

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

"На правах рукопису"
УДК 004.4

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
“ ” _____ 2024р.

Магістерська дисертація
на здобуття ступеня магістра
за освітньо-професійною програмою
«Інженерія програмного забезпечення інтелектуальних кібер-фізичних
систем в енергетиці»
зі спеціальності 121 Інженерія програмного забезпечення
на тему: «Моделі та методи спільної розробки та інтеграції окремих частин
програмних продуктів на прикладі розробки веб-додатку кабінету аспіранта
кафедри»

Виконав: студент 2 курсу, групи ТВ-321мп

Лисик Юрій Юрійович
(прізвище, ім'я, по батькові)

(підпис)

Науковий керівник:

професор кафедри ПІЗЕ, доц., д.т.н., Недашківський О.Л.
(посада, вчене звання, науковий ступінь, ім'я ПРІЗВИЩЕ)

(підпис)

Рецензент:

професор кафедри ЦТЕ, проф., д.т.н., Отрох С. І
(посада, вчене звання, науковий ступінь, ім'я ПРІЗВИЩЕ)

(підпис)

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____
(підпис)

Київ — 2024

**Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ
ЕНЕРГЕТИКИ

Кафедра ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ЕНЕРГЕТИЦІ

Рівень вищої освіти другий (магістерський)

За освітньою програмою “Інженерія програмного забезпечення інтелектуальних
кібер-фізичних систем в енергетиці”

Спеціальності 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

Олександр КОВАЛЬ

(підпис)

« ____ » _____ 2024 р.

**З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ**

Лисику Юрію Юрійовичу

(прізвище, ім'я, по батькові)

1. Тема дисертації «Моделі та методи спільної розробки та інтеграції окремих
частин програмних продуктів на прикладі розробки веб-додатку кабінету аспіранта
кафедри»

Науковий керівник Недашківський Олексій Леонідович д.т.н.. доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “08” листопада 2023 року №5202-с

2. Строк подання студентом дисертації 10 січня 2024 року

3. Вихідні дані до роботи мова програмування JavaScript, програмна платформа
Node.js, серверна платформа Nest.js, фреймворк Next.js, автодокументаційна
платформа Swagger, веб-інструмент GitHub Actions, веб-інструмент GitHub Secrets,
платформа Docker, хмарна платформа Heroku, postgresSQL, процес автоматизації
CI/CD, веб-інструмент GitHub Workflow.

4. Перелік питань, які потрібно розробити: розглянути та дослідити різні моделі та
методи, що використовуються для комплексної розробки програмного
забезпечення для навчальних закладів; проаналізувати системи контролю версій з
метою забезпечення безперебійної, злагодженої та контрольованої співпраці
членів команди розробників; розробити схему організації спільної роботи;
розроблено систему автоматизації процесів створення та розгортання, розробити
застосунок, що отримує та інтегрує отриманий код на сервер.

5. Орієнтований перелік ілюстративного матеріалу UML UseCase діаграма, архітектура системи, схема організації спільної роботи, графічне представлення інтерфейсу розробленої системи, приклади роботи програмного модулю.

6. Дата видачі завдання «01» листопада 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	строки виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.11.2022	виконано
2	Дослідження предметної області	01.11.2022 - 01.12.2022	виконано
3	Постановка вимог до проектування системи	01.12.2022 - 15.12.2022	виконано
4	Дослідження існуючих рішень	15.12.2022 - 03.01.2023	виконано
5	Підготовка публікацій	03.01.2023 - 03.03.2023	виконано
6	Розробка програмного продукту	03.03.2023 - 20.09.2023	виконано
7	Тестування	20.09.2023 - 15.10.2023	виконано
8	Захист програмного продукту	16.10.2023 – 20.10.2023	виконано
9	Підготовка магістерської дисертації	21.10.2023 – 17.11.2023	виконано
10	Передзахист	18.12.2023 - 22.12.2023	виконано
11	Захист	15.01.2024 – 19.01.2024	виконано

Студент

(підпис)

Юрій ЛИСИК

(ім'я ПРІЗВИЩЕ)

Науковий керівник

(підпис)

Олексій НЕДАШКІВСЬКИЙ

(ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Актуальність теми. В умовах функціонування академічної спільноти, що швидко розвивається, розробка веб-додатку, спеціально розробленого для навчання в аспірантурі, є не лише вимогою часу, але й стратегічним імперативом. Проведене дослідження стосується гострої потреби в комплексних та інтегрованих рішеннях, підвищення адміністративної ефективності, академічних консультацій та комунікації під час навчання в аспірантурі та може бути використано в інших академічних задачах.

Актуальність полягає в значному потенціалі трансформувати традиційні практики, відкриваючи нову еру цифрової та орієнтованої на користувача підтримки аспірантів.

Мета роботи — підвищення ефективності інтеграції та автоматизації процесу розробки і розгортання частини програмних продуктів в середовищі експлуатації за допомогою розробки механізмів комплексної розробки та інтеграції окремих частин програмних продуктів.

Завдання дослідження:

- дослідити різні моделі та методи, що використовуються для комплексної розробки програмного забезпечення для навчальних закладів;
- проаналізувати системи контролю версій з метою забезпечення безперебійної, злагодженої та контрольованої співпраці членів команди розробників;
- розробити схему організації спільної роботи команди з елементами автоматизації процесів при створенні та розгортанні кабінету аспіранта кафедри;

Об'єкт дослідження — веб-додаток кабінету аспіранта кафедри.

Предмет дослідження — моделі та методи комплексної розробки та інтеграції окремих частин програмних продуктів.

Практична цінність полягає в значному вдосконаленні адміністративних процесів в офісах аспірантів, сприянні більш персоналізованому академічному консультуванню та сприянні безперервному спілкуванню між зацікавленими сторонами. Використовуючи підхід, керований даними, веб-додаток дозволяє приймати обґрунтовані рішення, оптимізуючи управління ресурсами та адаптуючись до динамічних потреб академічного середовища. Дизайн, орієнтований на користувача, забезпечує інтуїтивно зрозумілий і індивідуальний досвід, а спрощена інтеграція навчальних процесів сприяє більш згуртованій та ефективній академічній роботі.

Дисертація складається з вступу, п'яти розділів та висновків. Повний обсяг дисертації складає 98 сторінок, в тому числі 82 сторінка основного тексту, 5 таблиць, 16 рисунків, 3 сторінки списку використаних джерел у кількості 54 найменувань.

Ключові слова: кабінет аспіранта кафедри, веб-додаток, спільна розробка та інтегрування, автоматизація, співпраця університету, практичне навчання, освітній процес.

ABSTRACT

Actuality of theme. In a rapidly evolving academic community, developing a web application specifically designed for graduate education is not only a time requirement, but also a strategic imperative.

The conducted research addresses the urgent need for comprehensive and integrated solutions, improving administrative efficiency, academic counseling and communication during graduate studies and can be used in other academic tasks.

The relevance lies in the significant potential to transform traditional practices, ushering in a new era of digital and user-centric postgraduate support.

The purpose of the work is to increase the efficiency of the integration and automation of the process of development and deployment of part of the software products in the operating environment by means of the development of mechanisms for the complex development and integration of individual parts of the software products.

Objectives of the study:

- explore various models and methods used for complex software development for educational institutions;
- analyze version control systems in order to ensure uninterrupted, coordinated and controlled cooperation of development team members;
- to develop a scheme for the organization of joint work of the team with elements of process automation in the creation and deployment of the post-graduate student's office of the department;

The object of the study is a web application of the post-graduate student's office of the department.

The subject of research is models and methods of complex development and integration of individual parts of software products.

The practical value is in significantly improving administrative processes in graduate student offices, facilitating more personalized academic advising, and facilitating seamless communication between stakeholders. Using a data-driven approach, the web application enables informed decision-making, optimizing resource management and adapting to the dynamic needs of the academic environment. The user-centered design provides an intuitive and personalized experience, while the simplified integration of learning processes promotes a more cohesive and effective academic work.

The thesis consists of an introduction, five chapters and conclusions. The total volume of the dissertation is 98 pages, including 82 pages of the main text, 5 tables, 16 figures, 3 pages of the list of used sources in the amount of 54 names.

Keywords: office of a graduate student of the department, web application, joint development and integration, automation, university cooperation, practical training, educational process.

ЗМІСТ

ВСТУП.....	10
1 ПОСТАНОВКА ЗАДАЧІ ДИСЕРТАЦІЇ.....	13
Висновки до розділу 1	15
2 РЕАЛІЗАЦІЯ ТА ПОБУДОВА РОЗРОБЛЕНОЇ СИСТЕМИ	16
2.1 Опис існуючих аналогів кінцевих програмних систем	17
2.1.1 Електронний простір eCampus.....	18
2.1.2 Система Moodle	19
2.1.3 Порівняння існуючих систем з розробленим забезпеченням	22
2.2 Опис залучених програмних засобів для спільної розробки	23
2.2.1 Мова програмування JavaScript	24
2.2.2 Мова програмування TypeScript	25
2.2.3 Платформа Node.js	26
2.2.4 Серверна платформа Nest.js	27
2.2.5 Фреймворк Next.js	28
2.2.6 Бібліотека React.js.....	29
2.2.7 Автодокументаційна платформа Swagger.....	31
2.2.8 База даних PostgreSQL	32
2.2.9 Веб-інструмент GitHub Actions.....	34
2.2.10 Веб-інструмент GitHub Secrets	36
2.2.11 Платформа Docker	37
2.2.12 Платформа Heroku.....	38
2.2.13 Веб-інструмент GitHub Workflow	40

2.3	Архітектура MVVM	42
	Висновки до розділу 2	44
3	ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ	46
3.1	Серверна частина програмного забезпечення	47
3.2	Клієнтська частина програмного забезпечення	51
	Висновки до розділу 3	55
4	РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ	57
4.1	Інсталяція програмного забезпечення.....	57
4.2	Вимоги до обчислюваної техніки	58
4.3	Демонстрація функціоналу програмного забезпечення.....	60
	Висновки до розділу 4	66
5	РОЗРОБЛЕННЯ СТАРТАПУ ДЛЯ СПІЛЬНОЇ РОЗРОБКИ ТА ІНТЕГРАЦІЇ ОКРЕМИХ ЧАСТИН ПРОГРАМНИХ ПРОДУКТІВ	68
5.1	Опис ідеї стартапу	68
5.2	Технологічний аудит ідеї проекту	75
5.3	Аналіз ринкових можливостей запуску стартап-проекту	76
	Висновки до розділу 5	81
	ВИСНОВКИ.....	82
	СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	84
	ДОДАТОК А.....	87
	ДОДАТОК Б	92

ВСТУП

У динамічному ландшафті програмної інженерії розробка та інтеграція складних веб-додатків вимагають ретельної оркестровки моделей і методів. Ця дипломна магістерська дисертація розпочинає всебічне дослідження «Моделей і методів комплексної розробки та інтеграції» в контексті створення складного веб-додатку, спеціально розробленого для адміністративних тонкощів офісу аспіранта шановного факультету.

Технологічна база. В основі цього дослідження лежить технологічний пакет, ретельно відібраний не тільки для задоволення функціональних вимог сучасної веб-програми, але й для прикладу останніх галузевих стандартів. Використовуючи універсальність і ефективність JavaScript як основної мови програмування в поєднанні з середовищем виконання Node.js і серверною платформою Nest.js, дисертація орієнтується в складному ландшафті серверної розробки. Платформа самодокументування Swagger служить маяком для прозорості та інтуїтивно зрозумілої документації API, забезпечуючи міцну основу для спільної розробки.

Життєвий цикл розробки. Інтеграція GitHub Actions і GitHub Secrets стає ключовим компонентом у забезпеченні спрощеного та безпечного життєвого циклу розробки. Процес автоматизації CI/CD, який підтримується робочими процесами GitHub, обіцяє безперервну інтеграцію та конвеєр доставки, який є не тільки ефективним, але й адаптованим до мінливих потреб проекту. У міру розвитку екосистеми проекту Docker стає ключовим інструментом, пропонуючи контейнеризацію для послідовного розгортання в різноманітних середовищах. Саме розгортання бездоганно організовано на хмарній платформі Heroku, наголошуючи на масштабованості та доступності.

Управління даними. Рівень постійності веб-програми базується на PostgreSQL, потужній системі керування реляційною базою даних із відкритим кодом. Цей вибір ґрунтується на необхідності надійного керування даними, що має вирішальне значення в середовищі, де інформація є невід'ємною частиною

адміністративних робочих процесів.

Фокус дослідження. Дослідження в рамках цієї дисертації не обмежуються лише теоретичними дослідженнями, але поширюються на практичне застосування цих технологій у контексті кабінету аспіранта. Завдяки ретельному аналізу ролі та взаємодії кожної технології мета полягає в тому, щоб зрозуміти, як ці компоненти можуть бути гармонійно інтегровані, утворюючи послідовне та ефективне ціле. Процес автоматизації CI/CD, робочі процеси GitHub, Docker і розгортання на Heroku разом підкреслюють прагнення не лише досліджувати моделі та методи, але й створювати їх у реальному світі.

Структура дисертації. Подальші розділи цієї дисертації розгортаються структуровано, заглиблюючись у теоретичні основи, оглядаючи відповідну літературу та згодом переходячи до практичної сфери розробки веб-додатків. Кожна технологія в стеку ретельно перевіряється на предмет її конкретного внеску в комплексні цілі розвитку та інтеграції. Згідно з висновком дисертації, синтез отриманих результатів сприятиме не лише теоретичному розумінню, але й запропонує практичне розуміння безперервної взаємодії технологій у створенні веб-додатку, адаптованого до унікальних потреб офісу аспіранта.

Ця дослідницька робота, що базується на синтезі теоретичних та практичних знань, має за мету внести вагомий внесок не лише в академічну науку, але й надати конкретні рішення та структури для розробників, адміністраторів і викладачів, які діють в області перетину технологій і академічного середовища.

У роботі буде проведений аналіз поточного стану освітнього процесу в галузі програмної розробки, визначені ключові вимоги та функціональні можливості платформи. Висвітлені аспекти безпеки даних та намічені шляхи впровадження та подальшого розвитку інноваційного проекту.

Дипломна робота "Моделі та методи спільної розробки та інтеграції окремих частин програмних продуктів на прикладі розробки веб-додатку кабінету аспіранта кафедри" націлена на створення реального інструменту, що підтримує студентів у їхньому активному навчанні та розвитку, спрямованого на підвищення якості

університетської освіти в галузі програмної розробки.

Результати дослідження мають практичний вплив на удосконалення університетських проектів, розвиток студентських навичок, ефективне керування проектами та стимулювання інноваційних підходів.

Рекомендації для практичного застосування отриманих результатів включають створення адаптивної та інтуїтивно зрозумілої платформи для максимальної доступності користувачів, впровадження механізмів заохочення активності учасників та постійне оновлення функціоналу з урахуванням потреб користувачів. Забезпечення безпеки та конфіденційності даних користувачів через використання передових методів шифрування є також ключовим аспектом надійності та стабільності платформи.

Робота складається зі вступу, 5 розділів, висновків до кожного розділу, загальних висновків, списку використаних джерел, 2 додатків на 11 сторінках. Загальний обсяг дисертації — 98 сторінки. Основний зміст викладено на 82 сторінках. Роботу проілюстровано 5 таблицями, 16 рисунками.

1 ПОСТАНОВКА ЗАДАЧІ ДИСЕРТАЦІЇ

Об'єкт: веб-додаток для кабінету аспіранта.

