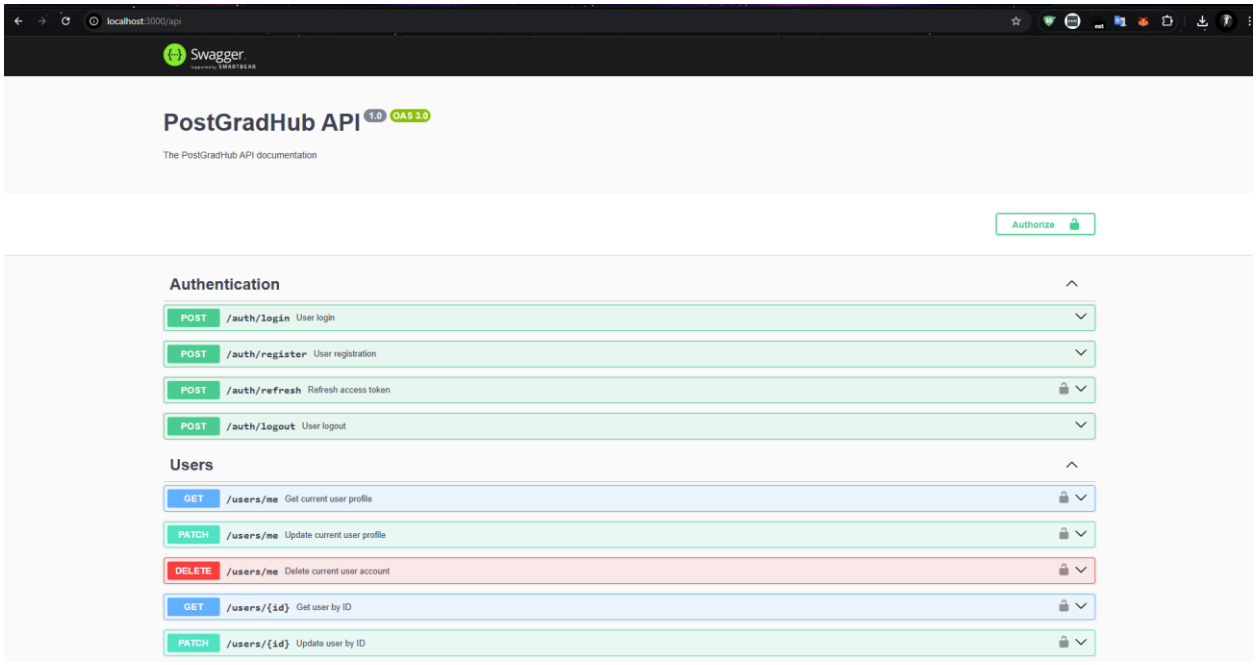


Рисунок 5.3 – Swagger-документація бекенд частини додатку

Після розгортання бекенд частини перейдемо до фронтенду. В кореневій директорії фронтенд додатку створимо файл .env.local та вкажемо в ньому змінну NEXT_PUBLIC_API_URL (див. рисунок 5.4).

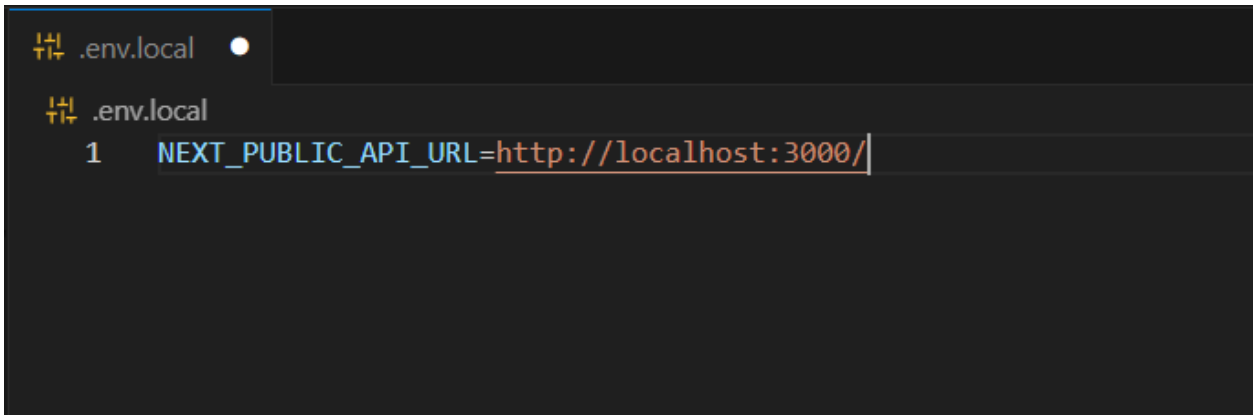


Рисунок 5.4 – Вміст файлу .env.local

Після цього запустимо команду npm i, яка встановить усі необхідні пакети, зазначені в package.json файлі. Після цього виконаємо команду npm run dev. Якщо усі етапи, зазначені вище, виконані правильно, в консолі побачимо повідомлення про коректний запуск фронтенд частини додатку (див. рисунок 5.5).

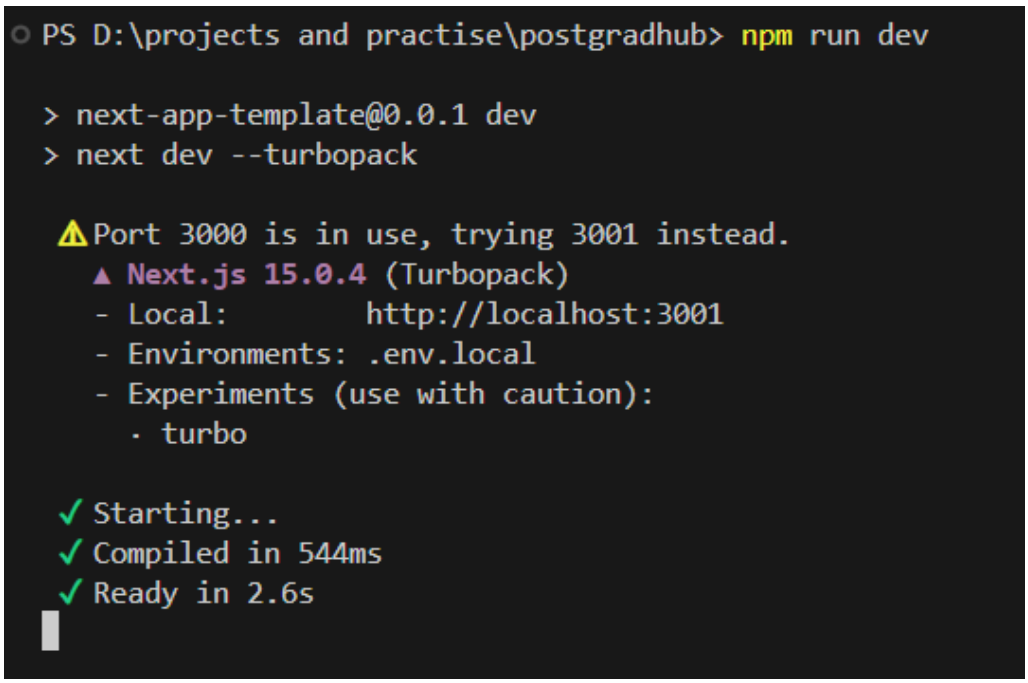


Рисунок 5.5 – Логи в консолі в разі успішного запуску

Архітектура застосунку передбачає взаємодію браузера з Next.js-фронтендом, який спілкується з Nest.js-бекендом, а той, у свою чергу, використовує PostgreSQL для зберігання даних. Діаграму розгортання наведено на рисунку 5.6.

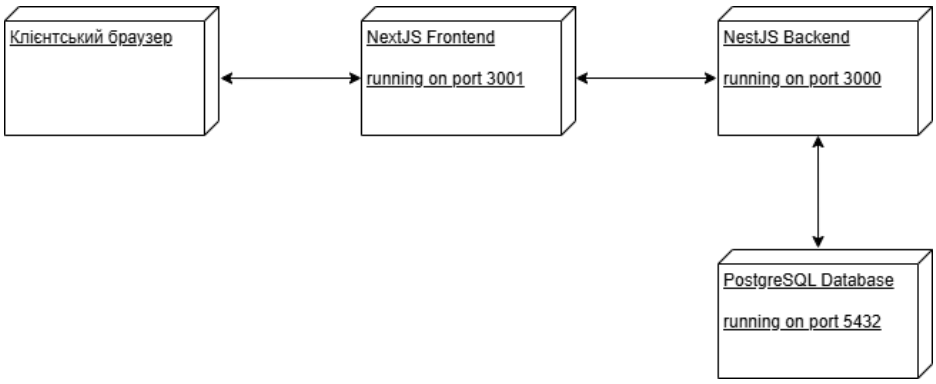


Рисунок 5.6 – Діаграма розгортання

Таким чином в разі коректного виконання усіх кроків додаток готовий до роботи.

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі.

Для підтримки та автоматизації оновлень проекту використовується GitHub Actions. Коли в гілку main надходить коміт з тегом версії (наприклад, v1.0.0), автоматично запускається процес збірки. Для фронтенду на Next.js виконується next build, для бекенду на Nest.js - nest build. Зібрані артефакти автоматично деплояться на відповідні хостинги: фронтенд на Vercel, а бекенд - на обрану платформу. Це дозволяє користувачам завжди отримувати актуальну версію застосунку без ручного втручання.

Висновки до розділу

У даному розділі було детально розглянуто процес розгортання та супроводу програмного забезпечення. Наведено покрокову інструкцію з локального встановлення додатку, що включає налаштування середовища, запуск

бекенд- та фронтенд-частин за допомогою Docker та Node.js. Особливу увагу приділено автоматизації процесів завдяки використанню GitHub Actions, що дозволяє забезпечити безперервну інтеграцію та доставку нових версій. Запропонована архітектура розгортання демонструє ефективну взаємодію між компонентами системи. Впроваджені механізми підтримки та оновлення гарантують стабільну роботу застосунку та оперативне реагування на зміни, що є важливим аспектом сучасного програмного забезпечення.

ВИСНОВКИ

У ході виконання дипломного проекту було розроблено вебзастосунок для адміністрування роботи відділу аспірантури (PostgradHub), який автоматизує ключові процеси, пов’язані з підготовкою аспірантів у закладах вищої освіти. Система забезпечує централізований облік даних, управління індивідуальними планами робіт, подання та узгодження звітів, бронювання дат захистів дисертацій, а також аналітику для адміністрації.

Метою проекту було створення інструменту, який підвищить ефективність управління аспірантурою шляхом автоматизації рутинних процесів. Всі поставлені завдання виконано:

Проведено аналіз предметної області та існуючих рішень, що підтвердило необхідність спеціалізованого продукту.

Розроблено архітектуру системи на основі Next.js (фронтенд), NestJS (бекенд) та PostgreSQL (БД), що забезпечує масштабованість і безпеку.

Реалізовано функціональні модулі для різних ролей (аспірант, науковий керівник, голова відділу аспірантури), включаючи:

– реєстрацію/авторизацію;
– управління завданнями та звітами;
– календар захистів;
– аналітичні панелі.

Запроваджено механізми безпеки: JWT-аутентифікацію, шифрування даних, контроль доступу.

Протестовано якість ПЗ за допомогою SonarCloud, Lighthouse та ручних тестів, що підтвердило відповідність вимогам.

Розроблений застосунок покриває всі ключові процеси аспірантури, зменшуючи час на адміністративні операції та знижуючи ризик помилок. Система є користувацьки орієнтованою, що підтверджено високими оцінками за зручність інтерфейсу (Lighthouse: 92+ балів за доступність).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) React. The library for web and native user interfaces. URL: https://react.dev/ (дата звернення: 13.05.2025).

2) Next.js by Vercel - The React Framework. URL: https://nextjs.org/ (дата звернення: 13.05.2025).

3) NestJS - A progressive Node.js framework. URL: https://nestjs.com/ (дата звернення: 13.05.2025).

4) Prisma | Instant Postgres plus an ORM for simpler db workflows. URL: https://www.prisma.io/ (дата звернення: 13.05.2025).

5) Moodle. Learning Platform or Learning Management System (LMS). URL: https://moodle.org/?lang=uk (дата звернення: 03.04.2025).

6) PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/docs/ (дата звернення: 13.05.2025).

7) Postman: The World's Leading API Platform. URL: https://www.postman.com/product/api-client/ (дата звернення: 16.05.2025).

8) Visual Studio Code - Code Editing. Redefined. URL: https://code.visualstudio.com/ (дата звернення: 13.05.2025).

9) Docker: Accelerated Container Application Development. URL: https://www.docker.com/ (дата звернення: 14.05.2025).

10) Cross Site Scripting (XSS). URL: https://owasp.org/www-community/attacks/xss/ (дата звернення: 18.05.2025).

11) SQL Injection. URL: https://www.w3schools.com/sql/sql_injection.asp (дата звернення: 19.05.2025).