Предмет: моделі та методи комплексної розробки та інтеграції окремих частин програмних продуктів.

Мета: підвищення ефективності інтеграції та автоматизації процесу розробки і розгортання частини програмних продуктів в середовищі експлуатації за допомогою розробка механізмів комплексної розробки та інтеграції окремих частин програмних продуктів.

Задачі, які потрібно розв'язати для подолання мети:

- проаналізувати сучасний стан програмних рішень, які використовуються на кафедрах;
- проаналізувати вагомі продуктивні дослідження та практики, пов'язані з розробкою та інтеграцією програмних продуктів для академічних цілей;
- дослідити різні моделі та методи, що використовуються для комплексної розробки програмного забезпечення в цьому контексті;
- підкреслити важливість інтеграції для забезпечення безперебійної роботи;
- проаналізувати системи контролю версій за допомогою, щоб забезпечити безперебійну співпрацю між членами команди;
- дослідити методи та моделі безпечного керування конфіденційною інформацією про проект;
- дослідити методи та моделі ефективного розгортання компонентів програми;
- дослідити методи оптимізації контейнеризації;
- проаналізувати варіанти розміщення веб-додатку та рішення для баз даних;
- розробити систему автоматизації процесів створення та розгортання;
- розробити застосунок, що отримує та інтегрує отриманий код на сервер.

Користувачі: викладачі, аспіранти та адміністративний персонал кафедри.

Вхідні дані:

- специфічні вимоги та порядок роботи кабінету аспіранта;
- відомості про існуючі програмні рішення та їх обмеження;
- джерела даних у реальному часі (наприклад: реєстраційні дані, ролі користувачів);

Вихідні дані: повнофункціональний веб-додаток, розроблений і розміщений на Heroku, з безперебійними конвеєрами CI/CD на основі GitHub і безпечним керуванням конфіденційною інформацією через секрети GitHub.

Стек технологій:

- дії GitHub: для постійної інтеграції та розгортання;
- секрети GitHub: для безпечного керування конфіденційною інформацією проекту;
- файли yml: для визначення конвеєрів і робочих процесів CI/CD;
- dockerfile і docker: для контейнеризації компонентів програми;
- docker compose: для оркестровки контейнера;
- heroku: як хмарна платформа для розміщення веб-програми;
- postgresSQL: як обране рішення бази даних для зберігання даних;
- CI/CD: для автоматизації процесів створення та розгортання;
- swagger: для продуктивності роботи серверної частини та тестування експлуатації запитів відповідно;
- робочий процес GitHub: для оптимізації управління проектами та співпраці;
- організація GitHub: для забезпечення ефективної командної співпраці та контролю версій.

Висновки до розділу 1

Розділ, присвячений проблемі формування ефективних моделей і методів спільної розробки та інтеграції для веб-додатку, дає цінну інформацію про виклики та можливості, які проект намагається вирішити.

Таким чином, проблема формування ефективних моделей і методів спільної розробки та інтеграції для веб-додатку є технічним завданням, який проект готовий вирішити. Визнаючи складність розробки веб-додатків, наголошуючи на інтеграції та використовуючи підхід, орієнтований на користувача та керований даними, проект закладає основу для інноваційних рішень, які можуть змінити адміністративну та академічну підтримку в академічних відділах.

2 РЕАЛІЗАЦІЯ ТА ПОБУДОВА РОЗРОБЛЕНОЇ СИСТЕМИ

Розпочинаючи від ідеї до матеріальної функціональної сутності, впровадження та побудова розробленої системи розгортається як ретельний процес. В основі цієї трансформації лежить стратегічне об'єднання різних технологій, кожна з яких створює єдину структуру веб-додатку, створеного для кабінету аспірантів.

Початок цієї цифрової екосистеми передбачає ретельне налаштування середовища розробки. Ця фаза забезпечує оптимізацію процесу спільного кодування, встановлення контролю версій через Git і створення гілок для сприяння паралельній розробці.

Після того як фундамент закладено, основна архітектура веб-додатку набуває форми. Використовуючи потенціал JavaScript, Node.js і Nest.js, система ретельно розроблена, щоб втілити модульність, оперативність і ефективність. Тонкощі визначення маршрутів і створення надійної серверної інфраструктури розкриваються, створюючи магістраль програми.

Ключовим аспектом цієї розробки є інтеграція Swagger, самодокументованої платформи, яка виходить за межі простої документації. Він стає маяком для розробників і зацікавлених сторін, пропонуючи прозорий погляд на тонкощі структури API, сприяючи співпраці та забезпечуючи надійну основу для майбутніх зусиль щодо розробки.

Процес впровадження синхронізується з безперервною інтеграцією та розгортанням (CI/CD). GitHub Actions виконує ключову роль у воркфлоу, автоматизуючи кроки, такі як компіляція коду, тестування та розгортання. Захист конфіденційної інформації здійснюється через використання GitHub Secrets, що гарантує цілісність процесу розробки. Контейнеризація за допомогою Docker виступає ключовим елементом, який укладає програму та її залежності в

контейнери. Це не лише підвищує ефективність розгортання, але й забезпечує узгодженість у різних середовищах, сприяючи бездоганній взаємодії з користувачем.

Досліджено параметри конфігурації та стратегії розгортання, використовуючи масштабовану інфраструктуру Heroku для забезпечення оптимального рішення для хостингу.

Рівень даних веб-додатку оживає завдяки бездоганній інтеграції PostgreSQL. Визначаються схеми, створюються таблиці та встановлюються зв'язки між програмою та базою даних, формуючи основу для ефективного керування даними.

Суворі методології тестування, такі як модульне, інтеграційне та наскрізне тестування, служать гарантіями надійності системи. Процес забезпечення якості впевнює, що розроблена система відповідає та перевершує очікування.

Працездатність програми забезпечується за допомогою впровадження інструментів моніторингу та механізмів обробки помилок, що забезпечує стійкість і надійність перед обличчям викликів під час виконання.

2.1 Опис існуючих аналогів кінцевих програмних систем

Перш ніж заглиблюватися в тонкощі інноваційної веб-програми, розробленої для офісу аспірантів, доцільно отримати розуміння існуючих рішень, які залишили свій слід у сфері освітніх технологій. Ландшафт систем онлайн-навчання та академічного менеджменту зазнав значних змін протягом багатьох років, з низкою рішень, що задовольняють різноманітні потреби навчальних закладів.

У цьому підрозділі проводиться дослідницьке порівняння трьох відомих систем: eCampus, Moodle та розробленого унікального рішення. Аналізуючи архітектурну основу, ключові характеристики та адаптивність цих систем, це дозволяє позиціонувати розроблене рішення і визначити його унікальні якості.

Аналізуючи кожен з цих систем, відкриваються ідеї, які не тільки

підкреслюють сильні сторони та внесок цих визнаних рішень, але й надають можливість глибше оцінювати передовий дизайн та функціональність даної веб-програми. Розпочнемо з аналізу eCampus, визнаного лідера в галузі онлайн-навчання та академічного менеджменту.

2.1.1 Електронний простір eCampus

eCampus – це інтегрована освітня платформа, призначена для модернізації та оптимізації управління освітою.

Ключові атрибути eCampus включають:

- комплексне керування курсами - eCampus надає комплексне рішення для створення, керування та проведення курсів онлайн. Викладачі можуть легко завантажувати матеріали курсу, завдання та іспити;
- ефективність і доступність - eCampus пропонує зручний інтерфейс для викладачів і студентів, підвищуючи доступність і ефективність;
- співпраця в реальному часі - платформа сприяє взаємодії за допомогою таких інструментів, як дискусійні форуми, чати та відеоконференції;
- автоматизоване оцінювання та виставлення оцінок - eCampus спрощує процеси оцінювання та виставлення оцінок, пропонуючи функції для автоматичного оцінювання питань із варіантами відповідей;
- управління цифровим вмістом - це дозволяє викладачам керувати цифровими навчальними матеріалами, замінюючи традиційні підручники та роздаткові матеріали;
- масштабованість - eCampus є масштабованим і адаптованим до закладів різного розміру, пропонуючи індивідуальні рішення для конкретних потреб офісу аспірантів.

На завершення дослідження eCampus розкриває трансформаційну парадигму в сфері освіти. Цифровий ландшафт створює умови для впровадження

інноваційних методів у сфері навчання, а eCampus є прикладом еволюції традиційних освітніх систем. У даному розділі розглядаються ключові функції та характеристики, які визначають платформу eCampus, відзначаючи її роль у створенні динамічних, доступних та спільних віртуальних навчальних середовищ. Централізуючи ресурси, пропонуючи інтерактивні інструменти та сприяючи спілкуванню між студентами та викладачами, eCampus значною мірою сприяє подоланню географічних бар'єрів і розширенню освітніх можливостей. Зручність і гнучкість, яку він надає, відповідають мінливим потребам сучасних учнів, пропонуючи персоналізований та адаптований досвід навчання.

Орієнтуючись у складнощах цифрової ери, eCampus виступає як життєво важливий інструмент у формуванні майбутнього освіти. Постійний розвиток технологій і методологій електронного навчання підкреслює важливість таких платформ, як eCampus, у сприянні плавному переходу до віртуальних і гібридних моделей навчання. Наслідки виходять за рамки зручності, торкаючись інклюзивності, залучення та демократизації освіти.

Проте вкрай важливо визнати, що ефективність eCampus залежить від продуманого впровадження, постійного вдосконалення та інтеграції педагогічних принципів. Оскільки технології продовжують розвиватися, потенціал eCampus буде змінювати освітній ландшафт.

2.1.2 Система Moodle

Moodle слугує системою управління навчанням (LMS) з відкритим вихідним кодом, яка дозволяє закладам створювати та проводити онлайн-курси. Основні характеристики Moodle включають:

- індивідуалізація - модульність Moodle дає змогу навчальним закладам адаптувати свої онлайн-навчальні середовища відповідно до конкретних потреб;

- модульність - Moodle дозволяє установам розширювати функціональність за допомогою плагінів і розширень, покращуючи процес навчання;
- спільнота активних користувачів - Moodle отримує переваги від спільноти активних користувачів, які роблять внесок у його розвиток і забезпечують регулярні оновлення та вдосконалення;
- природа відкритого вихідного коду - як програмне забезпечення з відкритим кодом, Moodle усуває прив'язку до постачальника та пропонує налаштування та адаптацію;
- розширені інструменти оцінювання - Moodle пропонує такі цифрові інструменти, як тести, завдання та рубрики оцінювання, щоб спростити процес оцінювання;
- онлайн-співпраця - Moodle надає цифрові засоби співпраці, включаючи дискусійні форуми та чати, щоб відтворити традиційні особисті взаємодії;
- відстеження прогресу - звіти про курси та аналітика в Moodle надають інструкторам інформацію для відстеження прогресу студентів і покращення їхніх методів навчання.

Включення Moodle у дану систему знаменує стратегічний стрибок у покращенні досвіду навчання. Moodle, система управління навчанням (LMS) з відкритим вихідним кодом, є наріжним каменем у зміні того, як керуються курсами та як студенти взаємодіють із навчальним контентом.

Сила Moodle полягає в його універсальності. Він служить централізованим центром для організації курсів, доставки контенту та спільного навчання. За допомогою таких модулів, як форуми, тести та завдання, Moodle перетворює статичні курси на динамічний інтерактивний досвід для аспірантів.

Повна інтеграція — це не просто технічне досягнення; мова йде про створення єдиної взаємодії з користувачем. Інтерфейси програмного інтерфейсу та з'єднувачі доповнюють розрив між веб-програмою та Moodle, забезпечуючи плавний перехід користувачів між функціями обох систем.

Можливості адаптивного навчання Moodle переосмислюють персоналізовану освіту. Розробляючи шляхи навчання на основі індивідуального прогресу, він створює середовище, де робота кожного аспіранта є вагомою в змістовності. Механізми відстеження прогресу надають цінну інформацію, що дозволяє своєчасно втручатися у студентів, яким може знадобитися додаткова підтримка.

Окрім своєї ролі в управлінні курсами, Moodle сприяє співпраці. Дискусійні форуми, функції обміну повідомленнями та інструменти для спільної роботи створюють віртуальну навчальну спільноту, покращуючи загальний освітній досвід для аспірантів.

Інструменти оцінювання Moodle надійні, що дозволяє ефективно оцінювати успішність студентів. Тести, завдання та механізми оцінювання плавно інтегруються в ширший профіль студента в веб-додатку, забезпечуючи повний огляд академічного прогресу.

Вирішальним фактором є масштабованість. Стратегія інтеграції гарантує, що система може вмістити зростаючу кількість користувачів і курсів. Крім того, він передбачає й охоплює майбутні розробки в Moodle, позиціонуючи дану систему для постійного узгодження з оновленнями та новими функціями.

Таким чином, Moodle — це не просто інтегрований інструмент; це революція в управлінні навчанням. Ця співпраця між веб-програмою та Moodle виходить за рамки технологій — вона означає прагнення створити збагачене, спільне та персоналізоване навчальне середовище для спільноти аспірантів.

2.1.3 Порівняння існуючих систем з розробленим забезпеченням

Нова система, створена з використанням таких технологій, як GitHub Actions,

Docker, Heroku та PostgreSQL, являє собою сучасне, добре адаптоване та масштабоване рішення для оптимізації управління кабінетом аспірантів.

Ключові моменти:

- розширена автоматизація - GitHub Actions і конвеєри CI/CD спрощують процеси створення та розгортання, забезпечуючи ефективність і надійність;
- безпечне керування - використання секретів GitHub підвищує безпеку конфіденційної інформації проекту, роблячи дану систему безпечним рішенням;
- налаштованість - модульність технологічного стеку дозволяє створити індивідуальне рішення з бездоганною інтеграцією таких компонентів, як Docker, PostgreSQL і GitHub Workflow;
- співпраця - організація GitHub сприяє ефективній командній співпраці та контролю версій, покращуючи робочий процес і управління проектами;
- мобільна доступність - дана система підтримує мобільний доступ, забезпечуючи зручність використання смартфонів і планшетів;
- масштабованість - використання хмарних технологій через Heroku забезпечує масштабованість, щоб відповідати мінливим потребам кабінету аспірантів;
- цифрова трансформація - дана система являє собою стрибок від аналогових методів до цифрової, інтегрованої та високоефективної платформи, покращуючи доступність і масштабованість, забезпечуючи безпеку даних і простоту використання.

2.2 Опис залучених програмних засобів для спільної розробки

Під час спільної розробки проекту для підтримки спільних зусиль було

ретельно вибрано потужний набір програмних засобів. Основною мовою програмування є JavaScript, яка керує динамічними аспектами веб-програми. Виконання на стороні сервера забезпечує програмна платформа Node.js, відома своїми асинхронними можливостями та масштабованістю.

Nest.js виступає системою на стороні сервера, будуючись на міцній основі Node.js. Його модульна архітектура забезпечує ремонтпридатність і масштабованість. Доповнює це фреймворк Next.js, інструмент на основі React, який покращує рендеринг на стороні клієнта для оптимізації продуктивності сторінки.

Для ясності документації API Swagger виступає як самодокументована платформа, допомагаючи розробникам зрозуміти, тестувати та співпрацювати над API. GitHub Actions і GitHub Secrets покращуючи робочий процес розробки. Перший автоматизує процеси CI/CD, забезпечуючи безперебійне тестування та розгортання, тоді як другий безпечно керує конфіденційною інформацією.