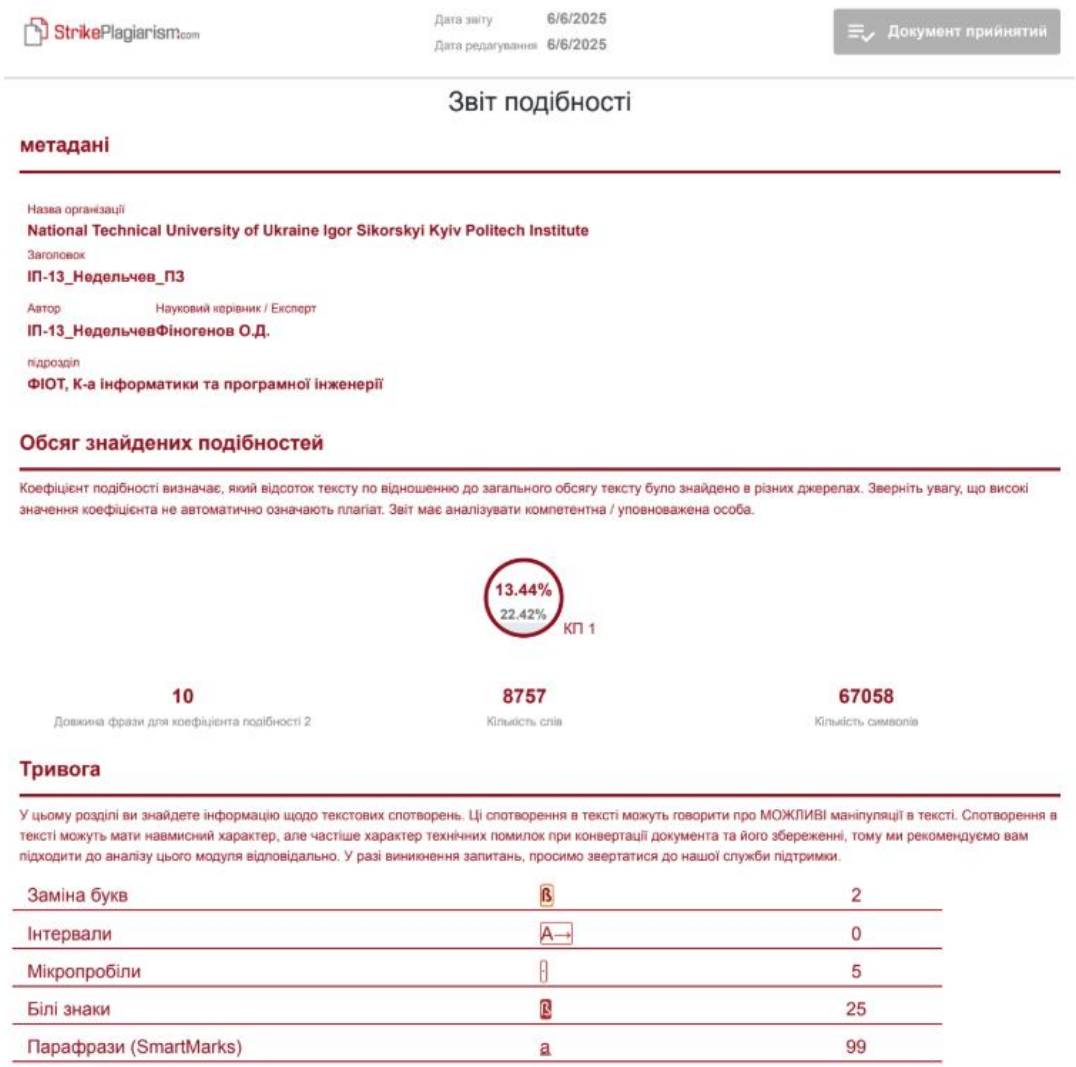
12) Understanding Lighthouse: a Comprehensive Tool for Web Performance Auditing. URL: https://medium.com/@MakeComputerScienceGreatAgain/understanding-lighthouse-a-comprehensive-tool-for-web-performance-auditing-97b0d5e67289 (дата звернення: 21.05.2025).

13) SonarQube Cloud Online Code Review as a Service Tool. URL: https://www.sonarsource.com/products/sonarcloud/ (дата звернення: 21.05.2025).

14) Swagger: API Documentation & Design Tools for Teams. URL: https://swagger.io/ (дата звернення: 13.05.2025).

ДОДАТКИ

ДОДАТОК А Звіт подібності



StrikePlagiarism.com

Дата звіту 6/6/2025
Дата редагування 6/6/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-13_Недельчев_ПЗ
Автор Науковий керівник / Експерт
ІП-13_НедельчевФіногенов О.Д.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

13.44%
22.42%
КП 1

10	8757	67058
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	2
Інтервали	A→	0
Мікропробіли	¶	5
Білі знаки	₢	25
Парафрази (SmartMarks)	a	99

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

WEB-застосунок адміністрування роботи відділу аспірантури

Текст програми

КПІ.ІП-1323.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олексій ФІНОГЕНОВ

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Євген НЕДЕЛЬЧЕВ

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
Фронтенд: https://github.com/RRedBaron/postgradhub-fe
Бекенд: https://github.com/RRedBaron/postgradhub-be

Файл auth.service.ts
Реалізація функціональної задачі FR-2

```
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { JwtService } from '@nestjs/jwt';
import * as bcrypt from 'bcryptjs';
import { JwtPayload } from './jwt-payload.interface';
import { PrismaService } from '../../../prisma/prisma.service';
import { Gender, Role } from '@prisma/client';


@Injectable()
export class AuthService {
  constructor(
    private prisma: PrismaService,
    private jwtService: JwtService,
    private config: ConfigService,
  ) {}

  async validateUser(email: string, pass: string) {
    const user = await this.prisma.user.findUnique({ where: { email } });
    if (!user) return null;
    const valid = await bcrypt.compare(pass, user.password);
    return valid ? user : null;
  }

  private getTokens(userId: string, email: string, role: Role) {
    const payload: JwtPayload = { sub: userId, email, role };
```

```
  const accessToken = this.jwtService.sign(payload);
  const refreshToken = this.jwtService.sign(payload, {
    secret: this.config.get<string>('JWT_REFRESH_SECRET'),
    expiresIn: this.config.get<string>('JWT_REFRESH_EXPIRES_IN'),
  });
  return { accessToken, refreshToken };
}

private async saveRefreshToken(userId: string, token: string) {
  const hash = await bcrypt.hash(token, 10);
  await this.prisma.user.update({
    where: { id: userId },
    data: { currentHashedRefreshToken: hash },
  });
}

async login(user: { id: string; email: string; role: Role }) {
  const tokens = this.getTokens(user.id, user.email, user.role);
  await this.saveRefreshToken(user.id, tokens.refreshToken);
  return {
    access_token: tokens.accessToken,
    refresh_token: tokens.refreshToken,
  };
}

async register(
  email: string,
  password: string,
  firstName: string,
  secondName: string,
```

```
  lastName: string,
  role: Role,
  gender: Gender,
  groupId?: string,
) {
  const hashedPassword = await bcrypt.hash(password, 10);
  const user = await this.prisma.user.create({
    data: {
      email,
      password: hashedPassword,
      firstName,
      lastName,
      secondName,
      gender,
      role: role || Role.PhD,
      groupId: groupId || undefined,
    },
  });
  return this.login(user);
}

async refreshTokens(userId: string, refreshToken: string) {
  const user = await this.prisma.user.findUnique({ where: { id: userId } });
  if (!user || !user.currentHashedRefreshToken) return null;
  const matches = await bcrypt.compare(
    refreshToken,
    user.currentHashedRefreshToken,
  );
  if (!matches) return null;
  const tokens = this.getTokens(user.id, user.email, user.role);
```

```
    await this.saveRefreshToken(user.id, tokens.refreshToken);
    return {
      access_token: tokens.accessToken,
      refresh_token: tokens.refreshToken,
    };
  }
}
```

Файл language-context.tsx
Реалізація функціональної задачі FR-7

```
"use client";

import {
  createContext,
  useContext,
  useState,
  ReactNode,
  useEffect,
} from "react";
import { useRouter, usePathname } from "next/navigation";
import { useLocale } from "next-intl";

type Language = "en" | "uk";

interface LanguageContextType {
  language: Language;
  setLanguage: (lang: Language) => void;
}

const LanguageContext = createContext<LanguageContextType | undefined>(
```

```
  undefined
);

export function LanguageProvider({ children }: { children: ReactNode }) {
  const [language, setLanguage] = useState<Language>("en");
  const router = useRouter();
  const pathname = usePathname();
  const locale = useLocale();

  useEffect(() => {
    setLanguage(locale as Language);
  }, [locale]);

  const handleLanguageChange = (newLang: Language) => {
    setLanguage(newLang);
    const newPath = pathname.replace(`/${locale}`, `/${newLang}`);
    router.push(newPath);
  };

  return (
    <LanguageContext.Provider
      value={{ language, setLanguage: handleLanguageChange }}
    >
      {children}
    </LanguageContext.Provider>
  );
}

export function useLanguage() {
  const context = useContext(LanguageContext);
```

```
  if (context === undefined) {
    throw new Error("useLanguage must be used within a LanguageProvider");
  }
  return context;
}
```

Файл language-context.tsx

Реалізація функціональної задачі FR-12

```
import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  UseGuards,
  Request,
} from '@nestjs/common';
import { DocumentsService } from './documents.service';
import { CreateDocumentDto, UpdateDocumentDto } from './dto/documents.dto';
import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';
import { RolesGuard } from '../auth/roles.guard';
import { Roles } from '../auth/decorators/roles.decorator';
import { Role } from '@prisma/client';

@Controller('documents')
@UseGuards(JwtAuthGuard, RolesGuard)
export class DocumentsController {
  constructor(private readonly documentsService: DocumentsService) {}

  @Post()
  @Roles(Role.SUPERVISOR)
  create(@Request() req, @Body() createDocumentDto: CreateDocumentDto) {
    return this.documentsService.create(req.user.id, createDocumentDto);
  }

  @Get()
  findAll(@Request() req) {
    return this.documentsService.findAll(req.user.id);
  }
```

```
  @Get('student/:studentId')
  @Roles(Role.SUPERVISOR)
  getStudentDocuments(@Request() req, @Param('studentId') studentId: string) {
    return this.documentsService.getStudentDocuments(req.user.id, studentId);
  }

  @Get(':id')
  findOne(@Request() req, @Param('id') id: string) {
    return this.documentsService.findOne(req.user.id, id);
  }

  @Patch(':id')
  update(
    @Request() req,
    @Param('id') id: string,
    @Body() updateDocumentDto: UpdateDocumentDto,
  ) {
    return this.documentsService.update(req.user.id, id, updateDocumentDto);
  }

  @Delete(':id')
  @Roles(Role.SUPERVISOR)
  remove(@Request() req, @Param('id') id: string) {
    return this.documentsService.remove(req.user.id, id);
  }

  @Post('initialize')
  initializeDocuments(@Request() req) {
    return this.documentsService.initializeUserDocuments(req.user.id);
  }
}
```

Файл analytics.service.ts

Реалізація функціональної задачі FR-13

```
import { Injectable, ForbiddenException } from '@nestjs/common';
import { PrismaService } from 'prisma/prisma.service';
import { Role, TaskStatus } from '@prisma/client';
import {
  TaskStatsDto,
  SupervisorStatsDto,
  YearStatsDto,
} from './dto/analytics.dto';

@Injectable()
export class AnalyticsService {
```

```
constructor(private prisma: PrismaService) {}

async getTaskStats(userId: string): Promise<TaskStatsDto> {
  const user = await this.prisma.user.findUnique({
    where: { id: userId },
    select: { role: true },
  });

  let whereClause = {};
  if (user.role === Role.SUPERVISOR) {
    whereClause = {
      createdById: userId,
    };
  }

  const tasks = await this.prisma.task.findMany({
    where: whereClause,
    select: {
      deadline: true,
      status: true,
    },
  });

  const totalTasks = tasks.length;
  if (totalTasks === 0) {
    return { onTime: 0, late: 0 };
  }

  const onTimeTasks = tasks.filter(
    (task) =>
      task.status === TaskStatus.COMPLETED && task.deadline >= new Date(),
  ).length;

  return {
    onTime: Math.round((onTimeTasks / totalTasks) * 100),
    late: Math.round(((totalTasks - onTimeTasks) / totalTasks) * 100),
  };
}

async getSupervisorStats(userId: string): Promise<SupervisorStatsDto[]> {
  const user = await this.prisma.user.findUnique({
    where: { id: userId },
    select: { role: true },
  });
```

```
    if (user.role !== Role.HEAD) {
      throw new ForbiddenException('Only head can view supervisor statistics');
    }

    const supervisors = await this.prisma.user.findMany({
      where: { role: Role.SUPERVISOR },
      select: {
        id: true,
        firstName: true,
        lastName: true,
        tasks: {
          select: {
            deadline: true,
            status: true,
          },
        },
      },
    });

    return supervisors.map((supervisor) => {
      const totalTasks = supervisor.tasks.length;
      const onTimeTasks = supervisor.tasks.filter(
        (task) =>
          task.status === TaskStatus.COMPLETED && task.deadline >= new Date(),
      ).length;

      return {
                                                    name:              `${supervisor.lastName}
${supervisor.firstName[0]}.${supervisor.firstName[0]}.`,
        onTime:
          totalTasks === 0 ? 0 : Math.round((onTimeTasks / totalTasks) * 100),
        total: totalTasks,
      };
    });
  }

  async getYearStats(userId: string): Promise<YearStatsDto[]> {
    const user = await this.prisma.user.findUnique({
      where: { id: userId },
      select: { role: true },
    });

    let whereClause = {};
    if (user.role === Role.SUPERVISOR) {
      whereClause = {
```

```
      createdById: userId,
    };
  }

  const tasks = await this.prisma.task.findMany({
    where: whereClause,
    select: {
      deadline: true,
      status: true,
    },
  });

  const yearStats = new Map<
    number,
    { onTime: number; late: number; total: number }
  >();

  tasks.forEach((task) => {
    const year = task.deadline.getFullYear();
    const isOnTime =
      task.status === TaskStatus.COMPLETED && task.deadline >= new Date();

    if (!yearStats.has(year)) {
      yearStats.set(year, { onTime: 0, late: 0, total: 0 });
    }

    const stats = yearStats.get(year)!;
    stats.total++;
    if (isOnTime) {
      stats.onTime++;
    } else {
      stats.late++;
    }
  });

  return Array.from(yearStats.entries())
    .map(([year, stats]) => ({
      year,
      onTime: Math.round((stats.onTime / stats.total) * 100),
      late: Math.round((stats.late / stats.total) * 100),
    }))
    .sort((a, b) => a.year - b.year);
 }
}
```

Файл users.service.ts

Реалізація функціональної задачі FR-14

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { PrismaService } from 'prisma/prisma.service';

@Injectable()
export class UsersService {
  constructor(private prisma: PrismaService) {}

  async findOne(id: string) {
    const user = await this.prisma.user.findFirst({
      where: { id, isDeleted: false },
    });
    if (!user) throw new NotFoundException('User not found');
    return user;
  }

  async findAll() {
    return this.prisma.user.findMany({
      where: { isDeleted: false },
    });
  }
  async updateUser(id: string, data: any) {
    return this.prisma.user.update({
      where: { id },
      data,
    });
  }

  async softDeleteUser(id: string) {
    await this.findOne(id);
    return this.prisma.user.update({
      where: { id },
      data: { isDeleted: true, deletedAt: new Date() },
    });
  }
}
```

Файл tasks.service.ts

Реалізація функціональної задачі FR-10

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { PrismaService } from 'prisma/prisma.service';
import { CreateTaskDto, UpdateTaskDto, TaskStatus } from './dto/tasks.dto';

@Injectable()
```

```
export class TasksService {
  constructor(private prisma: PrismaService) {}

  async create(userId: string, createTaskDto: CreateTaskDto) {
    return this.prisma.task.create({
      data: {
        ...createTaskDto,
        status: TaskStatus.PENDING,
        userId,
        createdById: userId,
      },
    });
  }

  async findAll(userId: string) {
    return this.prisma.task.findMany({
      where: { userId },
      orderBy: { deadline: 'asc' },
    });
  }

  async findOne(userId: string, id: string) {
    const task = await this.prisma.task.findFirst({
      where: { id, userId },
    });

    if (!task) {
      throw new NotFoundException(`Task with ID ${id} not found`);
    }

    return task;
  }

  async update(userId: string, id: string, updateTaskDto: UpdateTaskDto) {
    await this.findOne(userId, id);

    return this.prisma.task.update({
      where: { id },
      data: updateTaskDto,
    });
  }

  async remove(userId: string, id: string) {
    await this.findOne(userId, id);
```

```
    return this.prisma.task.delete({
      where: { id },
    });
  }
}
```

Файл profile.tsx
Реалізація функціональної задачі FR-16

```
"use client";

import { useProfile, useUpdateProfile } from "@/lib/hooks/useProfile";
import { Button, Input, Select, SelectItem, Avatar, Chip } from "@heroui/react";
import { useState, useEffect } from "react";
import { useTranslations } from "next-intl";
import { Gender, Role } from "@/types/default";
import { PatternFormat } from "react-number-format";
import { addToast } from "@heroui/toast";

const genderOptions = [
  { key: Gender.MALE, label: "Male" },
  { key: Gender.FEMALE, label: "Female" },
  { key: Gender.OTHER, label: "Other" },
];

const groupOptions = [
  { key: "group1", label: "Group 1" },
  { key: "group2", label: "Group 2" },
  { key: "group3", label: "Group 3" },
];

const getRoleColor = (role: Role) => {
  switch (role) {
    case Role.PhD:
      return "primary";
    case Role.SUPERVISOR:
      return "success";
    case Role.HEAD:
      return "warning";
    default:
      return "default";
  }
};

const ProfileSkeleton = () => {
  return (
```

```
   <section className="flex flex-col gap-6 py-8 md:py-10 max-w-4xl mx-auto">
     <div className="flex flex-col md:flex-row items-center gap-8 p-8 rounded-lg
border border-gray-200 dark:border-gray-800">
          <div className="w-24 h-24 rounded-full bg-gray-200 dark:bg-gray-700
animate-pulse" />
       <div className="flex-1 flex flex-col gap-2">
         <div className="flex items-center gap-2">
          <div className="h-6 w-48 bg-gray-200 dark:bg-gray-700 rounded animate-
pulse" />
          <div className="h-6 w-20 bg-gray-200 dark:bg-gray-700 rounded animate-
pulse" />
         </div>
         <div className="h-4 w-32 bg-gray-200 dark:bg-gray-700 rounded animate-
pulse" />
       </div>
       <div className="h-10 w-24 bg-gray-200 dark:bg-gray-700 rounded animate-
pulse" />
     </div>

     <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
       {[...Array(6)].map((_, index) => (
         <div key={index} className="flex flex-col gap-2">
          <div className="h-4 w-20 bg-gray-200 dark:bg-gray-700 rounded animate-
pulse" />
              <div className="h-10 w-full bg-gray-200 dark:bg-gray-700 rounded
animate-pulse" />
         </div>
       ))}
     </div>
   </section>
  );
};

export default function Profile() {
  const { data: profile, isLoading } = useProfile();
  const { mutate: updateProfile, isPending } = useUpdateProfile();
  const t = useTranslations("profile");
  const [editMode, setEditMode] = useState(false);
  const [formData, setFormData] = useState({
    firstName: "",
    lastName: "",
    secondName: "",
    email: "",
    phone: "",
    gender: Gender.OTHER,
```

```
  group: "",
});

useEffect(() => {
  if (profile) {
    setFormData({
      firstName: profile.firstName || "",
      lastName: profile.lastName || "",
      secondName: profile.secondName || "",
      email: profile.email || "",
      phone: profile.phone || "",
      gender: profile.gender || Gender.OTHER,
      group: profile.group || "",
    });
  }
}, [profile]);

const handleInputChange = (field: string, value: string) => {
  setFormData((prev) => ({
    ...prev,
    [field]: value,
  }));
};

const handleSave = () => {
  if (!profile) return;

  updateProfile(
    {
      ...profile,
      ...formData,
    },
    {
      onSuccess: () => {
        setEditMode(false);
        addToast({
          title: "Success",
          description: "Profile updated successfully",
          color: "success",
        });
      },
      onError: (error) => {
        addToast({
          title: "Error",
          description: "Failed to update profile",
```

```
          color: "danger",
        });
      },
    }
  );
};

if (isLoading || !profile) {
  return <ProfileSkeleton />;
}

return (
  <section className="flex flex-col gap-6 py-8 md:py-10 max-w-4xl mx-auto">
    <div className="flex flex-col md:flex-row items-center gap-8 p-8 rounded-lg
border border-gray-200 dark:border-gray-800">
      <Avatar
        size="lg"
        src={"https://i.pravatar.cc/150?u=a042581f4e29026704d"}
      />
      <div className="flex-1 flex flex-col gap-1">
        <div className="flex items-center gap-2">
          <span className="text-xl font-bold">
            {formData.firstName} {formData.lastName}
          </span>
          {profile.role && (
            <Chip color={getRoleColor(profile.role)} variant="flat">
              {profile.role}
            </Chip>
          )}
        </div>
        <span className="text-md text-gray-500 dark:text-gray-400">
          {formData.email}
        </span>
      </div>
      <Button
        color="primary"
        onPress={() => (editMode ? handleSave() : setEditMode(true))}
        isLoading={isPending}
      >
        {editMode ? t("save") : t("edit")}
      </Button>
    </div>

    <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
      <Input
```

```
  label={t("fullName")}
  placeholder="Your First Name"
  value={formData.firstName}
  onChange={(e) => handleInputChange("firstName", e.target.value)}
  isReadOnly={!editMode}
/>
<Input
  label={t("lastName")}
  placeholder="Your Last Name"
  value={formData.lastName}
  onChange={(e) => handleInputChange("lastName", e.target.value)}
  isReadOnly={!editMode}
/>
<Input
  label={t("secondName")}
  placeholder="Your Second Name"
  value={formData.secondName}
  onChange={(e) => handleInputChange("secondName", e.target.value)}
  isReadOnly={!editMode}
/>
<Input
  label={t("email")}
  placeholder="Your Email"
  value={formData.email}
  onChange={(e) => handleInputChange("email", e.target.value)}
  isReadOnly={!editMode}
/>
<div>
  <label className="block text-sm font-medium mb-1">{t("phone")}</label>
  <PatternFormat
    format="+38 (###) ###-##-##"
    mask="_"
    customInput={Input}
    value={formData.phone}
    onValueChange={(values) => {
      handleInputChange("phone", values.value);
    }}
    placeholder="+38 (000) 000-00-00"
    isReadOnly={!editMode}
    allowEmptyFormatting={editMode}
  />
</div>

<Select
  label={t("gender")}
```

```
          selectedKeys={formData.gender ? [formData.gender] : []}
          onChange={(e) => handleInputChange("gender", e.target.value)}
          isDisabled={!editMode}
        >
          {genderOptions.map((opt) => (
            <SelectItem key={opt.key}>{opt.label}</SelectItem>
          ))}
        </Select>

        <Select
          label={t("group")}
          selectedKeys={formData.group ? [formData.group] : []}
          onChange={(e) => handleInputChange("group", e.target.value)}
          isDisabled={!editMode}
        >
          {groupOptions.map((opt) => (
            <SelectItem key={opt.key}>{opt.label}</SelectItem>
          ))}
        </Select>
      </div>
    </section>
  );
}
```

Файл StudentsList.tsx

Реалізація функціональної задачі FR-14

```
"use client";

import {
  Table,
  TableHeader,
  TableBody,
  TableColumn,
  TableRow,
  TableCell,
  Chip,
  Avatar,
  Progress,
  Button,
} from "@heroui/react";
import { useTranslations } from "next-intl";

const students = [
  {
    id: 1,
```

```
    name: "John Doe",
    email: "john.doe@example.com",
    avatar: "https://i.pravatar.cc/150?img=1",
    progress: 75,
    status: "active",
    lastActivity: "2024-03-15",
  },
  {
    id: 2,
    name: "Jane Smith",
    email: "jane.smith@example.com",
    avatar: "https://i.pravatar.cc/150?img=2",
    progress: 45,
    status: "active",
    lastActivity: "2024-03-14",
  },
  {
    id: 3,
    name: "Mike Johnson",
    email: "mike.johnson@example.com",
    avatar: "https://i.pravatar.cc/150?img=3",
    progress: 90,
    status: "active",
    lastActivity: "2024-03-15",
  },
];

const getStatusColor = (status: string) => {
  switch (status) {
    case "active":
      return "success";
    case "inactive":
      return "warning";
    default:
      return "default";
  }
};

export const StudentsList = () => {
  const t = useTranslations("supervisor");

  return (
    <div className="space-y-4">
      <div className="flex justify-between items-center">
        <h2 className="text-xl font-semibold">{t("studentsList")}</h2>
```

```
  <Button color="primary" variant="flat">
    {t("addStudent")}
  </Button>
</div>

<Table aria-label="Students list">
  <TableHeader>
    <TableColumn>{t("student")}</TableColumn>
    <TableColumn>{t("progress")}</TableColumn>
    <TableColumn>{t("status")}</TableColumn>
    <TableColumn>{t("lastActivity")}</TableColumn>
    <TableColumn>{t("actions")}</TableColumn>
  </TableHeader>
  <TableBody>
    {students.map((student) => (
      <TableRow key={student.id}>
        <TableCell>
          <div className="flex items-center gap-3">
            <Avatar src={student.avatar} size="sm" />
            <div className="flex flex-col">
              <span className="font-medium">{student.name}</span>
              <span className="text-small text-default-500">
                {student.email}
              </span>
            </div>
          </div>
        </TableCell>
        <TableCell>
          <div className="flex flex-col gap-1">
            <Progress
              value={student.progress}
              color={student.progress >= 70 ? "success" : "warning"}
              className="max-w-md"
            />
            <span className="text-small text-default-500">
              {student.progress}%
            </span>
          </div>
        </TableCell>
        <TableCell>
          <Chip
            color={getStatusColor(student.status)}
            variant="flat"
            size="sm"
          >
```

```
                {t(`status.${student.status}`)}
              </Chip>
            </TableCell>
            <TableCell>
              <span className="text-small text-default-500">
                {new Date(student.lastActivity).toLocaleDateString()}
              </span>
            </TableCell>
            <TableCell>
              <div className="flex gap-2">
                <Button size="sm" variant="light">
                  {t("viewDetails")}
                </Button>
                <Button size="sm" color="primary" variant="flat">
                  {t("message")}
                </Button>
              </div>
            </TableCell>
          </TableRow>
        ))}
      </TableBody>
    </Table>
  </div>
 );
};
```