Docker інкапсулює програму та її залежності, сприяючи узгодженості в різноманітних середовищах і спрощуючи розгортання. Хмарна платформа Heroku виконує роль хостингу, забезпечуючи масштабованість і легкість розгортання. PostgreSQL бере на себе кермо системи керування реляційною базою даних, закріплюючи рівень даних веб-додатку.

Прагнення до автоматизації втілено в процесах CI/CD, що забезпечує постійну інтеграцію та ефективність розгортання. GitHub Workflow, працюючи в тандемі з GitHub Actions, керує цим конвеєром, візуально представляючи процес розробки.

У поєднанні цей арсенал утворює інтегровану екосистему, яка дає змогу команді розробників спільно долати складнощі створення, тестування та розгортання веб-програми. Кожен інструмент, ретельно підібраний, відображає прагнення передовим практикам, масштабованості та створенню стійкого та зручного програмного продукту.

2.2.1 Мова програмування JavaScript

Під час спільної розробки проекту для об'єднання зусиль було уважно відібрано набір потужних програмних засобів. Основною мовою програмування для динамічних аспектів веб-програми є JavaScript. Серверна частина реалізована за допомогою Node.js з його асинхронними можливостями та масштабованістю. Nest.js виступає як система на стороні сервера, побудована на міцній основі Node.js, забезпечуючи модульність і масштабованість. Next.js, фреймворк на основі React, використовується для оптимізації рендерингу на стороні клієнта.

Swagger виступає як самодокументована платформа для API, допомагаючи розробникам розуміти, тестувати та співпрацювати над API. GitHub Actions і GitHub Secrets покращують робочий процес розробки, автоматизуючи процеси CI/CD та керуючи конфіденційною інформацією відповідно. Docker інкапсулює програму та її залежності, спрощуючи розгортання та забезпечуючи узгодженість в різних середовищах.

Heroku виступає як хмарна платформа для хостингу, забезпечуючи масштабованість та легкість розгортання. PostgreSQL відповідає за систему керування реляційною базою даних у веб-додатку. Процеси CI/CD інтегровані в щоденну розробку за допомогою GitHub Workflow та GitHub Actions, надаючи постійну інтеграцію та ефективність розгортання.

Узгоджений набір інструментів утворює інтегровану екосистему, яка дозволяє розробникам успішно вирішувати завдання створення, тестування та розгортання веб-програми. Кожен інструмент відображає передові практики, масштабованість та створення стійкого та зручного програмного продукту.

2.2.2 Мова програмування TypeScript

TypeScript, наднабір JavaScript, увійшов до технічного інструментарію як надійний рівень статичної типізації. Додавши статичні типи до традиційно динамічно типізованого JavaScript, TypeScript заявляє про себе як проактивний

охоронець від потенційних помилок під час розробки.

Сумісність панує на висоті, оскільки TypeScript легко інтегрується з існуючими кодовими базами JavaScript. Цей набір забезпечує безперебійний процес впровадження, дозволяючи проекту використовувати переваги розширеної статичної типізації, не порушуючи встановлені робочі процеси.

Розширена підтримка інструментів є відмінною рисою TypeScript. Автозавершення, інструменти рефакторингу та інтелектуальна навігація по коду стають невід'ємною частиною, підвищуючи продуктивність розробника та спрощуючи кодування.

Відповідно до стандартів ECMAScript, TypeScript залишається синхронізованим із функціями JavaScript, що розвиваються. Таке узгодження дає змогу команді розробників використовувати найновіші можливості, використовуючи вдосконалення статичного набору тексту, властиві TypeScript.

Читабельність і ремонтпридатність отримали істотний приріст. Явне визначення типів даних у TypeScript не тільки покращує документацію коду, але й сприяє покращенню читабельності коду. Цей характер самодокументування спрощує розуміння, сприяє співпраці та простоті обслуговування.

Отже, TypeScript виступає як надійний компаньйон JavaScript, впроваджуючи в проект передбачуваність, надійність і розширені інструменти розробника. Це посилення забезпечує стійку кодову базу, яку можна підтримувати, що відповідає прагненням до надійної веб-програми.

2.2.3 Платформа Node.js

Node.js, який є основою операцій на стороні сервера, забезпечує безліч функцій і можливостей, які глибоко впливають на надійність і масштабованість веб-програми.

Node.js відомий своєю асинхронною моделлю виконання, що дозволяє

виконувати операції без блокування. Це означає, що сервер може ефективно обробляти кілька запитів одночасно, підвищуючи швидкість реагування програми. Керована подіями архітектура, керована циклом подій, гарантує, що сервер завжди готовий відповідати на нові запити, сприяючи оптимальній продуктивності навіть за високих навантажень.

Масштабованість є визначальною характеристикою Node.js. Його архітектура розроблена для обробки великої кількості одночасних підключень, що робить його чудовим вибором для програм із різним навантаженням користувачів. Ця масштабованість має вирішальне значення, оскільки програма зростає, забезпечуючи бездоганний досвід для користувачів незалежно від попиту.

Node.js поставляється з npm (Node Package Manager), надійною екосистемою з великою колекцією пакетів і модулів. Це обширне сховище інструментів з відкритим кодом прискорює процес розробки. Він надає рішення для загальних функціональних можливостей, що дозволяє команді інтегрувати існуючі, добре перевірені модулі, заощаджуючи час і зусилля.

Узгодженість між клієнтським і серверним JavaScript є значною перевагою. Розробники, знайомі з JavaScript для зовнішньої розробки, можуть плавно перейти до серверної розробки за допомогою Node.js. Ця уніфікованість сприяє багаторазовому використанню коду, спрощує робочий процес розробки та скорочує криву навчання для членів команди.

Наприклад, можливості в реальному часі: Node.js добре підходить для програм реального часу, таких як програми чату та онлайн-ігри. Його керована подіями архітектура дозволяє миттєво реагувати на дії користувача, забезпечуючи безперебійний інтерактивний досвід.

Будучи платформою з відкритим вихідним кодом, Node.js виграє від живої та активної спільноти. Постійні оновлення та вдосконалення гарантують, що даний додаток відповідає останнім розробкам в екосистемі Node.js, включаючи підвищення продуктивності, виправлення безпеки та нові функції.

Таким чином, Node.js — це не просто середовище виконання на стороні

сервера; це стратегічний вибір для даного проекту, який забезпечує асинхронні можливості, масштабованість, багату екосистему та уніфікований досвід розробки. Його вплив виходить за рамки базових серверних операцій, формуючи саму основу продуктивності та швидкодії веб-програми.

2.2.4 Серверна платформа Nest.js

Nest.js, розширення Node.js, відіграє ключову роль у визначенні серверної архітектури веб-програми. З акцентом на модульність і масштабованість Nest.js дає змогу команді розробників створювати компоненти на стороні сервера, які не тільки зручні в обслуговуванні, але й легко розширюються.

Модульна архітектура фреймворку організовує код у багаторазово використовувані модулі, покращуючи організацію та керування проектом. Кожен модуль включає певну функціональність, сприяючи чіткій та організованій структурі проекту.

Надійна система ін'єкції залежностей є відмінною особливістю Nest.js. Ця система дозволяє впроваджувати залежності в компоненти, сприяючи слабкому зв'язку та покращуючи ремонтпридатність, роблячи компоненти модульними та взаємозамінними.

Декоратори та метадані забезпечують механізм для визначення класів та їхніх ролей у програмі. Декоратори, як виразні анотації, покращують читабельність коду, тоді як метадані пропонують важливу інформацію про структуру програми, сприяючи бездоганній інтеграції та розумінню.

Nest.js легко інтегрується з TypeScript, використовуючи його функції статичного набору для підвищення якості коду та продуктивності розробника. Вбудована підтримка TypeScript дозволяє розробникам використовувати мовні можливості, такі як інтерфейси та перевірка типу.

Проміжне програмне забезпечення та перехоплювачі в Nest.js пропонують

надійний підхід до обробки запитів і відповідей. Функції проміжного програмного забезпечення можна застосовувати глобально або локально до певних маршрутів, забезпечуючи гнучкість у управлінні циклом запит-відповідь HTTP. Перехоплювачі дозволяють трансформувати дані до досягнення обробників маршрутів.

Окрім можливостей для окремого додатка, Nest.js розроблено з урахуванням масштабованості, що робить його добре придатним для архітектур мікросервісів. Його модульна структура та підтримка шаблонів обміну повідомленнями полегшують розробку розподілених і масштабованих систем. Інтеграція з такими інструментами обміну повідомленнями, як RabbitMQ і Redis, покращує зв'язок між мікросервісами.

Таким чином, Nest.js — це архітектурний інструмент, який виходить за рамки простого статусу фреймворку. Це дає змогу команді розробників створювати компоненти на стороні сервера, які є не тільки надійними та масштабованими, але також придатними для обслуговування та адаптацією до мінливих потреб веб-програми.

2.2.5 Фреймворк Next.js

Next.js, фреймворк на основі React, революціонізує візуалізацію проекту на стороні клієнта. Його майстерність полягає у відображенні на стороні сервера (SSR), створенні сторінок на сервері для оптимізації часу завантаження та чудового SEO.

У сфері створення статичних сайтів (SSG) Next.js блищить. Попереднє відтворення сторінок під час створення дає статичні HTML-файли, які ефективно обслуговуються з CDN для підвищення продуктивності.

Автоматичне поділ коду є видатним, розбиваючи код JavaScript на керовані частини для швидшого початкового завантаження сторінки та оптимізації ресурсів.

Next.js повністю інтегрує підтримку TypeScript, підвищуючи якість коду завдяки перевагам статичного набору тексту та розширеним інструментам.

Маршрути API спрощено, що дозволяє створювати безсерверні функції в рамках проекту. Це спрощує розробку функціональних можливостей серверної частини, сприяючи єдиному підходу до логіки на стороні клієнта та на стороні сервера.

Гаряча заміна модулів (HMR) представляє оперативне перезавантаження компонентів під час розробки, усуваючи потребу в повному оновленні сторінки та прискорюючи цикл розробки.

Універсальні методи отримання даних, що охоплюють генерацію статичного сайту, рендеринг на стороні сервера та рендеринг на боці клієнта, дають змогу розробникам адаптувати свій підхід до конкретних потреб сторінки чи компонента.

Next.js виходить за рамки своєї ролі фреймворку React, ставши потужним інструментом, що формує стратегію візуалізації на стороні клієнта. Сильні сторони SSR, SSG і автоматичного поділу коду сприяють більш ефективній, продуктивній і зручній для розробників веб-програмі.

2.2.6 Бібліотека React.js

React.js, динамічна бібліотека JavaScript, служить основою проекту, забезпечуючи декларативний підхід до створення інтерфейсів користувача. React.js, відомий своєю ефективністю та гнучкістю, дає нам змогу з легкістю створювати інтерактивні та безперебійні веб-додатки.

За своєю суттю React.js використовує декларативний синтаксис, що дозволяє розробникам описувати бажаний результат, не заглиблюючись у тонкощі покрокового процесу. Це не тільки спрощує розробку, але й покращує читабельність коду та зручність обслуговування.

Компонентна архітектура бібліотеки є ключовою перевагою, розбиваючи

інтерфейс користувача на багаторазові та модульні компоненти. Кожен компонент інкапсулює свою логіку, стиль і функціональність, сприяючи повторному використанню коду та модульній розробці.

React.js використовує Virtual DOM як метод оптимізації. Створюючи віртуальне представлення DOM і порівнюючи його з фактичним DOM, React мінімізує оновлення, зменшуючи обчислювальне навантаження та підвищуючи загальну продуктивність.

Забезпечуючи односторонній потік зв'язування даних, React.js спрощує керування даними за допомогою односпрямованого потоку даних від батьківських до дочірніх компонентів. Такий підхід полегшує відстеження та керування змінами стану в програмі.

JSX, синтаксичне розширення, представлене React.js, дозволяє бездоганно інтегрувати HTML-подібний код у JavaScript. Таке поєднання розмітки та логіки покращує читабельність коду та спрощує процес визначення компонентів інтерфейсу користувача.

Поява React Hooks забезпечує зміну парадигми, уможливлуючи використання функцій стану та життєвого циклу у функціональних компонентах. Цей підхід просуває принципи функціонального програмування та пропонує більш стислий і виразний стиль розробки.

Окрім своїх технічних переваг, React.js процвітає в надійній екосистемі та залученій спільноті. Наявність сторонніх бібліотек, інструментів і великої кількості онлайн-ресурсів прискорює розробку та забезпечує відповідність React.js новим тенденціям і найкращим практикам.

2.2.7 Автодокументаційна платформа Swagger

Swagger відіграє ключову роль у екосистемі розробки, надаючи низку функцій, які спрощують дизайн, розробку та документацію API.

Автоматизована документація API: здатність Swagger автоматично генерувати документацію API з анотацій вихідного коду спрощує часто стомлююче завдання створення документації. Ця автоматизація не тільки економить час, але й гарантує, що документація завжди синхронізована з фактичною реалізацією API.

Інтерактивне дослідження API: користувальницький інтерфейс Swagger пропонує розробникам динамічне та інтерактивне середовище для вивчення та тестування кінцевих точок API. Це дослідження в реальному часі покращує процес навчання, дозволяючи розробникам експериментувати з різними параметрами та розуміти поведінку API на льоту.

Стандартизований дизайн API: Специфікація OpenAPI, яку підтримує Swagger, представляє стандартизований підхід до дизайну API. Ця спільна мова сприяє чіткій комунікації між різними командами, забезпечує узгодженість принципів проектування та просуває найкращі практики протягом життєвого циклу розробки.

Генерація коду: підтримка Swagger генерації коду є потужною функцією, яка прискорює розробку. Створюючи клієнтські бібліотеки або серверні заглушки зі специфікацій Swagger/OpenAPI, розробники можуть швидко розпочати процес впровадження, зменшуючи ймовірність невідповідності між документацією та кодом.

Перевірка та тестування: Swagger допомагає у перевірці API, дозволяючи розробникам визначати та застосовувати обмеження через специфікації. Це не тільки забезпечує дотримання стандартів, але й сприяє надійному дизайну API. Крім того, інтерактивний характер Swagger UI служить ефективним інструментом тестування, що дозволяє розробникам надсилати запити, перевіряти відповіді та вирішувати проблеми безпосередньо в документації.

Співпраця та досвід розробників: повна та інтерактивна документація Swagger сприяє співпраці між командами розробників, менеджерами з продуктів та іншими зацікавленими сторонами. Спільне розуміння функцій API покращує спілкування, вирівнює очікування та оптимізує загальний робочий процес

розробки. Це середовище для співпраці сприяє позитивному досвіду розробника.

Інтеграція з екосистемою: Swagger легко інтегрується з різними інструментами та платформами, що ще більше підвищує його корисність. Ця інтеграція поширюється на фреймворки розробки, системи контролю версій і конвеєри безперервної інтеграції/безперервного розгортання (CI/CD), забезпечуючи плавний і злагоджений процес розробки.

Таким чином, Swagger виходить за рамки своєї ролі інструменту документування; він служить багатогранною платформою, яка позитивно впливає на розробку API. Його функції, починаючи від автоматизованого документування й закінчуючи інтерактивним дослідженням і генерацією коду, разом сприяють ефективній, стандартизованій та спільній розробці API.

2.2.8 База даних PostgreSQL

PostgreSQL, стійка система керування реляційними базами даних із відкритим вихідним кодом, є основою стеку технологій, забезпечуючи надійну інфраструктуру для керування різноманітними та складними сценаріями даних.