Файл page.tsx
Реалізація функціональної задачі FR-4

```
"use client";

import { ROUTES } from "@/common/enums/routes";
import { emailRegex } from "@/constants";
import { Link } from "@/i18n/navigation";
import { useLogin } from "@/lib/hooks";
import { Button } from "@heroui/button";
import { Input } from "@heroui/input";
import { addToast } from "@heroui/toast";
import { useRouter } from "next/navigation";
import { useForm } from "react-hook-form";
import { useTranslations } from "next-intl";

interface LoginForm {
  email: string;
  password: string;
```

```
}

export default function Login() {
  const { mutate: loginUser, isPending, error } = useLogin();
  const router = useRouter();
  const t = useTranslations("auth");

  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm<LoginForm>({
    mode: "onChange",
    defaultValues: {
      email: "",
      password: "",
    },
  });

  const onSubmit = (data: LoginForm) => {
    loginUser(data, {
      onSuccess: () => {
        addToast({
          title: t("welcome"),
          description: t("loggedIn"),
          color: "success",
        });
        router.push(ROUTES.HOME);
      },
    });
  };

  return (
    <section className="flex flex-col w-full items-center justify-center py-8">
      <div className="w-full max-w-[480px] rounded-lg p-8">
        <div className="text-center mb-8">
          <h1 className="text-3xl font-bold text-gray-900 dark:text-white">
            PostgradHub
          </h1>
          <p className="text-gray-600 dark:text-gray-400 mt-2">Welcome back</p>
        </div>

        <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">
          <div className="flex flex-row justify-between w-full items-center">
            <h2 className="text-xl font-semibold text-gray-900 dark:text-white">
```

```
          {t("login")}
        </h2>
        <Link
          className="text-blue-600 hover:text-blue-700 transition-colors"
          href="/auth/signup"
        >
          {t("signUp")}
        </Link>
      </div>

      <Input
        isRequired
        label={t("email")}
        {...register("email", {
          required: "Email is required",
          pattern: {
            value: emailRegex,
            message: "Invalid email address",
          },
        })}
        placeholder="example_email@gmail.com"
        isInvalid={!!errors.email}
        errorMessage={errors.email?.message}
      />

      <Input
        isRequired
        type="password"
        label={t("password")}
        {...register("password", {
          required: "Password is required",
        })}
        isInvalid={!!errors.password}
        errorMessage={errors.password?.message}
      />

      <Button
        type="submit"
        color="primary"
        variant="flat"
          className="w-full h-12 text-lg font-medium bg-blue-600 hover:bg-blue-
700 text-white transition-colors"
        disabled={isPending}
      >
        {isPending ? t("signingIn") : t("signIn")}
```

```
          </Button>

          {error && (
            <p className="text-red-500 text-sm text-center">
              {t("loginFailed")}: {error.message}
            </p>
          )}
        </form>
      </div>
    </section>
  );
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

WEB-застосунок адміністрування роботи відділу аспірантури

Програма та методика тестування

КПІ.ІП-1323.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олексій ФІНОГЕНОВ

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Євген НЕДЕЛЬЧЕВ

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробувань є веб-застосунок PostgradHub – система адміністрування роботи відділу аспірантури, що автоматизує робочі процеси закладів вищої освіти у підготовці PhD.

В якості технологічної платформи додатку були обрані наступні технології:

– серверна частина: NestJS, Prisma ORM, REST API, Swagger;
– субд: PostgreSQL 15;
– клієнтська частина: NextJS (React, TypeScript)

Додаток має коректно працювати в усіх сучасних браузерах, а саме в:

– Google Chrome (остання стабільна версія);
– Mozilla Firefox (остання стабільна версія);
– Safari (остання стабільна версія);
– Microsoft Edge (остання стабільна версія).

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка збереження даних;
- перевірка сумісності вебзастосунка з останніми версіями сучасних браузерів (Chrome, Opera, Firefox);
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

- Інтегроване середовище розробки (IDE): Visual Studio Code;
- Система управління базами даних: PostgreSQL;
- Система контейнеризації: Docker:
- SonarQube для статичного аналізу коду;
- Веббраузери Google Chrome, Mozilla Firefox, Safari, Microsoft Edge (останні стабільні версії) для мануального тестування користувацького інтерфейсу.

Порядок проведення тестування буде наступним:

– підготовка середовища: клонування репозиторію, створення локальних змінних середовища .env, запуск контейнерів за допомогою docker compose up;

– статичний аналіз коду: запуск SonarQube сканування, фіксування метрик;

– автоматизовані модульні та інтеграційні тести: запуск модульних та інтеграційних тестів;

– функціональне мануальне тестування UI: тестування типових сценаріїв (логін, створення плану, адміністрування, подання звітів) у Chrome, Firefox, Edge, Safari, перевірка адаптивності на мобільній ширині 360 на 640 пікселів через інструменти DevTools;

– перевірка API-контрактів: тестування працездатності серверної частини додатку за допомогою запитів, здійснених через Postman;