Відомий своєю надійністю та стабільністю, PostgreSQL є надійним партнером у забезпеченні цілісності даних, особливо в середовищах із високим рівнем транзакцій. Його здатність керувати значними наборами даних із постійною доступністю є ключовою для надійності програми.

Розширені типи даних, які підтримує PostgreSQL, включаючи масиви, JSON і hstore, пропонують багату палітру для моделювання складних структур. Ця гнучкість представлення даних ідеально узгоджується з динамічними потребами програми, дозволяючи зберігати та отримувати дані різних типів.

Розширюваність PostgreSQL за допомогою спеціальних функцій є видатною функцією. Використовуючи різні мови програмування, розробники можуть створювати індивідуальні функції та оператори, полегшуючи вбудовування

спеціальної бізнес-логіки безпосередньо в базу даних. Ця можливість розширення служить потужним інструментом для оптимізації обробки даних і підвищення загальної ефективності системи.

Складний оптимізатор запитів, вбудований у PostgreSQL, є ключовим гравцем на арені продуктивності. Його вміле оброблення складних запитів SQL забезпечує оптимальний пошук і маніпулювання даними, значно сприяючи швидкодії реагування програми, навіть в умовах складних операцій з базою даних.

Відданість PostgreSQL відповідності ACID і надійним механізмам керування паралелізмом забезпечує узгодженість даних у багатокористувацьких середовищах. Це дотримання фундаментальних принципів бази даних має вирішальне значення для безперебійної роботи програми, особливо в сценаріях, що вимагають високого паралелізму.

Масштабування інфраструктури бази даних стало можливим завдяки підтримці PostgreSQL для розділення, реплікації та кластеризації. Ці функції дають нам змогу адаптуватися до мінливих вимог програми, забезпечуючи плавне масштабування бази даних разом із зростанням бази користувачів і обсягу даних.

Розробка, керована спільнотою, гарантує, що PostgreSQL залишається в авангарді інновацій. Активна участь спільноти з відкритим кодом забезпечує регулярні оновлення, виправлення безпеки та інтеграцію нових функцій, разом покращуючи загальну функціональність і стійкість системи баз даних.

Безпека є першочерговою у PostgreSQL завдяки таким функціям, як підтримка SSL, шифрування даних у стані спокою та контроль доступу на основі ролей. Ці надійні заходи безпеки узгоджуються з зобов'язанням щодо захисту конфіденційної інформації та дотримання суворих стандартів захисту даних.

Підсумовуючи, PostgreSQL — це не просто база даних; це динамічний і багатофункціональний союзник, який посилює можливості додатка з керування даними. Його надійність, гнучкість, розширюваність, оптимізація запитів і відповідність галузевим стандартам роблять його незамінним елементом технологічної екосистеми.

2.2.9 Веб-інструмент GitHub Actions

GitHub Actions є ключовим елементом інструментарію розробки, безперешкодно організовуючи автоматизовані робочі процеси на платформі GitHub. Це інтегроване рішення CI/CD забезпечує спрощений підхід до створення, тестування та розгортання програм.

Перевага GitHub Actions полягає в його універсальності. Це дозволяє нам визначати спеціальні робочі процеси за допомогою синтаксису YAML, забезпечуючи високий рівень гнучкості та налаштування. Це дає змогу команді розробників автоматизувати широкий спектр завдань, від простих процесів збірки та тестування до складних конвеєрів розгортання.

Тісна інтеграція платформи зі сховищами GitHub гарантує, що робочі процеси запускаються автоматично під час таких подій, як надсилання коду, запити на отримання або нові випуски. Ця автоматизація значно зменшує ручне втручання, сприяючи більш ефективному життєвому циклу розробки.

GitHub Actions підтримує різноманітну екосистему попередньо створених дій і дозволяє нам створювати власні, забезпечуючи легку інтеграцію зі сторонніми інструментами та службами. Ця розширюваність гарантує, що робочі процеси можуть легко вписуватися в існуючий стек розробки.

Візуальне представлення робочих процесів в інтерфейсі GitHub Actions надає інтуїтивно зрозумілий спосіб зрозуміти хід виконання завдань і вирішити проблеми. Така прозорість покращує співпрацю між членами команди, сприяючи спільному розумінню процесів розробки та розгортання.

Крім того, GitHub Actions підтримує матричні збірки, що дозволяє нам тестувати програми в кількох середовищах і конфігураціях одночасно. Ця можливість є безцінною для забезпечення сумісності та надійності на різних платформах.

Підсумовуючи, GitHub Actions служить надійним центром автоматизації, який об'єднує різні компоненти процесів розробки та розгортання. Його гнучкість, можливості автоматизації та повна інтеграція зі сховищами GitHub сприяють створенню ефективного середовища розробки для спільної роботи.

2.2.10 Веб-інструмент GitHub Secrets

GitHub Secrets відіграє вирішальну роль у зміцненні безпеки робочих процесів GitHub Actions, дозволяючи безпечно зберігати та керувати конфіденційною інформацією, такою як ключі API, маркери доступу та ключі шифрування.

Однією з головних переваг GitHub Secrets є його здатність захищати конфіденційні дані від випадкового розголошення. Замість жорсткого кодування конфіденційної інформації безпосередньо у файли робочого процесу, Secrets надають безпечне сховище для зберігання цієї інформації, гарантуючи, що вона залишається конфіденційною та недоступною для неавторизованих користувачів.

GitHub Secrets бездоганно інтегрується з робочими процесами GitHub Actions, дозволяючи нам посилатися на ці секрети та використовувати їх під час виконання різних кроків у процесах автоматизації. Ця інтеграція покращує безпеку, запобігаючи ненавмисному розкриттю конфіденційної інформації в кодовій базі.

Секрети визначаються та керуються на рівні сховища, забезпечуючи детальний рівень контролю над доступом. Це означає, що різні репозиторії в організації GitHub можуть мати різні набори секретів, узгоджуючи принцип найменших привілеїв і гарантуючи, що доступ до конфіденційних даних жорстко контролюється.

Крім того, GitHub Secrets підтримує секрети певного середовища, що дозволяє нам адаптувати доступність конфіденційної інформації до конкретних гілок або середовищ у рамках робочого процесу розробки. Ця функція підвищує

гнучкість, зберігаючи високий рівень безпеки.

Платформа використовує галузеві стандарти шифрування для захисту конфіденційності збережених секретів. Це гарантує, що навіть на платформі GitHub лише авторизовані робочі процеси можуть отримати доступ і використовувати секрети під час їх виконання.

GitHub Secrets також відіграє ключову роль у сприянні співпраці в командах розробників. Забезпечуючи централізоване та безпечне сховище для конфіденційної інформації, це оптимізує спільний доступ до робочих процесів і гарантує, що члени команди можуть співпрацювати в процесах автоматизації без шкоди для безпеки.

Таким чином, GitHub Secrets є невід’ємним компонентом стратегії безпеки GitHub Actions. Його здатність безпечно зберігати, керувати та поширювати конфіденційну інформацію в поєднанні з детальним контролем доступу сприяє загальній надійності автоматизованих робочих процесів.

2.2.11 Платформа Docker

Docker, трансформаційна платформа контейнеризації, революціонує процес розгортання програм, інкапсулюючи програми та їхні залежності в портативні контейнери. Ця технологія докорінно змінює підхід до розробки, забезпечуючи узгодженість у різних середовищах і покращуючи масштабованість.

У своїй основі Docker використовує контейнеризацію, легкий і ефективний метод віртуалізації. Контейнери упаковують програму разом із її залежностями та середовищем виконання в самодостатній блок. Це гарантує послідовну роботу програми в різноманітних середовищах, від локальних машин розробки до виробничих серверів.

Архітектура Docker базується на Docker Engine, клієнт-серверній програмі з REST API, яка керує створенням, розгортанням і виконанням контейнерів. Цей движок працює на головній машині, забезпечуючи бездоганний інтерфейс для розробників для створення, доставки та запуску програм за допомогою контейнерів.

Однією з переконливих особливостей Docker є його здатність створювати зображення, які служать кресленнями для контейнерів. Ці зображення інкапсулюють програму, її залежності, системні бібліотеки та конфігурації. Ця узгодженість між середовищами розробки та виробництва пом'якшує сумнозвісну проблему «це працює на моїй машині», підвищуючи надійність у процесі розгортання.

Docker Hub, офіційний реєстр, містить величезне сховище публічних і приватних образів Docker. Це сховище спрощує обмін і розповсюдження попередньо створених образів, дозволяючи розробникам використовувати існуючі рішення та прискорювати процес розробки. Крім того, Docker Compose дозволяє визначати та оркеструвати багатоконтейнерні додатки, полегшуючи складні налаштування та зв'язок між контейнерами.

Вплив Docker на конвеєр CI/CD (безперервна інтеграція/безперервне розгортання) є значним. Інкапсулюючи додатки в контейнери, ми гарантуємо, що те саме середовище, яке використовувалося під час розробки, відображається під час тестування та виробництва. Така узгодженість мінімізує проблеми, пов'язані з розгортанням, прискорює цикл тестування та підвищує загальну надійність.

Ефективність платформи поширюється на використання ресурсів, оскільки контейнери Docker спільно використовують ядро ОС хост-машини. Ця легка природа забезпечує швидке створення контейнерів, ефективне масштабування та оптимальний розподіл ресурсів. Сумісність Docker з такими інструментами оркестровки, як Kubernetes, ще більше покращує його масштабованість, роблячи його ідеальним рішенням для розгортання архітектур мікросервісів.

Безпека є першорядним фактором, і Docker вирішує це, використовуючи

механізми ізоляції контейнерів. Кожен контейнер працює незалежно, обмежуючи потенційну вразливість. Крім того, Docker надає інструменти для сканування зображень, виявлення вразливостей і перевірки відповідності, підвищуючи рівень безпеки контейнерних програм.

Підсумовуючи, Docker змінює ландшафт розробки та розгортання програмного забезпечення, запроваджуючи стандартизований та ефективний підхід. Його технологія контейнеризації покращує узгодженість, масштабованість і безпеку, що робить її наріжним каменем стратегії розгортання програм.

2.2.12 Платформа Heroku

Heroku, хмарна платформа як послуга (PaaS), змінює правила гри, спрощуючи розгортання та керування програмами. Його зручний інтерфейс і зосередженість на продуктивності розробника роблять його невід'ємною частиною пакету розробників.

Ключова привабливість Heroku полягає в простоті використання. За допомогою лише кількох команд розробники можуть розгорнути програми без необхідності вникати в тонкощі управління інфраструктурою. Ця простота прискорює цикл розробки, дозволяючи нам більше зосереджуватися на кодуванні та менше на складнощах розгортання.

Heroku підтримує широкий спектр мов програмування, забезпечуючи гнучкість у стеку технологій. Незалежно від того, чи це Node.js, Python, Ruby чи Java, Heroku задовольняє різноманітні потреби в розробці. Ця універсальність має вирішальне значення, оскільки дозволяє нам вибрати найкращий інструмент для роботи, не будучи обмеженим платформою.

Платформа приймає модель на основі контейнерів, що дозволяє нам пакувати програми та залежності в легкі ізольовані контейнери, відомі як «dynos». Ця контейнеризація не тільки покращує масштабованість, але й забезпечує

узгодженість між середовищами розробки та виробництва, пом'якшуючи можливі проблеми з розгортанням.

Масштабування за допомогою Heroku — легке завдання. Платформа надає параметри як вертикального, так і горизонтального масштабування, що дозволяє нам регулювати ресурси відповідно до вимог програми. Повна інтеграція з ринком доповнень Heroku ще більше розширює функціональність, надаючи послуги, такі як бази даних, кешування та моніторинг, без необхідності налаштування вручну.

Функції постійної доставки Heroku спрощують процес випуску. Завдяки інтеграції з системою контролю версій Heroku Pipeline автоматизує етапи збірки, тестування та розгортання, забезпечуючи безперебійну та послідовну доставку. Ця автоматизація значно знижує ризик людської помилки під час процесу випуску.

Енергійна екосистема платформи включає багатий набір доповнень і розширень, які охоплюють усе: від керування базами даних до моніторингу продуктивності. Ця екосистема розширює можливості програми без потреби в значних налаштуваннях вручну, спрощуючи інтеграцію сторонніх служб.

Heroku надає пріоритет співпраці, надаючи надійні функції керування командою. Кілька членів команди можуть безперешкодно співпрацювати над проектом, маючи контрольований доступ до середовищ і ресурсів. Це середовище співпраці оптимізує зусилля щодо розвитку, сприяючи розвитку культури спільної відповідальності.

Безпека залишається головним пріоритетом для Heroku. Платформа включає різні заходи безпеки, включаючи шифрування даних під час передачі та в спокої, безпечний контроль доступу та регулярні оновлення безпеки. Дотримання Heroku галузевих стандартів безпеки узгоджується з зобов'язанням щодо захисту конфіденційних даних.

Підсумовуючи, поєднання простоти, універсальності, масштабованості та функцій співпраці робить Heroku безцінним активом у наборі інструментів розробки та розгортання. Його зосередженість на продуктивності розробників і безперебійній інтеграції контейнеризації та безперервної доставки ідеально

узгоджується з метою — створення надійних і масштабованих програм.

2.2.13 Веб-інструмент GitHub Workflow

Налаштування робочого процесу безперервної інтеграції/безперервного розгортання (CI/CD) від GitHub до Heroku за допомогою інтерфейсу командного рядка (CLI) Heroku — це потужний спосіб автоматизувати та оптимізувати процес розгортання програм.

GitHub Actions є основою робочого процесу CI/CD. Цей робочий процес, визначений у файлі конфігурації YAML у репозиторії GitHub, може бути ініційований такими подіями, як надсилання коду, запити на отримання або випуски. GitHub Actions дозволяє нам визначати різні кроки, включаючи створення та тестування коду програми.

Інтеграція Heroku CLI в робочий процес дозволяє нам легко розгортати програму на платформі Heroku. Heroku CLI, інструмент командного рядка, полегшує взаємодію зі службами Heroku та автоматизує завдання розгортання.

Для безпечної взаємодії між GitHub Actions і Heroku ми використовуємо ключі Heroku API. Ці ключі зберігаються як секрети GitHub, що забезпечує конфіденційність конфіденційної інформації. Потім GitHub Actions може отримати доступ до цих секретів як змінних середовища під час виконання робочого циклу, дозволяючи Heroku CLI автентифікувати та розгортати програму у вказаній програмі Heroku.

Файл конфігурації робочого циклу містить не лише кроки для встановлення залежностей, виконання тестів і створення програми, але також інтегрує команди Heroku CLI для розгортання. Це забезпечує комплексний та автоматизований підхід до всього процесу CI/CD.

Коли зміни надсилаються до репозиторію GitHub, робочий процес CI/CD запускається автоматично. GitHub Actions ініціює визначені кроки, а Heroku CLI,

автентифікований за допомогою ключа API, взаємодіє з платформою Heroku для розгортання програми в зазначеному середовищі.

Переваги цього робочого процесу від GitHub до Heroku значні. Він автоматизує процес розгортання, зменшуючи потребу в ручному втручанні та мінімізуючи ризик помилок. Узгодженість між середовищами розробки та виробництва забезпечується, оскільки весь робочий процес дотримується. Крім того, покращено аспект співпраці, що забезпечує стандартизований і автоматизований підхід для всієї команди.