– запускаємо Lighthouse-аудиту (у Chrome) і збір результатів.

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2025 р.

WEB-застосунок адміністрування роботи відділу аспірантури

Керівництво користувача

КПІ.ІП-1323.045440.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олексій ФІНОГЕНОВ

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Євген НЕДЕЛЬЧЕВ

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«PostgradHub» – це вебзастосунок, який автоматизує рутинні адміністративні процеси відділу аспірантури: тут можна реєструвати користувачів із ролями «аспірант», «керівник» та «адміністратор», створювати й підтримувати індивідуальні наукові плани, подавати та узгоджувати звіти, бронювати дати консультацій і захистів дисертацій, ставити завдання з дедлайнами й оперативно перевіряти їх виконання. Завдяки гнучкій системі ролей із точним розмежуванням прав доступу, адаптивному багатомовному інтерфейсу та інструментам для формування звітів і аналітики «PostgradHub» підвищує швидкість, достовірність і прозорість обробки освітньої інформації.

Основні можливості додатку:

– мультиролевий інтерфейс (аспірант / керівник / адміністратор);
– ведення профілів із науковими планами та прогресом;
– бронювання дат консультацій і захистів;
– створення завдань із дедлайнами;
– перевірка та оцінювання завдань;
– керування планом дисертації;
– звіти та аналітика.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність веб-браузеру Edge або Chrome з версіями 90.0 або вище, Firefox з версією 78.0 та вище, Safari з версією 14.0 або вище;

– наявність доступу до Інтернету.

2.2 Завантаження застосунку

Програмне забезпечення є веб-застосунком та не потребує завантаження з боку користувача.

2.3 Перевірка коректної роботи

Після запуска Docker контейнерів користувачу необхідно ввести локальну адресу застосунку в браузері, після чого він опиниться на головній сторінці додатку в якості гостя.

3 ВИКОНАННЯ ПРОГРАМИ

Після введення коректної локальної адреси застосунку в браузері користувач потрапляє на головну сторінку додатку (рис. 3.1).

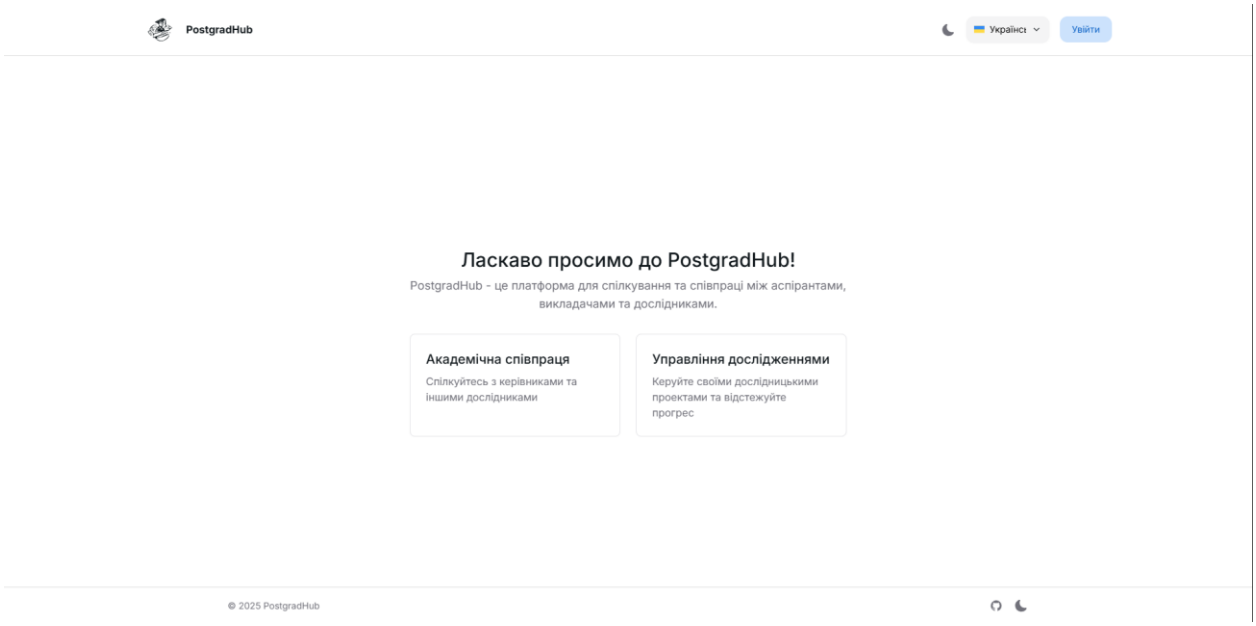


Рисунок 3.1 – Головна сторінка застосунку

Для того, щоб зареєструватись в системі користувач натискає на кнопку «Увійти» і потрапляє на сторінку авторизації (рис. 3.2).

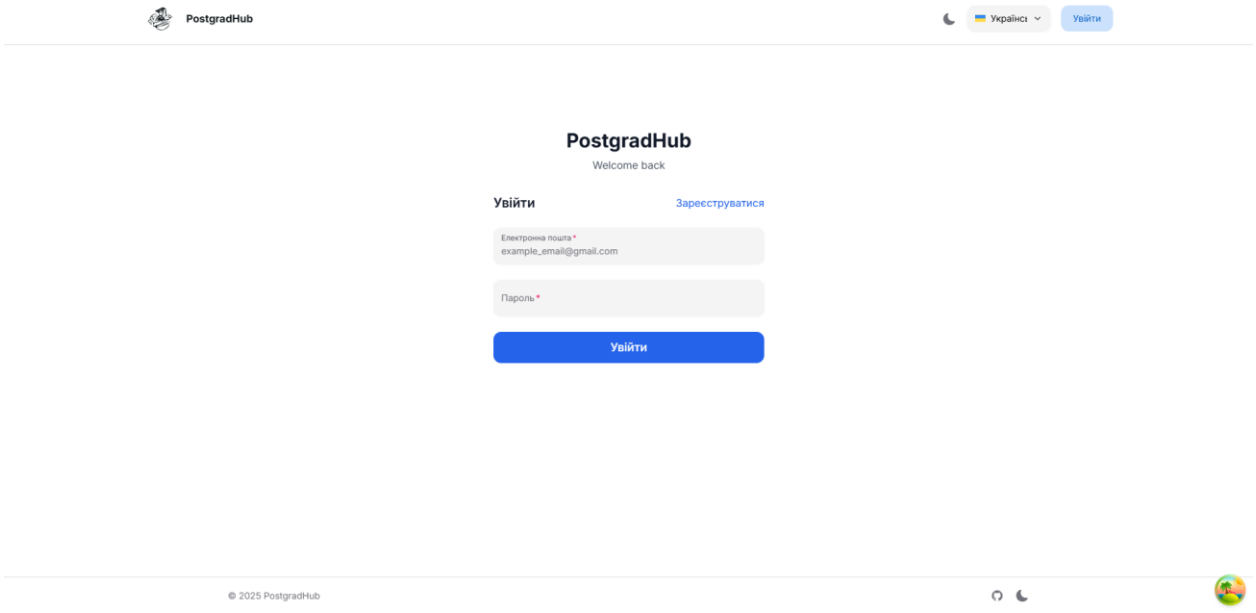


Рисунок 3.2 – Сторінка авторизації

Для того, щоб перейти на сторінку реєстрації (рис. 3.3), користувач має натиснути на гіперпосилання з текстом «Зареєструватися». На сторінці реєстрації користувач вводить свої ім’я, прізвище, номер телефону, контактну електронну пошту та інші дані і натискає на кнопку «Створити обліковий запис».