Таким чином, робочий процес GitHub до Heroku CI/CD з використанням Heroku CLI пропонує спрощене та автоматизоване рішення для розгортання програм. Ця інтеграція оптимізує конвеєр розробки, покращує співпрацю та забезпечує надійне та послідовне розгортання на платформі Heroku.

2.3 Архітектура MVVM

Архітектурний шаблон MVVM (Model-View-ViewModel) — це елегантний і ефективний вибір для розробки веб-додатків, які вимагають чіткого розподілу завдань, зручності обслуговування та тестування. Він організовує даний проект на три ключові компоненти, що описані далі та зображено на рисунку далі (Рис. 2.1).

1. **Модель:** модель представляє дані програми, бізнес-логіку та основні функції даної веб-програми. Він інкапсулює структури даних, взаємодію з базою даних, обробку даних та будь-які інші завдання, пов'язані з серверною частиною. У контексті даного проекту Модель буде відповідати за керування даними про аспірантів, курси, зарахування та інші аспекти діяльності аспірантського офісу.
2. **Перегляд:** перегляд — це компонент інтерфейсу користувача (UI) даної

програми. Він відповідає за відображення даних користувачам і фіксацію взаємодії користувачів. У шаблоні MVVM View зберігається максимально простим і позбавленим прикладної логіки. Він має в першу чергу зосереджуватися на візуалізації даних і обробці введених користувачем даних. Цей компонент даного проекту буде відповідати за створення інтерфейсу користувача, з яким взаємодіють співробітники офісу аспірантів, викладачі та студенти.

3. ViewModel: ViewModel діє як посередник між моделлю та представленням. Він служить мостом, який надає дані в View і обробляє взаємодії користувача, а потім передає ці взаємодії в Model для обробки. Він також включає бізнес-логіку програми. У даному проекті ViewModel подбає про керування даними, пов'язаними з аспірантами, курсами та іншими науковими справами. Він також оброблятиме такі операції, як реєстрація, академічні консультації та планування курсів.

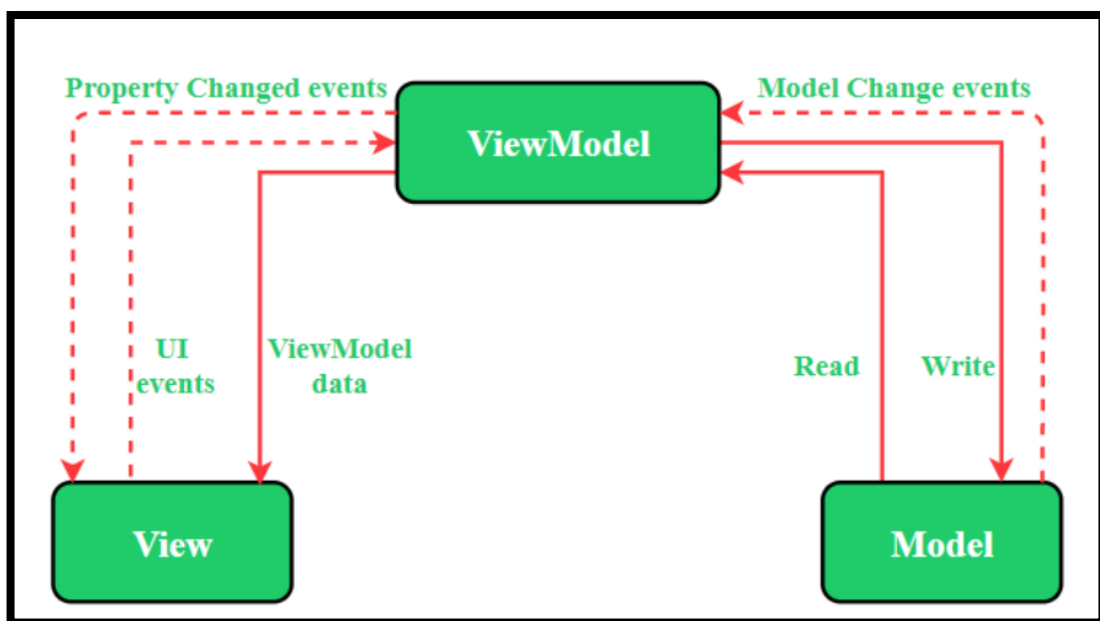


Рисунок 2.1 – Шаблон архітектури MVVM (Model View ViewModel)

Далі представлено як архітектура MVVM може принести користь даному проекту.

1. Поділ проблем: MVVM забезпечує чітке розмежування між інтерфейсом

користувача, даними програми та бізнес-логікою, а також взаємодією користувача. Таке розділення підвищує зручність обслуговування даної кодової бази та полегшує розширення та зміну програми.

2. Можливість тестування: MVVM сприяє тестуванню даної програми. Ви можете написати модульні тести для ViewModel, щоб гарантувати правильне функціонування бізнес-логіки. Крім того, оскільки View залишається простим, тестування інтерфейсу користувача стає більш простим.
3. Гнучкість: шаблон MVVM забезпечує гнучкість інтерфейсу користувача. Ви можете мати кілька представлень, які взаємодіють з тією самою ViewModel, що робить дану програму адаптованою до різних платформ і пристроїв.
4. Прив'язка даних: MVVM часто включає зв'язування даних, що спрощує процес оновлення інтерфейсу користувача при зміні основних даних. Це особливо корисно для програм, де потрібне оновлення даних у реальному часі або динамічне.

MVVM є цінним вибором, оскільки потрібна добре організована кодова база, яку можна тестувати та підтримувати. Це допоможе розділити зовнішню та серверну логіку, полегшуючи керування обома частинами програми та розширюючи їх. Він також узгоджується з принципами розробки сучасних веб-додатків і добре підходить для програм, які передбачають керування даними, взаємодію з користувачем і представлення академічної інформації аспірантам, викладачам і співробітникам.

Висновки до розділу 2

Підсумовуючи, впровадження та побудова розробленої системи підкреслюють стратегічну інтеграцію передових технологій і продуманий вибір

інструментів. Використовуючи можливості JavaScript, Node.js і Nest.js, ми створили надійну серверну архітектуру. GitHub Actions, GitHub Secrets і Docker утворюють єдиний конвеєр CI/CD, автоматизуючи робочі процеси розробки та забезпечуючи спрощений процес розгортання.

Прийняття Heroku як хмарної платформи забезпечує масштабоване та ефективне рішення для хостингу, а PostgreSQL, як система керування базами даних, підтримує зберігання та пошук даних із надійністю та продуктивністю.

Серверна архітектура, побудована на Node.js і Nest.js, втілює модульність і зручність обслуговування. Express.js забезпечує ефективне керування проміжним програмним забезпеченням і маршрутизацію, що сприяє загальній швидкості реагування сервера.

Робочий процес CI/CD, організований через GitHub Actions і GitHub Secrets, забезпечує безпечний і ефективний конвеєр розробки. Контейнеризація Docker покращує послідовність і масштабованість, а модель Heroku PaaS спрощує розгортання.

Під час створення системи дотримувалися принципів безпеки, масштабованості та ефективності. Адаптивність і стійкість сервера підсилюються механізмами балансування навантаження, що забезпечує оптимальну продуктивність навіть при коливаннях трафіку користувачів.

Отже, впровадження та побудова розробленої системи є прикладом гармонійної інтеграції технологій, сприяючи створенню надійної, масштабованої та безпечної основи для веб-додатку, який обслуговує офіс аспіранта. Таке стратегічне узгодження забезпечує не тільки поточну надійність системи, але й сприятливе положення для майбутніх удосконалень і адаптацій..

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Реалізація програмного забезпечення для веб-додатку в офісі аспірантів передбачає стратегічне поєднання технологій. GitHub Actions відіграє центральну роль в організації конвеєра CI/CD, бездоганно інтегруючись із робочим процесом GitHub, визначеним у файлах YAML. Ця автоматизація оптимізує завдання від компіляції коду до розгортання.

Безпека віддається пріоритету завдяки використанню секретів GitHub, які захищають конфіденційну інформацію, як-от ключі API та маркери доступу. Ці секрети надійно зберігаються, що гарантує, що лише авторизовані робочі процеси можуть отримати до них доступ і використовувати їх.

Файли YAML служать чіткими та читабельними сценаріями конфігурації для робочих процесів GitHub Actions, забезпечуючи прозорість і зручність обслуговування. Вони діють як креслення для автоматизованих процесів.

Docker і Dockerfiles є ключовими для контейнеризації програми та її залежностей. Це не тільки забезпечує узгодженість у різних середовищах, але й полегшує масштабованість і пом'якшує проблеми з розгортанням. Docker Compose додатково допомагає безперешкодно оркеструвати багатоконтейнерні програми.

Heroku служить обраною нами хмарною платформою для розгортання веб-програми. Використовуючи простоту моделі Heroku PaaS і Heroku CLI, процес розгортання оптимізований, пропонуючи масштабованість і просте керування.

PostgreSQL використовується як система керування реляційною базою даних, що забезпечує надійні можливості керування даними. Його розширені функції відповідають складним вимогам обробки даних студентів і академічних записів.

Принципи безперервної інтеграції та безперервного розгортання (CI/CD) є частиною філософії розробки. CI забезпечує регулярну інтеграцію, тестування та перевірку змін коду, тоді як CD автоматизує процес випуску, забезпечуючи швидке та надійне розгортання.

Організація GitHub централізовано структурує репозиторії, покращуючи співпрацю та забезпечуючи узгодженість у всій організації.

Підсумовуючи, реалізація програмного забезпечення є гармонійною інтеграцією дій GitHub, секретів GitHub, файлів YAML, Docker, Heroku, PostgreSQL, принципів CI/CD та організації GitHub. Кожен елемент відіграє вирішальну роль у формуванні ефективності, безпеки та масштабованості веб-програми для офісу аспірантів.

3.1 Серверна частина програмного забезпечення

Серверна частина розробленої системи служить основою, організовуючи потік даних, керуючи запитами та забезпечуючи безперебійну роботу веб-програми. Його архітектура та компоненти відіграють ключову роль у формуванні продуктивності, масштабованості та функціональності всієї системи.

Node.js, як основне середовище виконання, надає серверу можливість ефективно обробляти численні одночасні підключення. Його керована подіями архітектура робить його особливо придатним для додатків із високими вимогами до введення/виведення, забезпечуючи адаптивну та масштабовану основу.

Nest.js, обраний фреймворк для структурування серверного коду, надає безліч функцій. Його модульна архітектура дозволяє нам організовувати кодову базу на компоненти, які можна багаторазово використовувати. Використання ін'єкції залежностей покращує ремонтпридатність серверної логіки, спрощуючи керування та оновлення в міру розвитку програми з часом. Nest.js, створений за допомогою TypeScript, представляє надійну типізацію, яка додає додатковий рівень передбачуваності та надійності реалізації на стороні сервера.

Інтеграція Swagger у сервер є стратегічним кроком до вдосконалення аспекту документації API. За допомогою Swagger сервер автоматично створює вичерпну документацію API. Ця документація є цінним ресурсом для розробників, надаючи

чітке уявлення про доступні кінцеві точки, очікувані формати запитів/відповідей і будь-яку додаткову інформацію, необхідну для ефективного використання API.

У ширшому контексті серверної частини міркування щодо масштабованості та продуктивності є найважливішими. Архітектура розроблена з урахуванням зростання трафіку користувачів і обсягу даних. Механізми балансування навантаження, стратегії кешування та оптимізації на рівні сервера сприяють створенню надійної та чутливої серверної інфраструктури.

Крім того, серверна частина взаємодіє з базою даних PostgreSQL, керуючи зберіганням і пошуком даних. Ця інтеграція забезпечує послідовність даних, надійність і ефективну обробку складних запитів. Роль сервера в управлінні даними виходить за рамки простих транзакцій, включаючи стратегії цілісності даних, безпеки та дотримання найкращих галузевих практик.

Структура серверної частини системи представлена на рисунку 3.1.

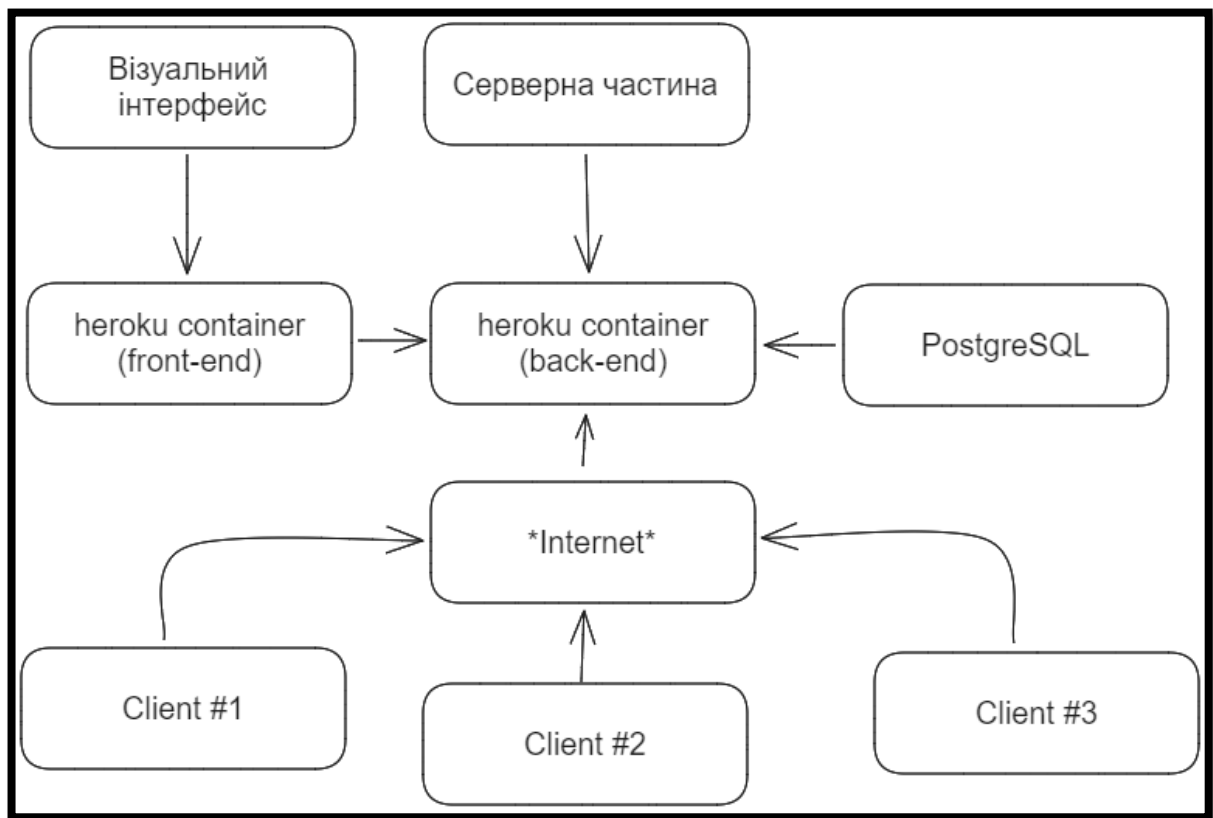


Рисунок 3.1 — Структура серверної частини розробленої системи

Структурний склад серверної частини системи відображає продуманий підхід до управління складнощами розробки веб-додатків. Розглянемо відмінні компоненти, які формують архітектуру сервера.

1. Архітектура, керована подіями:

Архітектура сервера розроблена за парадигмою, керованою подіями, де компоненти реагують на певні події та запускають відповідні дії. Такий підхід сприяє слабкому зв'язку між різними частинами системи, забезпечуючи кращу масштабованість і оперативність. Події можуть варіюватися від дій користувача до системних подій, таких як оновлення даних.

2. Стратегії кешування для оптимізації продуктивності:

Впровадження ефективних стратегій кешування є невід'ємною частиною структури сервера. Стратегічно кешуючи дані або обчислення, до яких часто звертаються, сервер оптимізує час відповіді та зменшує навантаження на основні

ресурси. Цей структурний елемент є особливо корисним для підвищення загальної продуктивності програми.

3. Комплексна реєстрація та моніторинг:

Структура сервера включає надійні механізми журналювання та моніторингу. Вичерпні журнали дають уявлення про поведінку системи, допомагають у налагодженні та вирішенні проблем. Інструменти моніторингу відстежують показники продуктивності, забезпечуючи проактивне виявлення потенційних вузьких місць або аномалій і сприяючи загальній надійності сервера.

Таким чином, унікальні структурні елементи, включені в серверну частину системи, такі як архітектура сервісів, дизайн, керований подіями, стратегії кешування та механізми журналювання та моніторингу, разом сприяють стійкості, масштабована та високопродуктивна основа для веб-програми.

На наступному рисунку 3.2, зображено ролі, які існують в базі даних та їх можливості.

Роль	Можливості
Викладач	Перегляд профілю аспірантів, які вивчаються під їх керівництвом, перегляд академічних досягнень своїх аспірантів, виставлення та оцінювання завдань та лекцій для аспірантів, обговорення та обмін повідомленнями з аспірантами, завантаження та редагування навчального матеріалу.
Аспірант	Перегляд свого профілю та особистої інформації, перегляд розкладу та завдань, які надаються викладачами, відправлення завдань та лекцій для оцінки викладачам, обговорення та обмін повідомленнями з викладачами та іншими аспірантами, доступ до бібліотеки навчального матеріалу.
Завідувач кафедри	Моніторинг академічної активності викладачів та аспірантів на кафедрі, затвердження курсів та навчальних планів.
Адміністратор	Управління доступом та ролями користувачів в системі, моніторинг та підтримка технічного функціонування системи, забезпечення безпеки та конфіденційності даних, резервне копіювання та відновлення даних, вирішення технічних питань та надання підтримки користувачам.

Рисунок 3.2 — Ролі та можливості користувачів веб-додатку

Будучи центральним центром програми, серверна частина ретельно розроблена, щоб відповідати не лише поточним вимогам, але й безперешкодно адаптуватися до майбутніх потреб. Його архітектура, фреймворки та інтеграція разом сприяють надійності, продуктивності та розширюваності всієї системи.

Таким чином, серверна частина - це не просто технічна складова; це механізм, який просуває веб-програму вперед, забезпечуючи оперативну та надійну взаємодію з користувачем.

3.2 Клієнтська частина програмного забезпечення

Клієнтська частина програмного забезпечення є ключовим елементом, який безпосередньо взаємодіє з користувачами, формуючи їхній досвід у веб-додатку, розробленому для кабінету аспірантів.

JavaScript служить основною мовою сценаріїв для реалізації логіки на стороні клієнта. Його динамічні можливості дозволяють створювати динамічні та інтерактивні функції, які покращують залучення користувачів. У тандемі з JavaScript можна інтегрувати TypeScript, надмножину JavaScript, щоб запровадити статичний тип. Це підвищує надійність коду та допомагає виявляти можливі помилки під час розробки.

Інтерфейс додатку створено з використанням сучасної та універсальної системи, такої як React.js або Vue.js. Ці фреймворки забезпечують структурований підхід до розробки інтерфейсів користувача, дозволяючи створювати динамічні компоненти, які плавно реагують на взаємодії користувача.

Управління станом є критично важливим аспектом клієнтської частини, що забезпечує узгоджений і чуйний інтерфейс користувача. Такі методи, як Redux або Context API у React.js, використовуються для ефективного керування та оновлення стану додатка, забезпечуючи плавну та безперебійну роботу користувача.

Для того, щоб веб-додаток витончено адаптувався до різних розмірів екрана, застосовано адаптивний підхід до дизайну, орієнтований на мобільні пристрої. Цей адаптивний дизайн забезпечує оптимальний перегляд і взаємодію на різних пристроях, від настільних ПК до планшетів і смартфонів.

Інтерфейс користувача розроблено з акцентом на надійні принципи UI/UX.

Інтуїтивно зрозуміла навігація, узгоджені елементи дизайну та доступність створюють привабливий і зручний інтерфейс.

Безперебійна взаємодія з сервером досягається завдяки чітко визначеним API. Кінцеві точки API RESTful сприяють ефективному обміну даними між клієнтом і сервером, забезпечуючи динамічне оновлення вмісту без необхідності повного перезавантаження сторінки.

Механізми маршрутизації на стороні клієнта, реалізовані за допомогою таких бібліотек, як React Router або Vue Router, забезпечують плавну навігацію в додатку. Користувачі можуть динамічно перемикатися між різними видами та компонентами, покращуючи загальну взаємодію з користувачем.

Стратегії комплексного тестування інтегровані в клієнтську частину для забезпечення надійності. Методи модульного тестування, інтеграційного тестування та наскрізного тестування використовуються для виявлення та вирішення потенційних проблем на ранніх стадіях процесу розробки, забезпечуючи високоякісну кодову базу на стороні клієнта.

Отже, клієнтська частина програмного забезпечення ретельно розроблена, щоб забезпечити безперебійну, адаптивну та орієнтовану на користувача роботу веб-програми, розробленої для офісу аспірантів..

Розглянемо Use-Case діаграму для клієнтської частини додатку, яка має наступний вигляд, що представлена на рисунку 3.3.

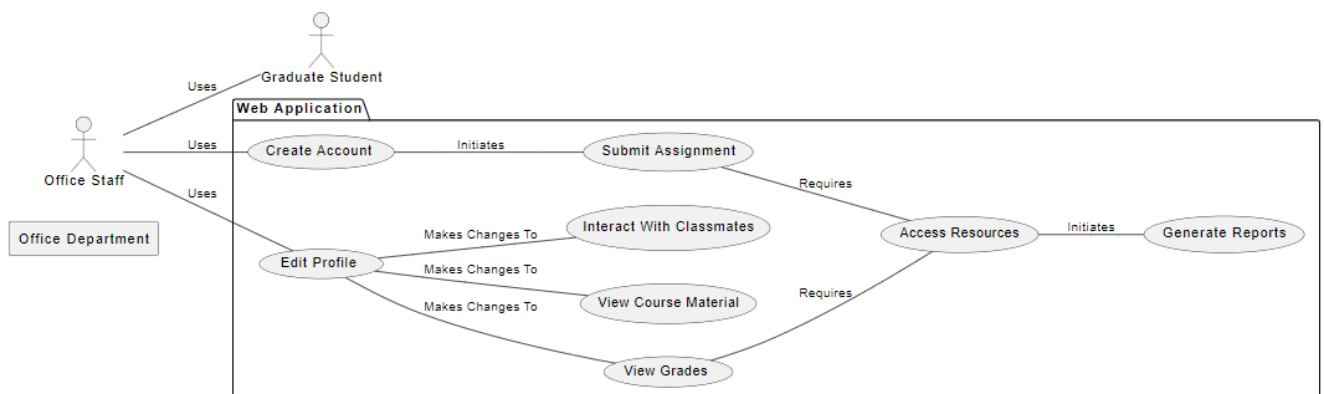


Рисунок 3.3 — Use-Case діаграма

На наступному рисунку 3.4 зображено розроблена схема організації спільної роботи.

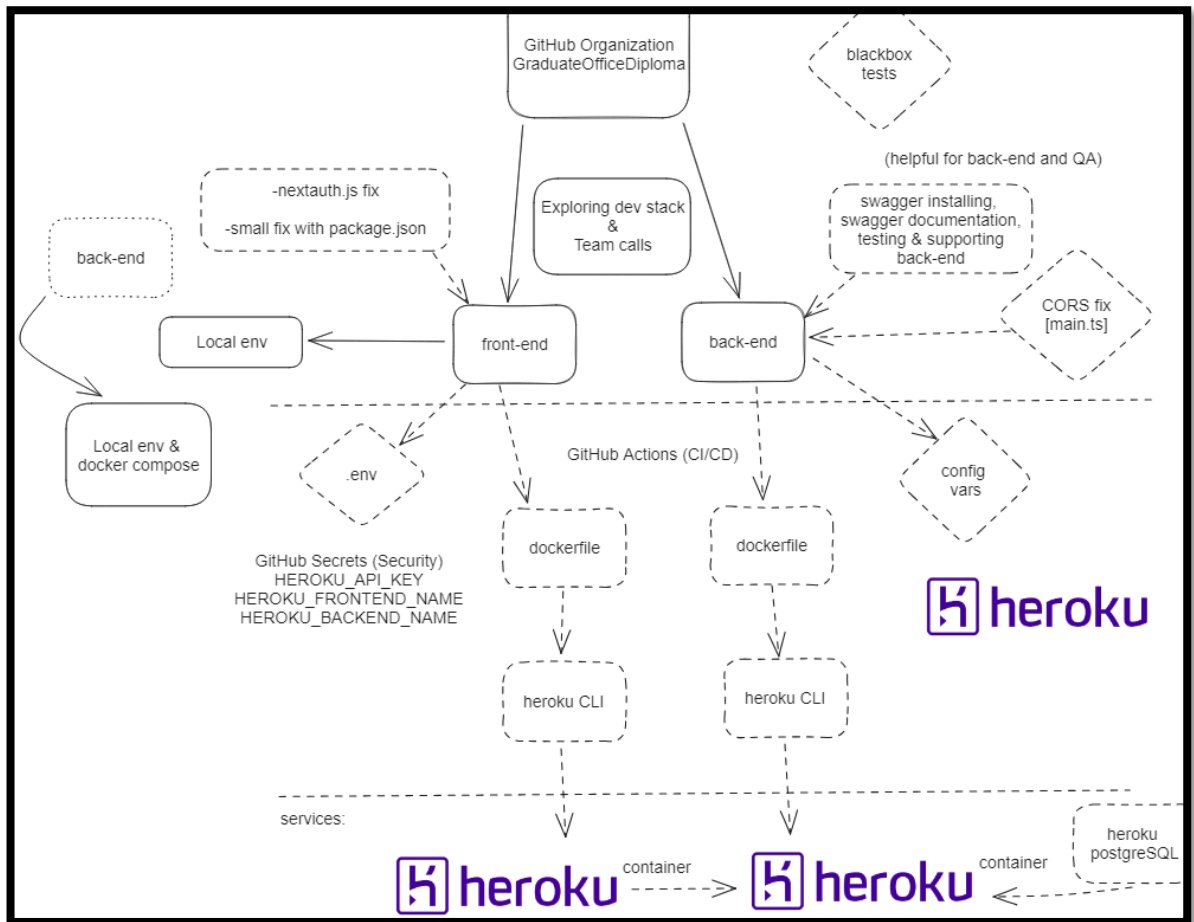


Рисунок 3.4 — Схема організації спільної роботи команди

На наступному рисунку 3.5 зображено пояснювальну легенду для рисунку 3.4. Вона додає ясності у взаємодії команди на проекті.

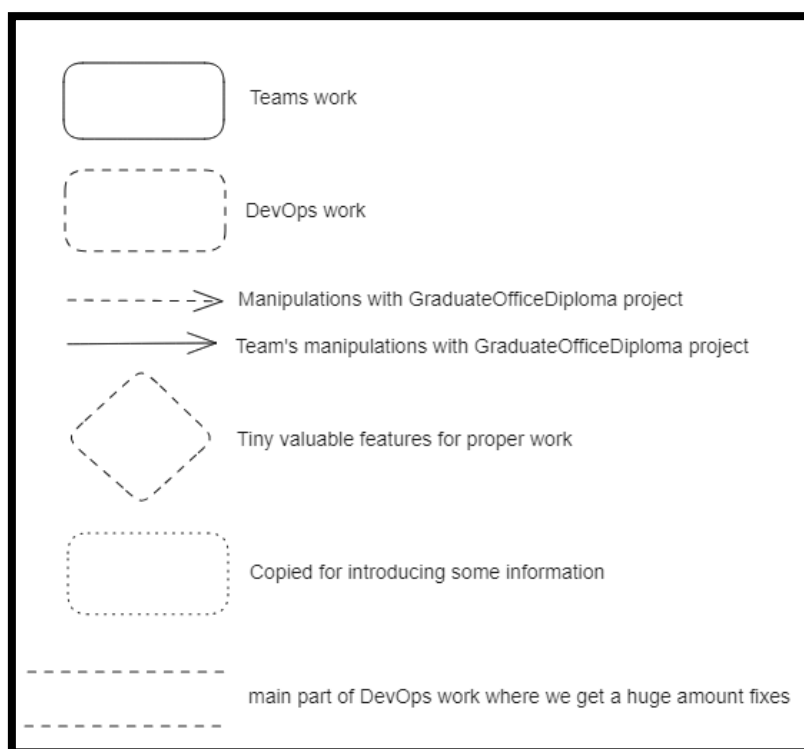


Рисунок 3.5 — Легенда

Уявіть цифровий інтерфейс, спроектований спеціально для аспірантів, де дедлайни відступають на другий план, а співпраця відбувається без труднощів. Це клієнтська частина веб-програми, портал, призначений перетворити академічний досвід.

Цей інтерфейс обладнаний інтуїтивно зрозумілими вкладками, бічними панелями, які вказують напрямок, і спадними меню, що розгортаються як корисні карти. Йдеться про персональний органайзер, де кожен розділ — це окремий простір для управління дедлайнами, взаємодії з наставниками чи відстеження прогресу досліджень.

Співпраця вигідніша за ланцюжки електронних листів і телефонних дзвінків. Уявіть обговорення дослідницьких ідей через відеоконференції чи розбір чернеток дисертацій у реальному часі з наставником, все це в межах програми. Робочі простори проектів перетворюються на віртуальні кімнати, де спільні документи, графіки та матеріали пульсують енергією командної роботи. Оновлення в реальному часі та контроль версій гарантують єдність робочих процесів, уникнення

непорозуміннь.

Проте цей портал не лише оптимізує завдання, а й сприяє активній спільноті. Форуми і дискусійні дошки наповнюються діалогами, а гейміфіковані системи винагород підтримують дружельобну конкуренцію. Роздуми в спільній боротьбі, мозковий штурм та налагодження зв'язків по всьому світу - все це можливо.

Цей портал також спрощує керування цитатами. Зручні посилання, інтеграція з інструментами керування цитуванням та автоматичне форматування бібліографії полегшують науковий процес. Дослідження зберігаються у хмарі, організовані та доступні для пошуку, як сади цифрової інформації. Інструменти розширеного пошуку дозволяють швидко знаходити потрібну інформацію. Дослідження стали симфонією, готовою вийти на волю світу.

І, нарешті, дані, колись холодні й непохитні, перетворюються на історії. Інтерактивні візуальні елементи взаємодіють на екрані, діаграми відтворюють тенденції, а карти створюють візерунки в приголомшливих деталях. Більше не потрібно мучитися з електронними таблицями, дозвольте даним самим розповісти свою історію, розкриваючи та освітлюючи шлях до новаторських відкриттів.