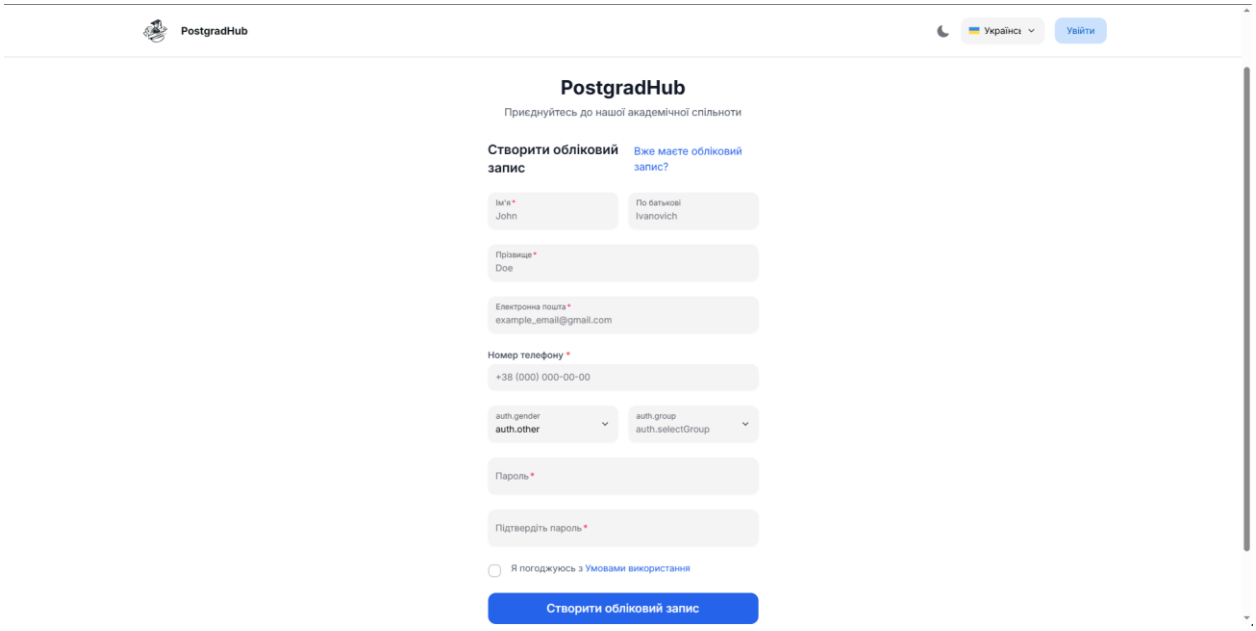


Рисунок 3.3 – Сторінка реєстрації

Після успішної реєстрації користувач знову опиняється на головній сторінці додатку, проте тепер він має доступ до свого профілю, сторінки перегляду завдань та сторінки управління дисертаційною роботою. Для того, щоб перейти в профіль, користувач натискає на зображення свого профілю та обирає пункт «Профіль» (рис. 3.4).

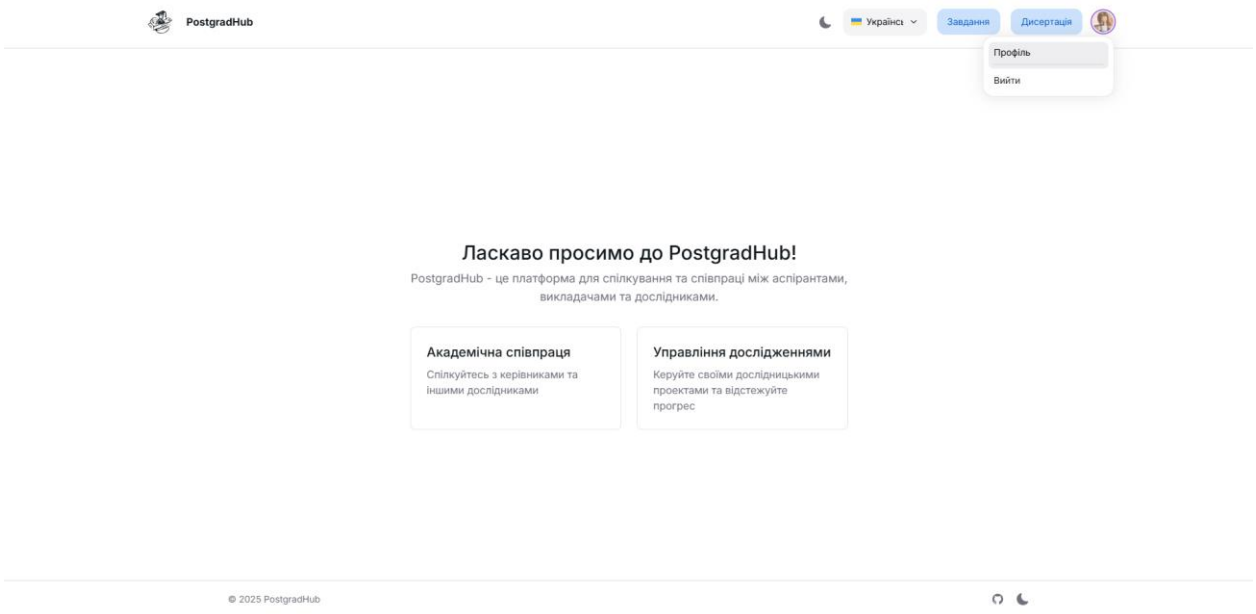


Рисунок 3.4 – Перехід з головної сторінки до профілю

Після цього користувач переходить на сторінку, де може переглянути та відредагувати свій профіль (рис. 3.5).

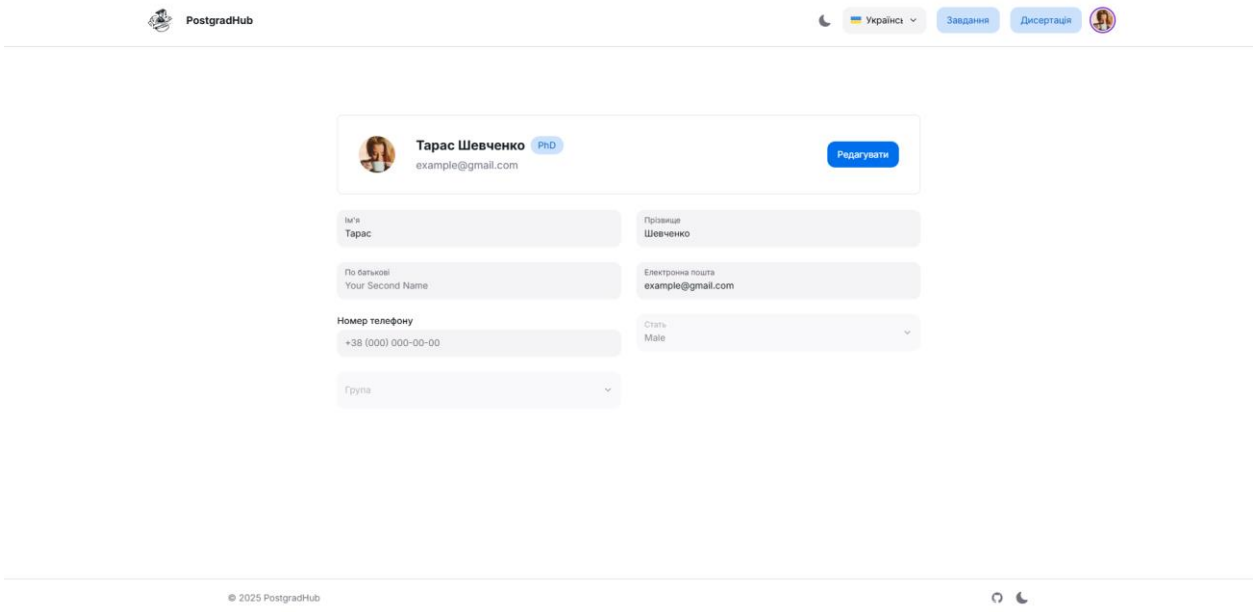


Рисунок 3.5 – Сторінка профілю

Для того, щоб переглянути актуальний список завдань, користувач має натиснути на кнопку «Завдання» у верхній навігаційній панелі додатку. Після цього він опиниться на сторінці перегляду списку завдань (рис. 3.6).

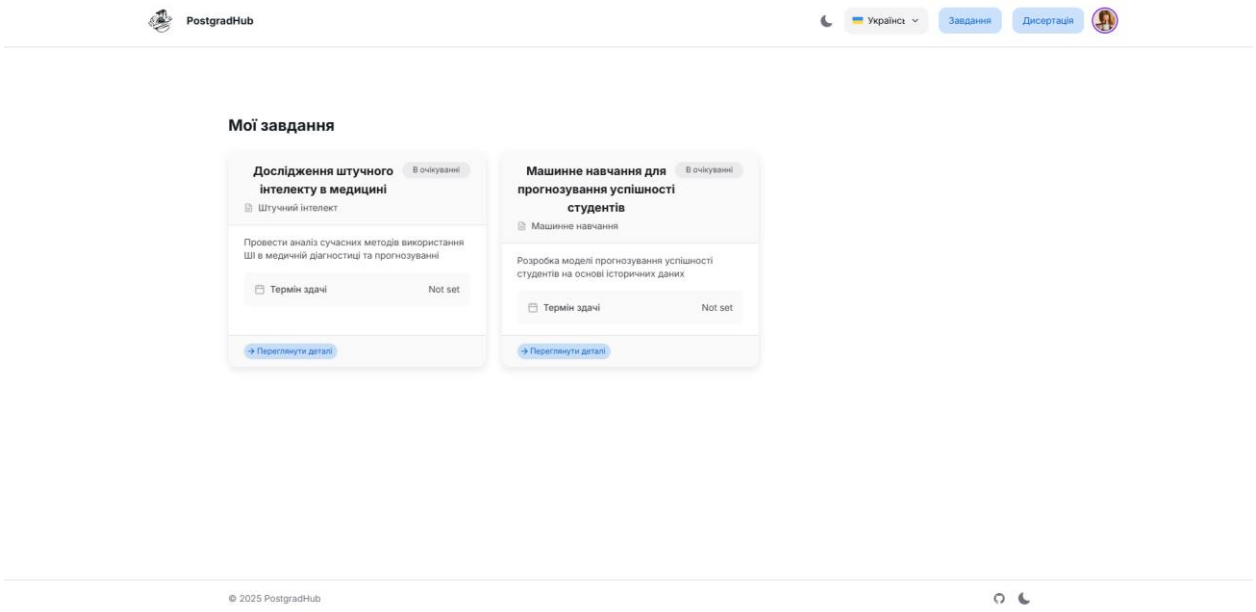


Рисунок 3.6 – Сторінка списку завдань

Для детального перегляду завдання та подання його на перевірку користувач обирає завдання зі списку та переходить на сторінку детального опису завдання (рис. 3.7).

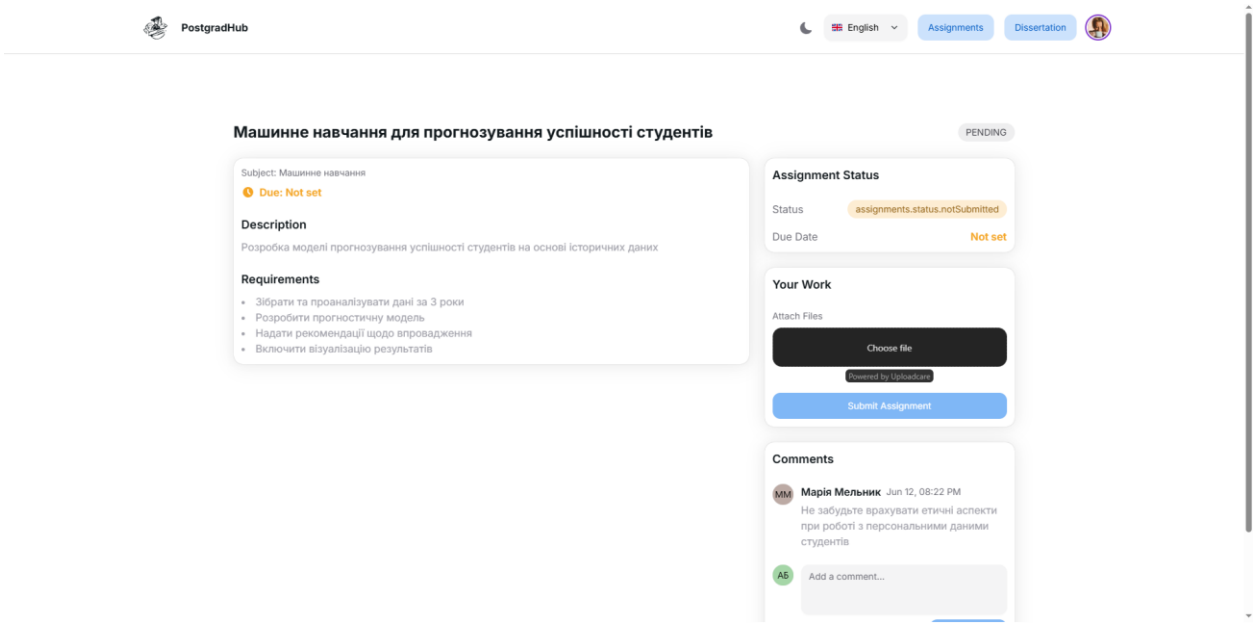


Рисунок 3.7 – Сторінка детального перегляду завдання

Щоб подати завдання на перевірку, користувач натискає на кнопку «Choose file». Після цього користувач має обрати файл на своєму комп’ютері, або завантажити його, перетягнувши в модальне вікно (рис. 3.8).

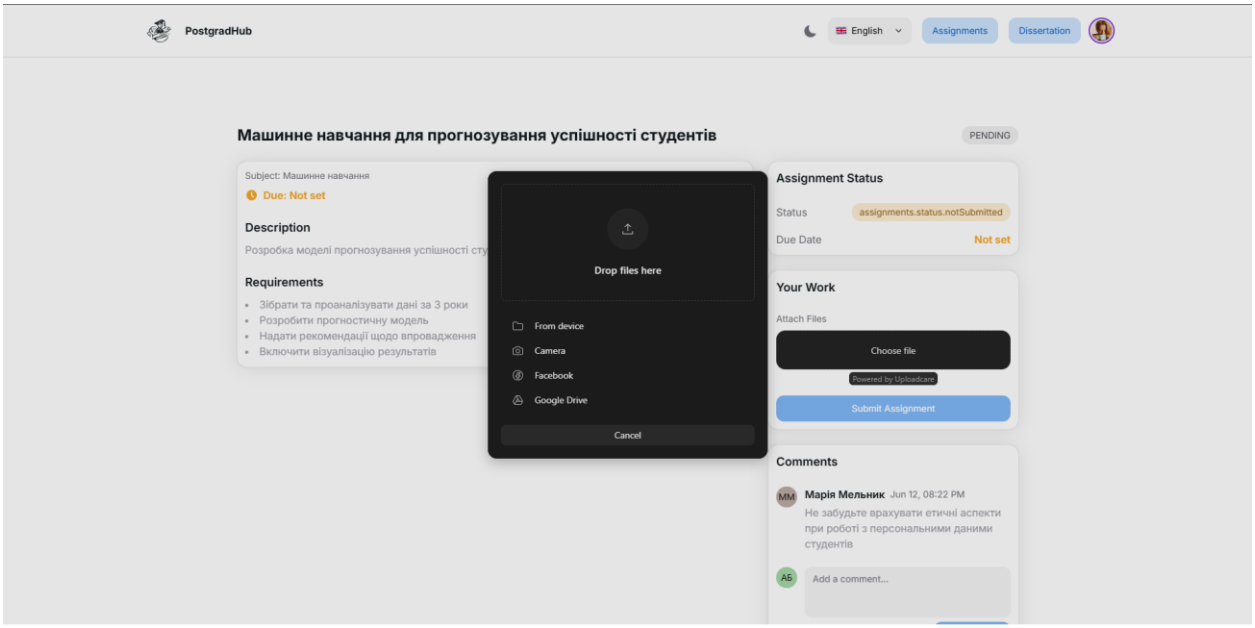


Рисунок 3.8 – Модальне вікно завантаження файлу

Після завантаження бажаного файлу користувач натискає на кнопку «Submit assignment» (рис. 3.9).

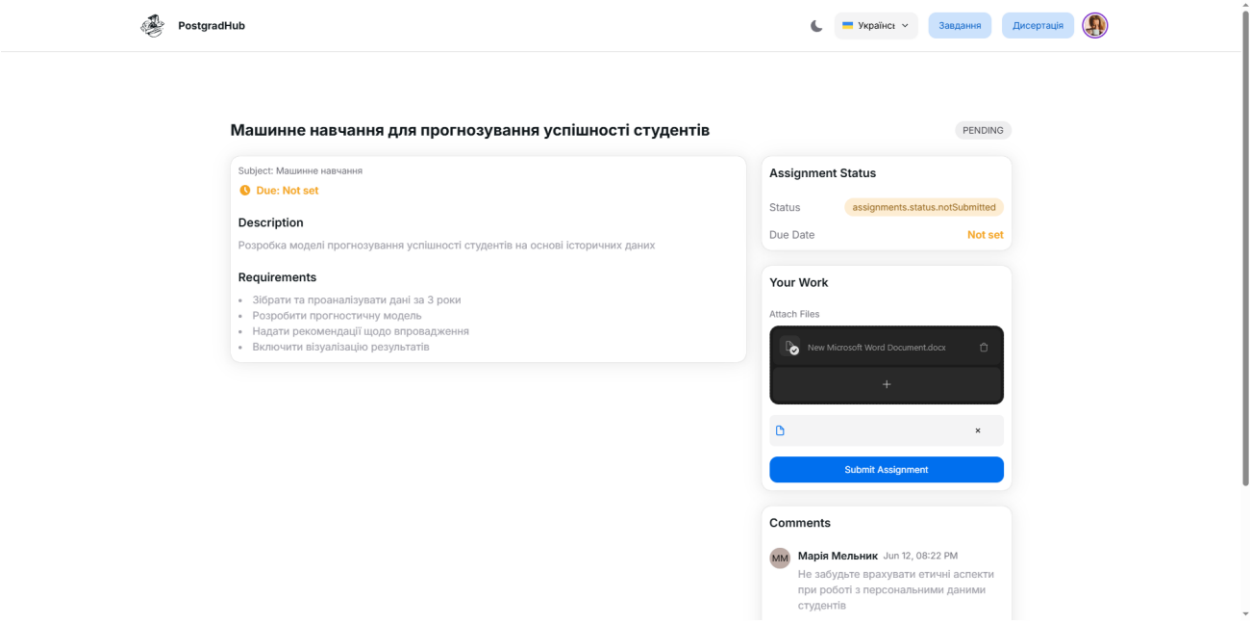


Рисунок 3.9 – Файл завантажено, завдання здано на перевірку

Для того, щоб перейти на сторінку управління дисертаційним проектом, користувач має натиснути на кнопку «Дисертація». Після чого він потрапить на сторінку з поданням теми дисертації на затвердження (рис. 3.10).

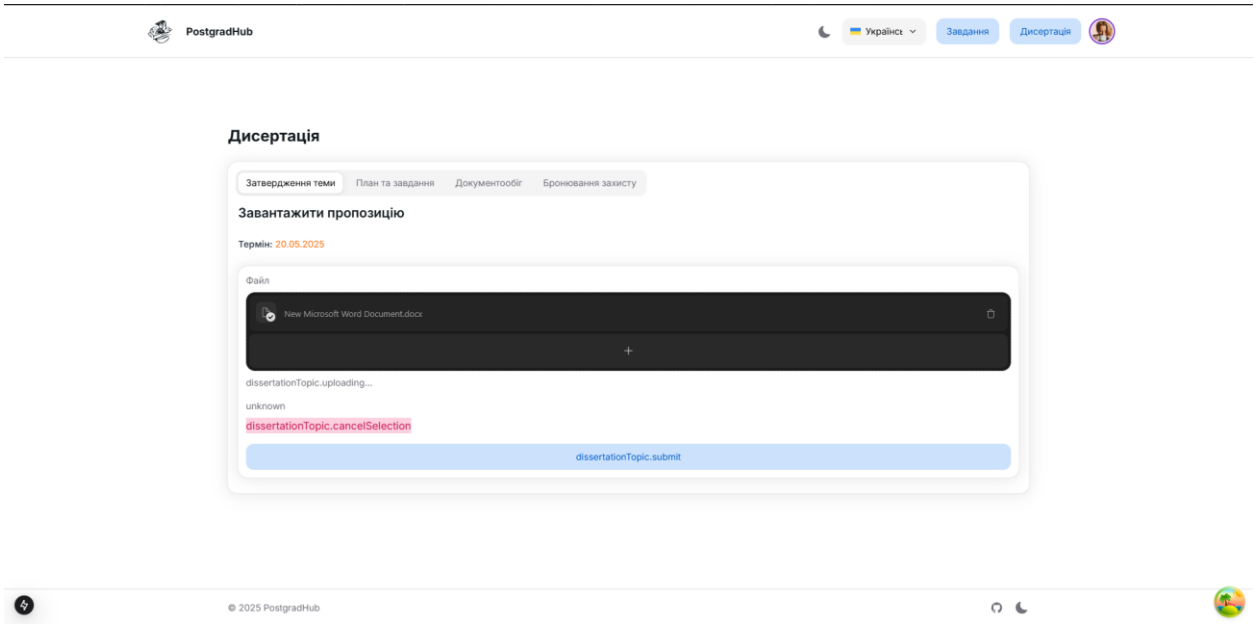


Рисунок 3.10 – Сторінка завантаження файлу з темою роботи

Для того, щоб переглянути власні задачі, або створити нові, користувач має перейти на вкладку «План та завдання» (рис. 3.11).

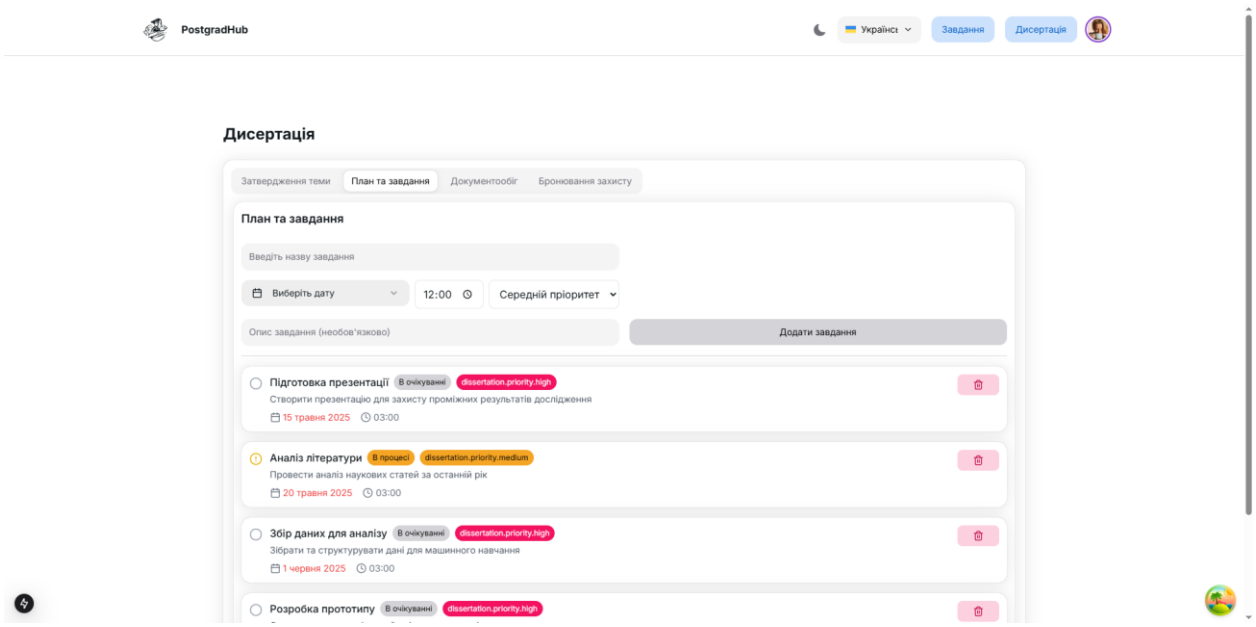


Рисунок 3.11 – Сторінка створення та перегляду власних завдань

Для того, щоб завантажити документи, необхідні для написання дисертації, користувач має перейти на вкладку «Документообіг» (рис. 3.12), де він побачить список необхідних файлів та зможе їх завантажити аналогічно до того, як це відбувається із завантаження завдань на перевірку.

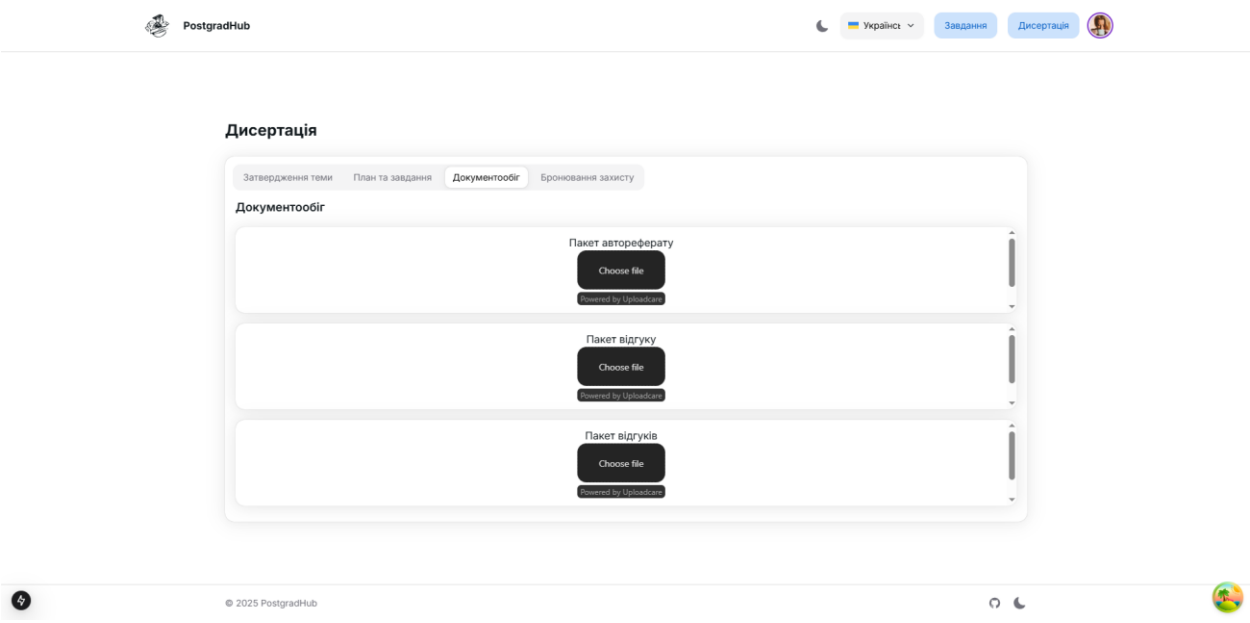


Рисунок 3.12 – Сторінка завантаження документів

Для того, щоб забронювати час консультації або захисту дисертації, користувач переходить на вкладку «Бронювання захисту» (рис. 3.13).