Висновки до розділу 3

Розділ "Опис програмної реалізації" системи включає ретельний аналіз використаних технологій та їх взаємодію для створення комплексного рішення. JavaScript та Node.js виступають основною основою розробки, забезпечуючи високу продуктивність та широкі можливості. Nest.js виступає серверним фреймворком, який сприяє створенню масштабованих та легко підтримуваних додатків.

Swagger використовується для автоматизації процесу документування API, що полегшує розробку та взаємодію з системою. GitHub Actions та Heroku впроваджені для автоматизації процесу CI/CD, забезпечуючи швидке тестування та

розгортання. Docker використовується для упаковки додатку та його залежностей у контейнер, забезпечуючи консистентність серед різних середовищ.

Мікросервісна архітектура взята за основу для полегшення масштабування та гнучкості системи. Важливий акцент зроблено на забезпеченні безпеки даних та стійкості серверної частини, використовуючи засоби балансування навантаження для оптимізації продуктивності.

Загалом, програмна реалізація системи враховує низку технічних аспектів для створення ефективного та надійного рішення, спрямованого на покращення університетських проектів та активне навчання студентів.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

В даному розділі розглядається процес взаємодії користувачів із розробленою програмною системою. Робота користувача з системою є важливим етапом, оскільки від неї залежить зручність та ефективність використання всіх функцій та можливостей, які пропонує цей інноваційний продукт. Розділ ретельно описує інтерфейс, доступні функції, а також способи взаємодії користувачів із системою для досягнення їхніх конкретних цілей та завдань. Проаналізовано важливі аспекти, такі як навігація, зручність використання та можливості налаштувань для кожного типу користувача. Розділ розкриває важливі аспекти користувацького досвіду та надає інсайти щодо того, як система сприймається та використовується користувачами в реальних умовах.

4.1 Інсталяція програмного забезпечення

Розпочати академічний огляд за допомогою цього програмного забезпечення легко — лише один клік. На відміну від традиційних завантажень і інсталяцій, система використовує хмарні технології та зберігається в контейнерах Heroku, з доступом з будь-якого пристрою, що має підключення до Інтернету.

Частина 1: Клієнтська частина

Heroku Harmony: перейдіть за посиланням Heroku, щоб отримати доступ до інтерфейсу клієнта. Простий клік переносить вас до особистого академічного командного центру, без необхідності завантаження чи встановлення. Додайте це посилання в закладки, і шлях до підвищення ефективності завжди буде доступним одним кліком.

Частина 2: Серверна частина

Heroku Encore: доступ до другого посилання Heroku для внутрішньої інфраструктури. Так само, як і клієнт, одним клацанням миші ви отримуєте безпечний доступ до прихованих функцій системи, забезпечуючи цілісність даних і безперебійну роботу.

Частина 3: Увертюра PostgreSQL

Heroku Encore (додатково): надана база даних PostgreSQL від Heroku бездоганно інтегрується з клієнтськими та серверними компонентами. Додаткова конфігурація не потрібна; база даних працює в ідеальній гармонії з рештою системи.

4.2 Вимоги до обчислюваної техніки

Щоб забезпечити безперебійну роботу цього програмного забезпечення, уважно вивчіть технічні характеристики клієнтських і серверних компонентів. Ці вимоги діють як гармонійний оркестр, забезпечуючи бездоганне відтворення кожної ноти та плавне виконання кожного навчального завдання.

Вимоги зі сторони Клієнту:

Веб-браузер: Шлюзом до інтерфейсу клієнта є сучасний веб-браузер, наприклад Chrome, Firefox, Edge або Safari. Ці авторитетні веб-переглядачі, якщо вони оновлюються, гарантують безперебійну та швидку роботу.

З'єднання з Інтернетом: стабільне з'єднання з Інтернетом служить порятунком для обміну даними між пристроєм і хмарним програмним забезпеченням. Настійно рекомендується мінімальна швидкість завантаження 10 Мбіт/с, щоб забезпечити оптимальний потік даних і безперебійну роботу користувача.

Операційні системи. Незважаючи на те, що програмне забезпечення розроблено для ефективної роботи в різних операційних системах, Windows 10 або новішої версії, macOS Catalina або новішої версії та ChromeOS рекомендуються для

найбільш ефективної та оптимізованої роботи.

Серверна частина:

Платформа хостингу: Heroku, хмарна платформа, забезпечує безпечну та масштабовану сцену для роботи сервера. Це усуває потребу в локальних установках або обслуговуванні сервера, дозволяючи вам зосередитися виключно на своїх академічних заняттях.

Пам'ять (ОЗП): щоб впоратися зі складним танцем зберігання даних, обробки та взаємодії з користувачем, серверу потрібен мінімум 1 ГБ оперативної пам'яті. Однак для великих дослідницьких проєктів, розширених функцій співпраці або очікуваного зростання бази користувачів наполегливо рекомендується 2 ГБ або більше для забезпечення оптимальної продуктивності.

Дисковий простір: серверу потрібен достатній дисковий простір для зберігання даних досліджень, файлів користувача та компонентів системи. Хоча потрібно щонайменше 500 МБ, більш масштабні дослідницькі проєкти або очікуване зростання можуть вимагати додаткового сховища для забезпечення безперебійного потоку інформації.

Посилення академічного досвіду: хоча це не є обов'язковим, наведені нижче вдосконалення можуть додатково збагатити програмне забезпечення, створивши більш захоплююче та продуктивне навчальне середовище:

Другий монітор: другий монітор пропонує перевагу ширшого робочого простору, дозволяючи переглядати дослідницькі статті та інші матеріали поряд з інтерфейсом клієнта. Це візуальне розширення може підвищити продуктивність і покращити концентрацію, сприяючи більш ефективному робочому процесу.

Навушники: зручні навушники можуть блокувати відволікання та занурити вас у процес дослідження, створюючи зосереджену атмосферу, яка дозволяє вам повністю задіяти функції програмного забезпечення та глибше заглибитися у свої навчальні заняття.

Пам'ятайте, що ці рекомендації ретельно розроблені, щоб забезпечити оптимальну продуктивність і безперебійну роботу користувача. Дотримуючись або

перевищуючи ці технічні вимоги, ви створите гармонійне середовище для процвітання програмного забезпечення, що дозволить вам досягти академічних успіхів з ефективністю та легкістю.

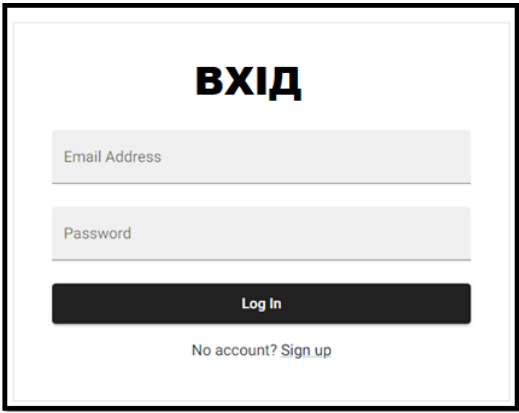
4.3 Демонстрація функціоналу програмного забезпечення

В цьому підрозділі ми введемо на передній план реальний функціонал розробленого програмного забезпечення, представляючи його в деталях та надаючи конкретні приклади його використання. Здійснимо глибокий розгляд ключових можливостей системи, розкриваючи їхню значущість для користувача та їх взаємодію в межах загальної функціональності.

Під час демонстрації функціоналу, акцент буде зроблено на практичному використанні програми в реальних умовах. Виокремимо основні опції, які полегшують роботу користувачів, а також представимо різноманітні сценарії використання, щоб продемонструвати універсальність та ефективність системи.

Цей підрозділ спрямований на введення читача у конкретний функціонал програми, надаючи повний обсяг можливостей та переваг.

Аспіранти отримують доступ до системи через облікові дані для входу, надані університетом (Рис 4.1).



The image shows a login screen with a white background and a black border. At the top, the word "ВХІД" (Login) is displayed in bold black text. Below it, there are two input fields: "Email Address" and "Password", both with light gray borders. Under the password field is a black button with the text "Log In" in white. At the bottom, there is a link that says "No account? Sign up" in a small, light gray font.

Рисунок 4.1 — Аутентифікація(Початковий екран)

Також є можливість реєстрації користувача – але користувач отримає лише роль Гостя, а для того, щоб користуватися веб-додатком – потрібно, щоб системний адміністратор видав певну роль – студента, або викладача в базі даних. Реєстрація користувача зображено на наступну рисунку (Рис.4.2).

The image shows a registration form with the title "РЕЄСТРАЦІЯ" in bold. It contains three input fields: "Full name", "Email Address", and "Password". Below these fields is a dark button labeled "Create account". At the bottom, there is a link that says "Have an account? Log in".

Рисунок 4.2 — Реєстрація

Далі буде представлення успішного розгортання проекту зі сторони аспіранта, після аутентифікації. На наступному рисунку 4.3 зображено Мої курси зі сторони аспіранта.

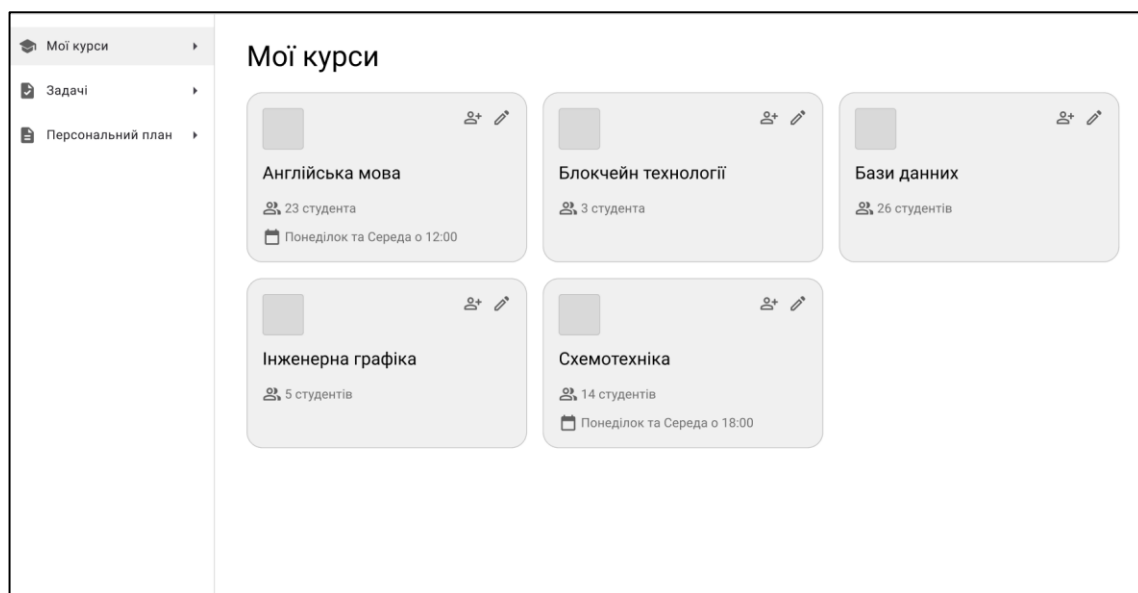


Рисунок 4.3 — Головна сторінка, мої курси зі сторони аспіранта

Після цього перевіримо будь-який курс, наприклад, Англійська мова, що буде зображено на рисунку 4.4.

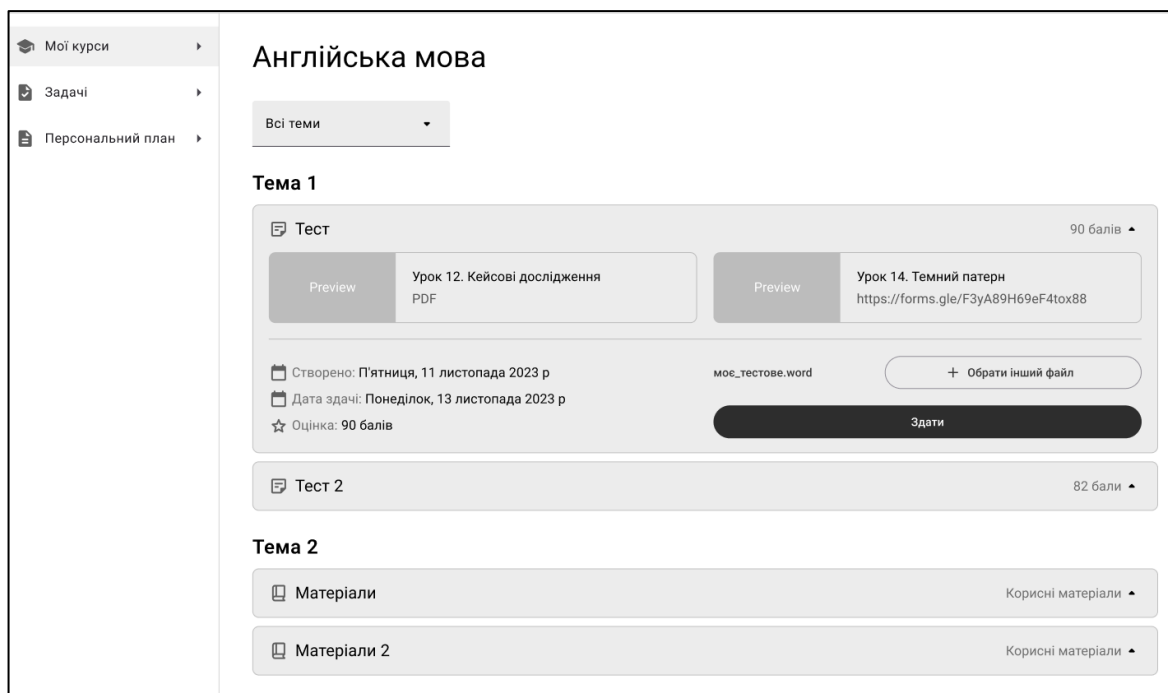


Рисунок 4.4 — Огляд курсу англійської мови та завдань з даного курсу

Також, як можна помітити є розділ Персональний план, його буде зображено на наступному рисунку 4.5.

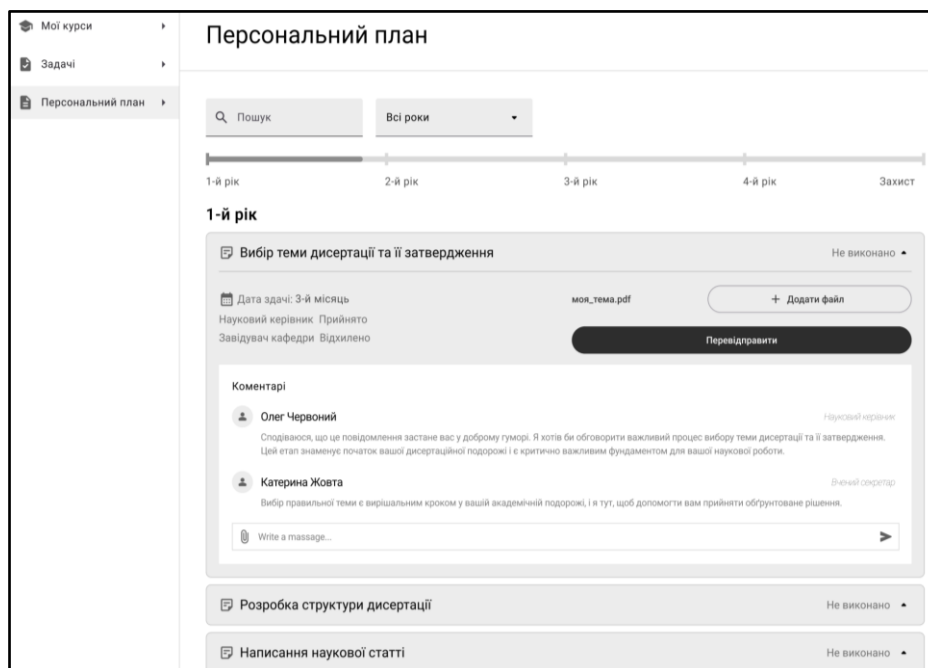


Рисунок 4.5 — Персональний план аспіранта

Розглянемо далі ідентичні сторінки, але з боку викладача. На рисунку 4.6

буде зображено огляд курсу зі сторони викладача курсу.