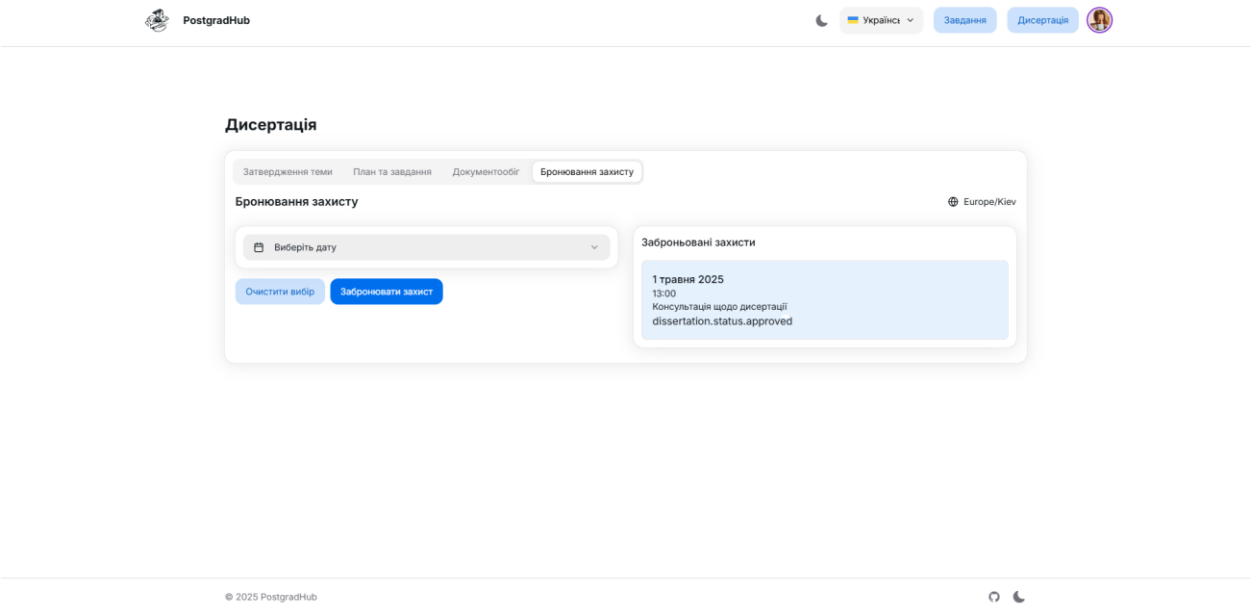


Рисунок 3.13 – Сторінка бронювання дати захисту роботи

Для власне бронювання користувач обирає бажані дату та час і натискає на кнопку «Забронювати захист» (рис. 3.14).

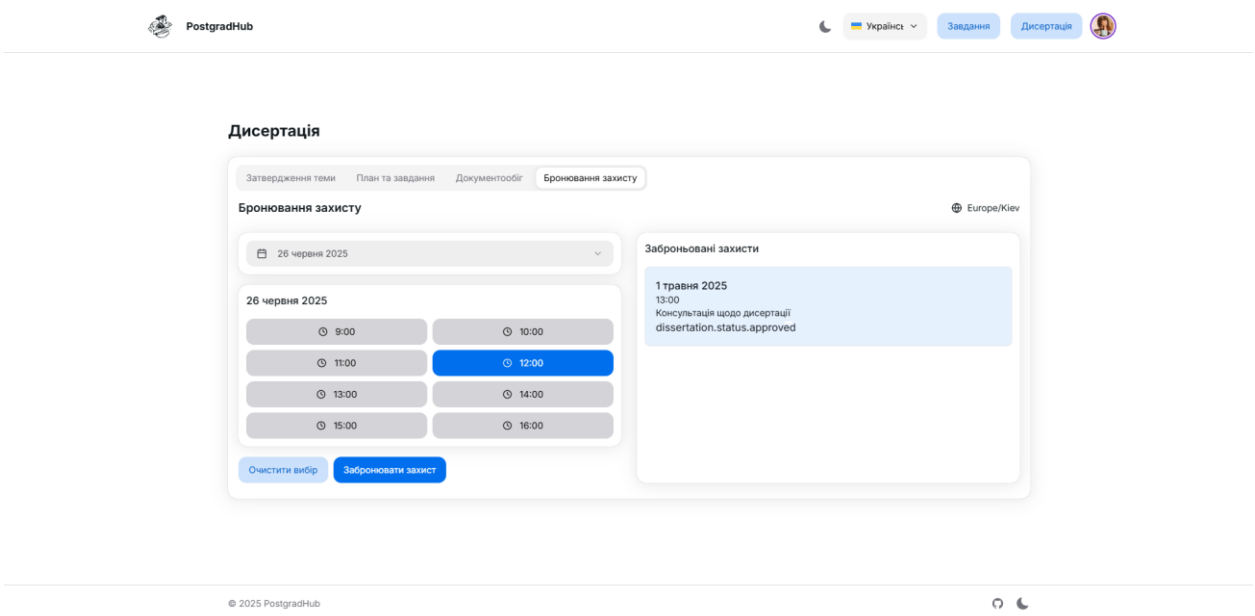


Рисунок 3.14 – Сторінка бронювання дати захисту

Для того, щоб переключити мову інтерфейсу, користувачу необхідно натиснути на випадаюче меню в навігаційній панелі (рис. 3.15) та обрати бажану мову.

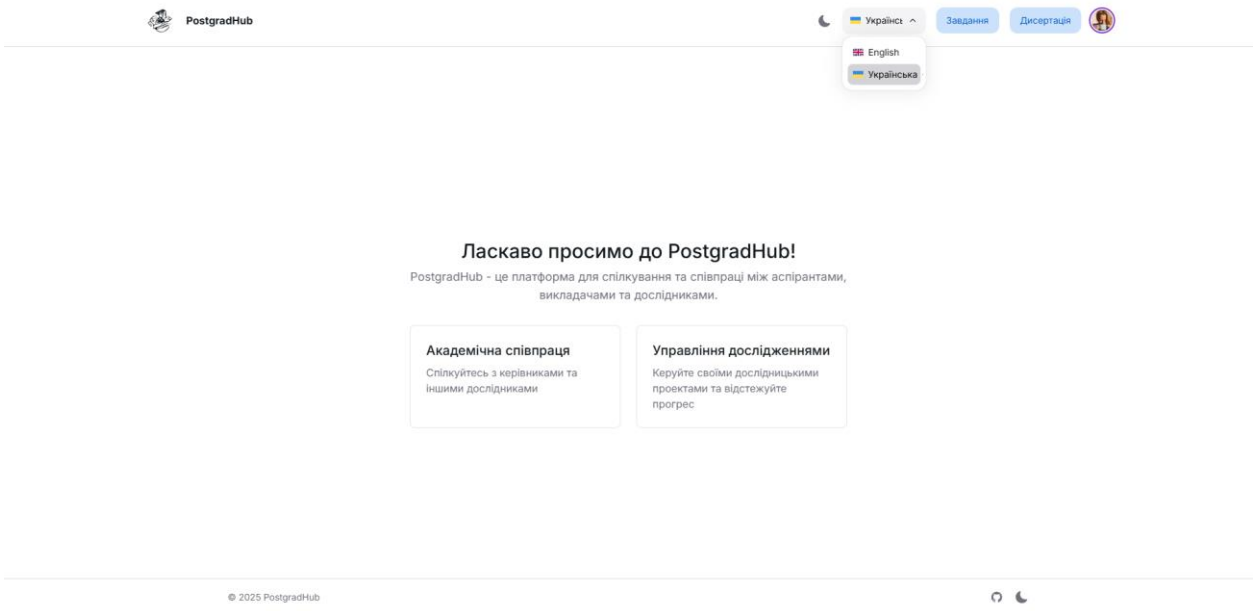


Рисунок 3.15 – Меню вибору мови інтерфейсу

Для того, щоб переключити кольорову тему інтерфейсу на темну, користувачу необхідно натиснути на зображення Місяця в навігаційній панелі додатку, після чого кольорова тема інтерфейсу зміниться на протилежну (рис. 3.16).

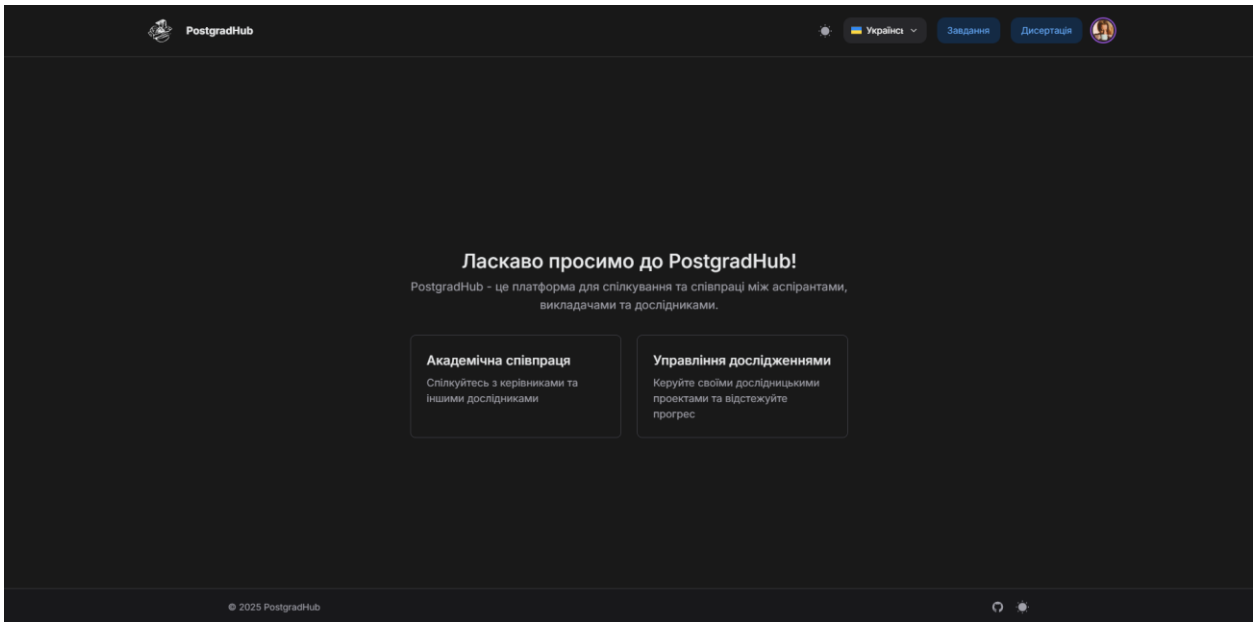


Рисунок 3.16 – Темна тема додатку

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

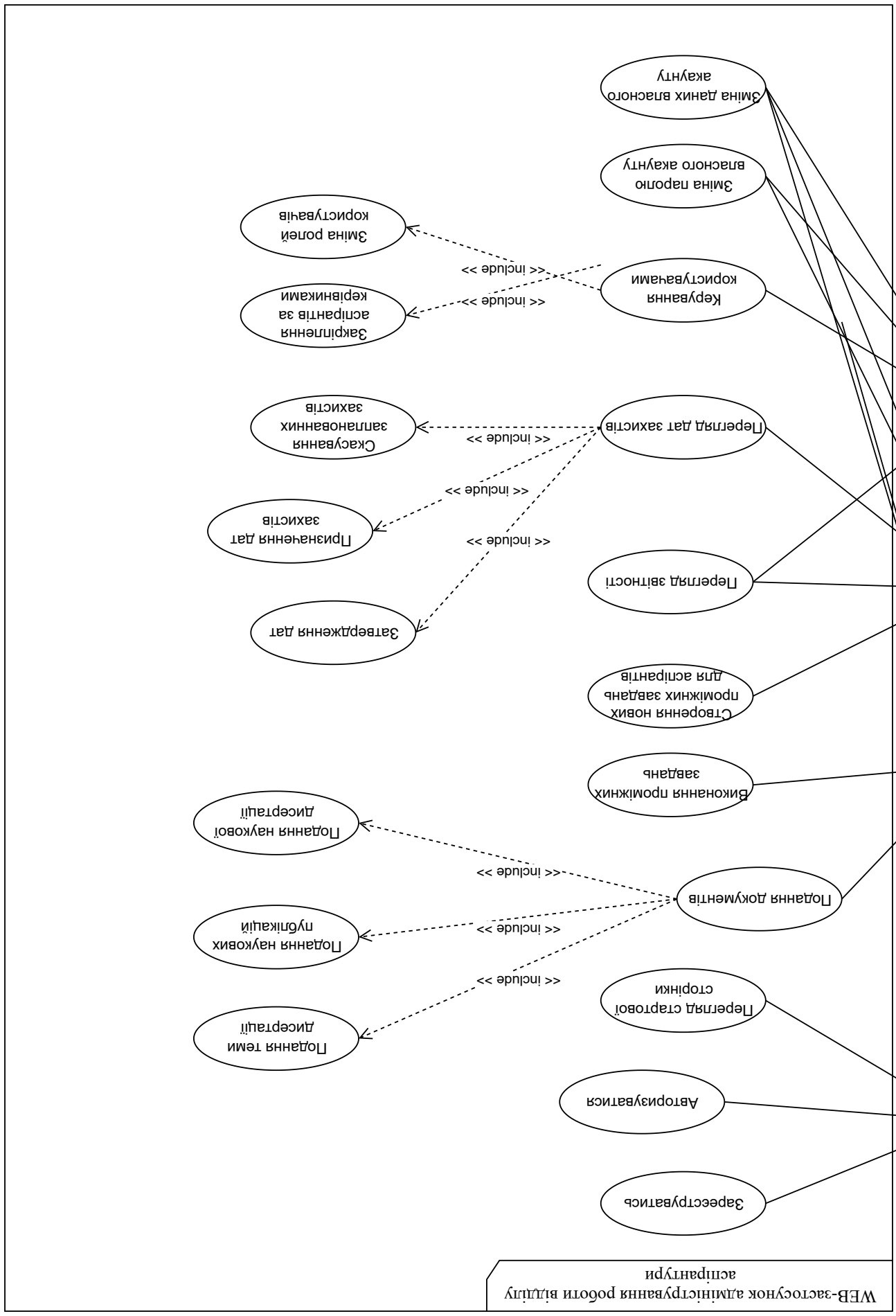
WEB-застосунок адміністрування роботи відділу аспірантури
Графічний матеріал
КПІ.ІП-1323.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олексій ФІНОГЕНОВ

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

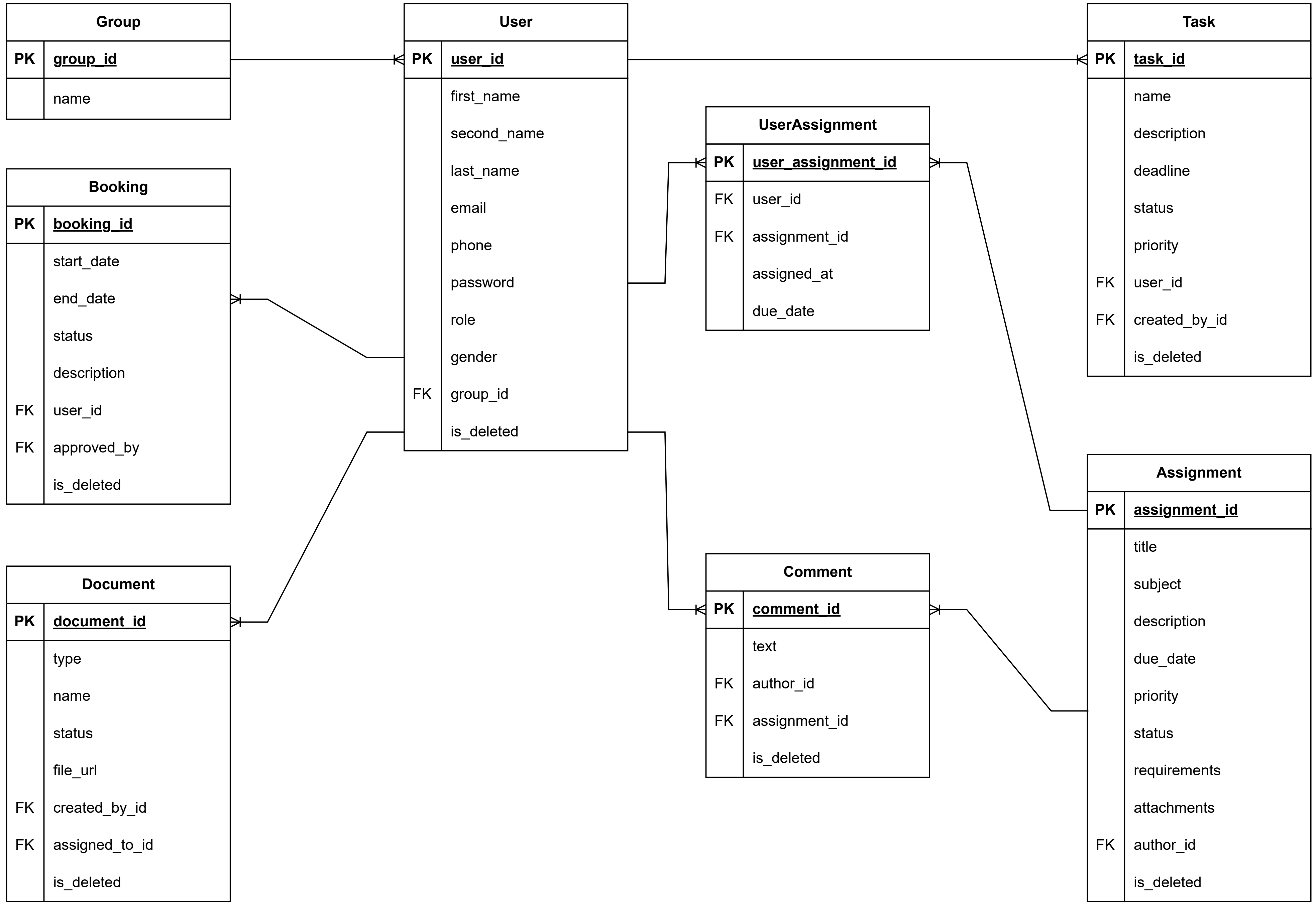
Виконавець:
_____ Євген НЕДЕЛЬЧЕВ

Київ – 2025

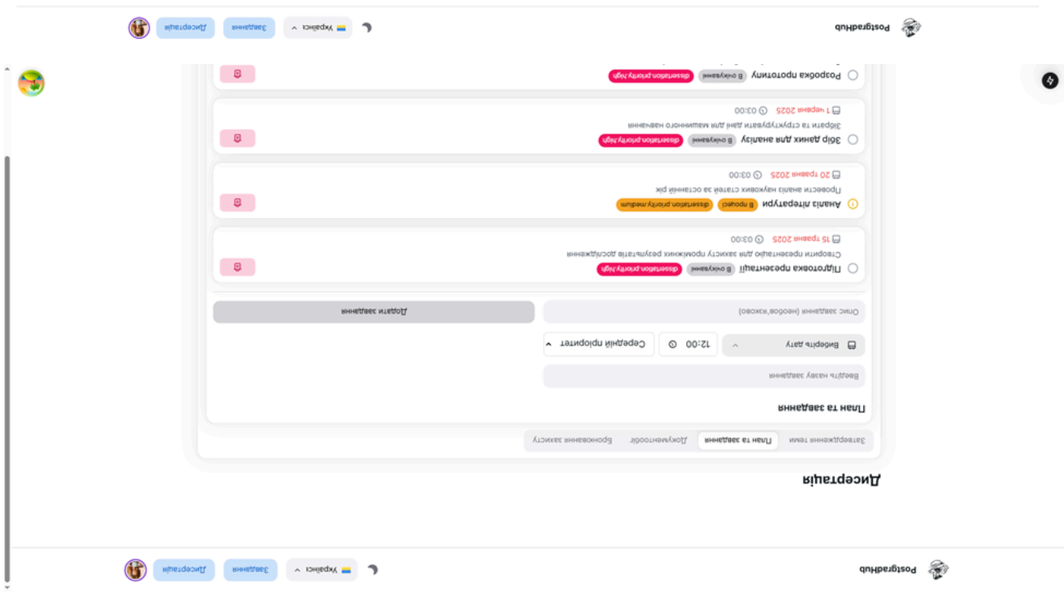


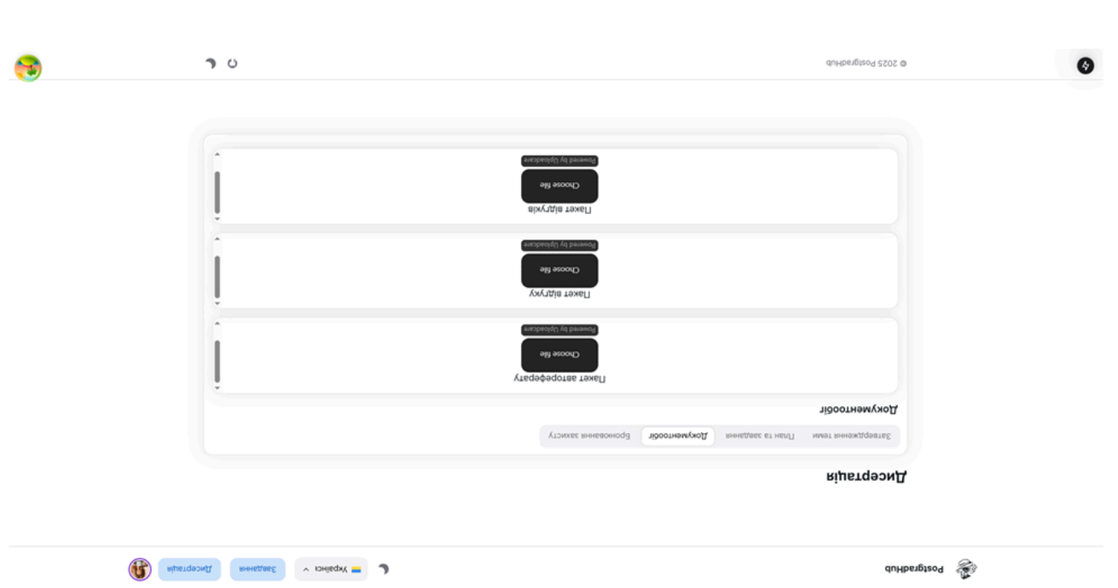
Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1323.045440.06.99.ССВ				
Розробив		Недельчев Є.О.			Схема структура варіантів використання		Літ.	Маса	Масштаб
Перевірив		Фіногенов О.Д.							
Т. контр.							Аркуш 1		Аркушів 1
Н. контр.		Бітковська І.І.			WEB-застосунок адміністрування роботи відділу аспірантури		КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-13		
Затвердив		Жарікова Е.В.							

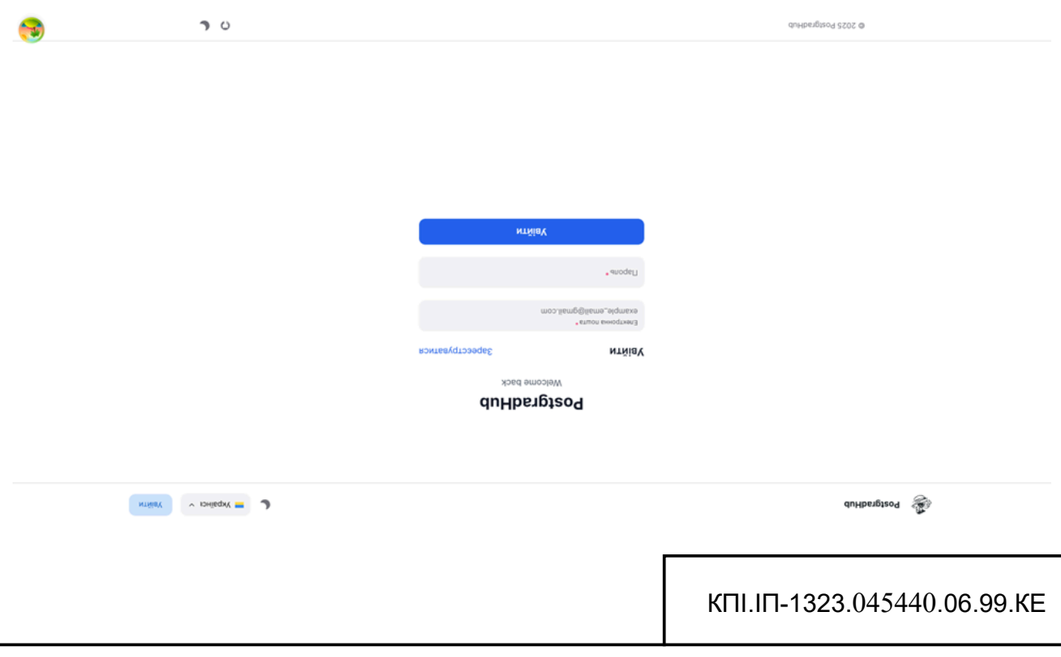
КПІ.ІП-1323.045440.06.99.СБД

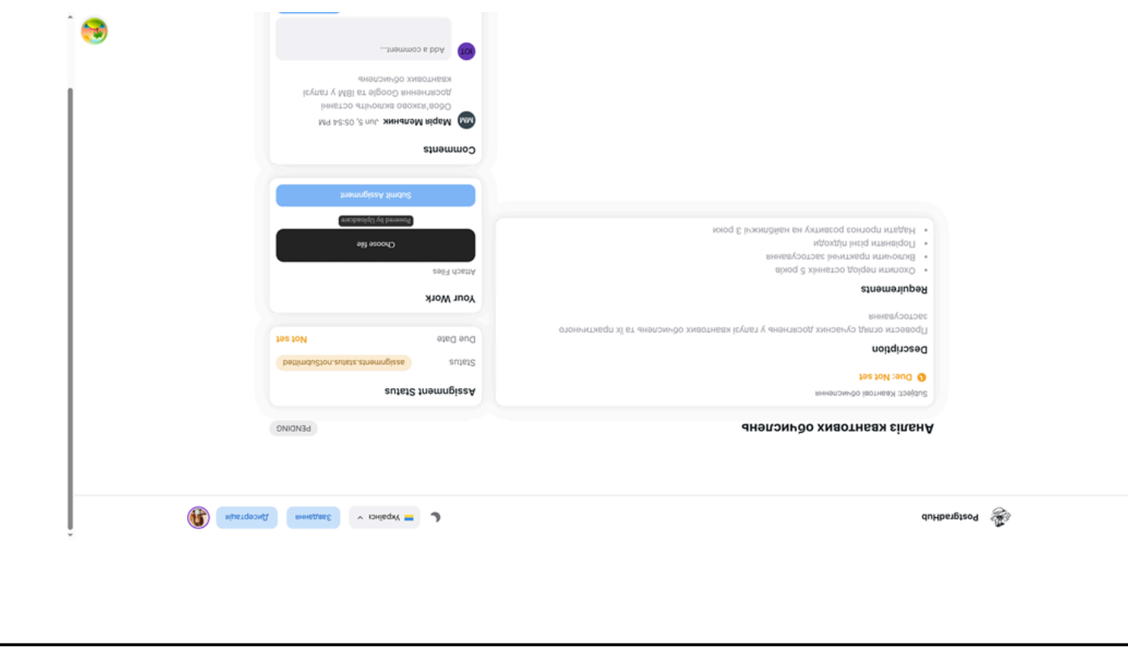


Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІП-1323.045440.06.99.СБД		
Розробив		Недельчев Є.О.			Схема бази даних	Літера	Маса	Масштаб
Перевірив		Фіногенов О.Д.					
Т. контр.						Аркуш 1	Аркушів 1
Н. контр.		Вітковська І.І.			WEB-застосунок адміністрування роботи відділу аспірантури	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-13	
Затвердив		Жаріков Е.В.					









										КПІ.ІП-1323.045440.06.99.КЕ
Зм.	Арк.	№ документа	Підпис	Дата	Кресленна виглядy екранних форм			Літера	Маса	Масштаб
Розробив		Недельчев Є. О.								
Перевірив		Фіногенов О. Д.								
Т. контр.								Аркуш 1		Аркушів 1
Н. контр.		Бітковська І. І.			WEB-застосунок адміністрування роботи відділу аспірантури			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-13		
Затвердив		Жарікова Е.В.								