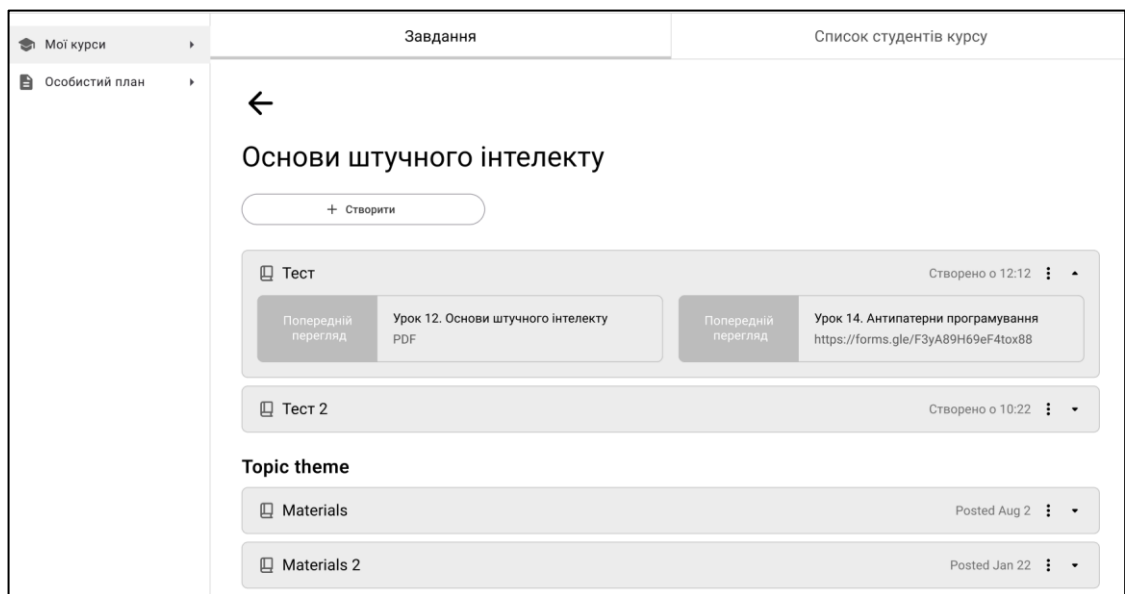


Рисунок 4.6 — Огляд курсу з боку викладача

Наступною сторінкою розглянемо сторінку списку студентів курсу, яка представлена на рисунку 4.7.

На сторінці відображено основні дані користувача, такі як ім'я, прізвище, аудит студента (здані роботи, оцінки, дати виконання робіт, при настиканні кінцевої іконки).

Також є можливість додавання нового аспіранту до обраного курсу.

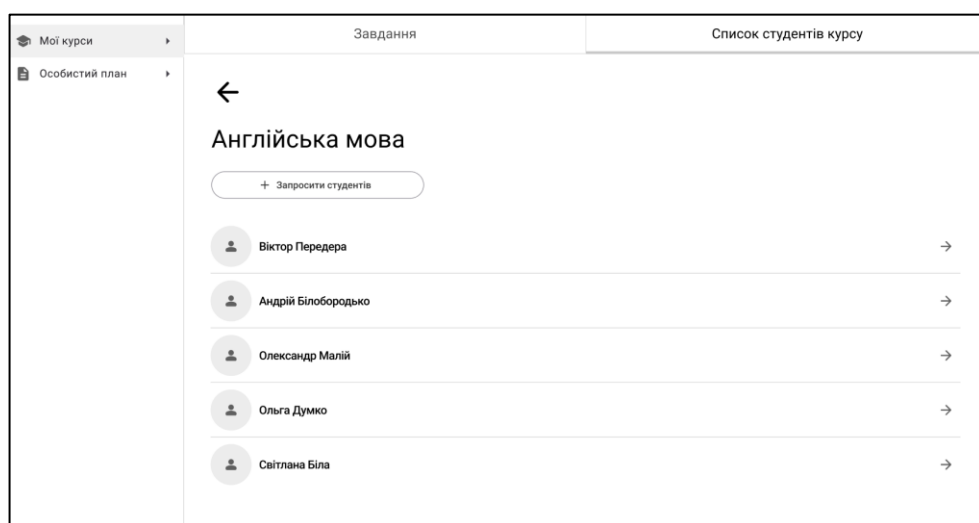


Рисунок 4.7 — Список студентів курсу

Слідуюча сторінка представляє подробиці про аспірантів, які завдання вони виконали, наскільки балів, тощо, дану сторінку зображено на рисунку 4.8.

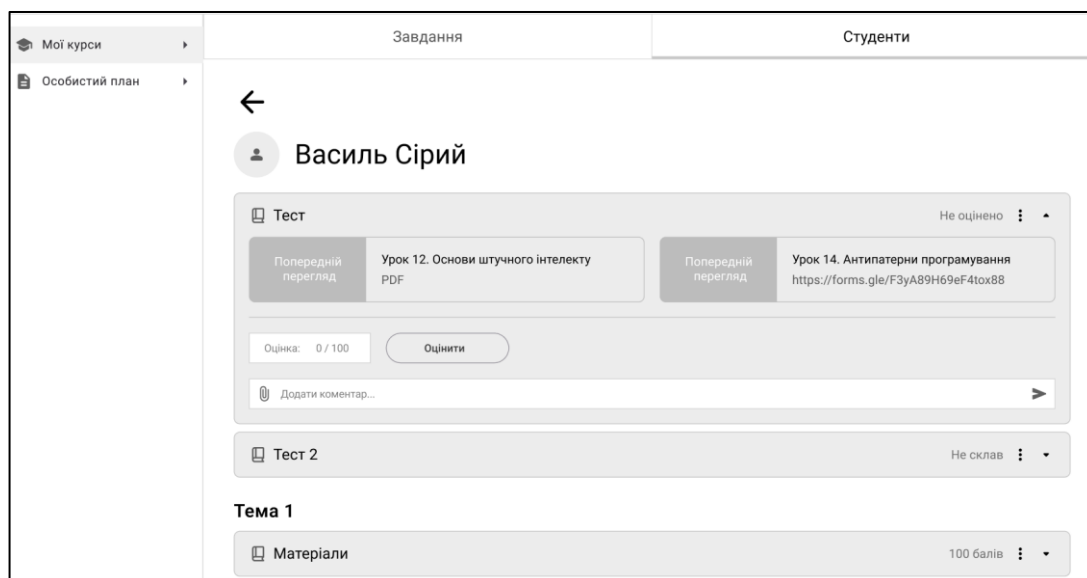


Рисунок 4.8 — Сторінка редагування профілю

Розглянемо сторінку персональних планів, що представлена на рисунку 4.9. Сторінка персональних планів — це інтерфейс, який надає можливість комунікації керівника персонального плану та аспіранта узгоджувати та виконувати завдання в якомога кращий спосіб.

З точки зору викладача під час навігації за особистим планом аспіранта у веб-додатку, інтерфейс розроблено таким чином, щоб запропонувати цілісне уявлення про навчальний шлях і прогрес студента.

Отримавши доступ до персонального плану, викладач стикається з важливою інформацією про студента, як-от ім'я, ідентифікатор та контактна інформація. Також представлено академічну специфіку, включаючи навчальну програму, спеціалізацію та передбачувану дату випуску.

Інтерфейс розгортається з уявленнями про академічну траєкторію студента. Це включає огляд завершених і поточних курсів разом із відповідними оцінками. Сукупний середній бал і значні академічні досягнення демонструються на помітному місці.

Основою інтерфейсу полягає в індивідуальному плані навчання аспіранта.

Тут викладач заглиблюється в детальну розбивку поточних записів, майбутніх курсів і запропонованих майбутніх семестрів. Деталі курсу, цілі та очікувані результати навчання студента легко доступні.

Для студентів, які займаються дослідженнями або проектами, викладач отримує інформацію за допомогою підсумків або оновлених відомостей про їхній прогрес. Дослідницькі статті, проектна документація або відповідні посилання можуть бути бездоганно інтегровані.

Додаток також спрощує спілкування. У календарі відображаються заплановані минулі та майбутні зустрічі між викладачем і студентом. Крім того, інструменти спілкування, такі як обмін повідомленнями або дискусійні форуми, сприяють бездоганній взаємодії.

Компоненти зворотного зв'язку та оцінювання дають змогу викладачу надати уявлення про успішність аспіранта, будь то на курсах, дослідницьких проектах чи інших академічних заняттях. Критерії оцінювання, рубрики та коментарі до завдань сприяють комплексному розумінню.

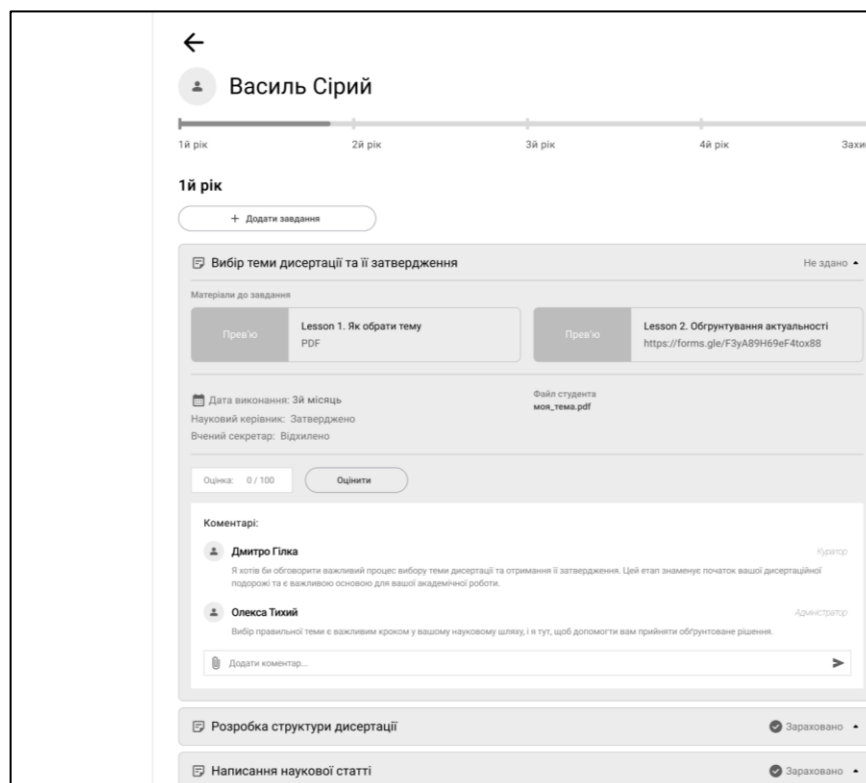


Рисунок 4.9 — Сторінка персональних планів аспірантів

Інтерфейс охоплює професійний розвиток аспіранта. Інформація, що стосується участі у семінарі, діяльності з розвитку навичок та інших етапів розвитку, збагачує розуміння викладача загального зростання студента та участі в академічній сфері. На наступному рисунку 4.10 зображено списки персональних планів аспірантів.

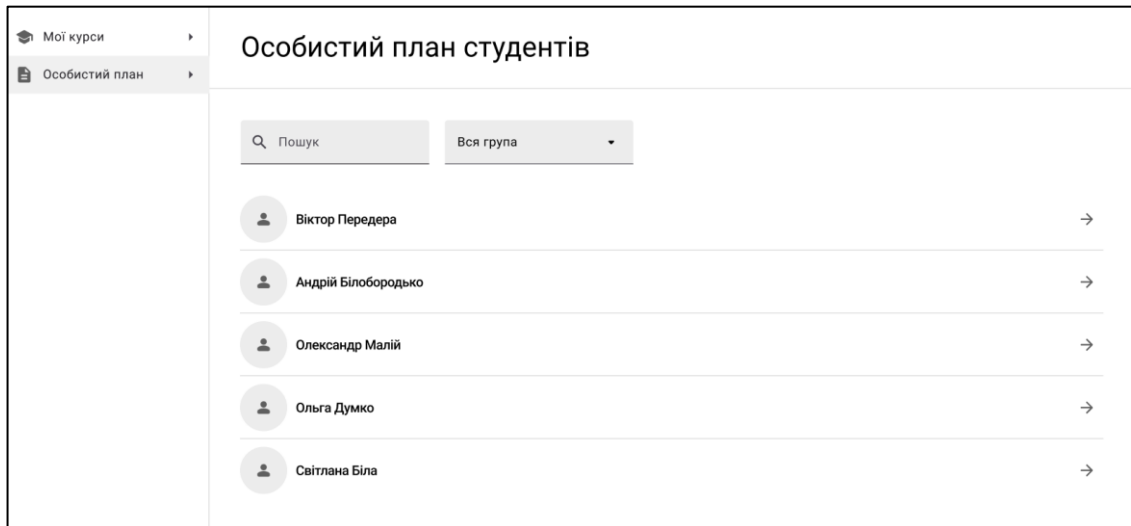


Рисунок 4.10 — Огляд особистих планів студентів

Аналізуючи рисунок 4.10, з даного списку студентів ми можемо самостійно знайти потрібний персональний план студента та оглянути його, або редагувати, натиснувши праворуч кінцеву іконку у відповідному рядку.

Висновки до розділу 4

Під час взаємодії користувача з системою програмного забезпечення безперебійний та інтуїтивно зрозумілий досвід має першочергове значення для ефективного використання. Дизайн і функціональність системи віддають перевагу принципам, орієнтованим на користувача, спрямованим на підвищення доступності та спрощення завдань.

Коли користувачі орієнтуються в системі, інтерфейс користувача виявляється ключовим елементом, пропонуючи естетично приємне та зручне середовище. Чіткі шляхи навігації та чітко визначені функції сприяють ефективній роботі користувача, зводячи до мінімуму криву навчання як для новачків, так і для досвідчених користувачів.

Інтеграція надійних функцій, таких як сторінка особистого плану, гарантує, що користувачі, зокрема аспіранти та викладачі, мають централізований центр для відстеження академічного прогресу, керування курсами та сприяння ефективній комунікації. Повний огляд, наданий на сторінці особистого плану, сприяє прийняттю обґрунтованих рішень і ефективному моніторингу академічних завдань.

Інструменти спільної роботи, вбудовані в систему, такі як канали зв'язку та механізми зворотного зв'язку, сприяють динамічному та інтерактивному взаємодії з користувачем. Повна інтеграція цих функцій сприяє ефективній комунікації між викладачами та студентами, створюючи сприятливе середовище для наставництва та академічного керівництва.

Крім того, система програмного забезпечення враховує різноманітні потреби своїх користувачів. Гнучкість платформи дозволяє виконувати різні ролі, включаючи аспірантів, викладачів і адміністраторів, гарантуючи, що кожна група користувачів може використовувати систему відповідно до своїх унікальних вимог.

У розділі «Робота користувача з програмним забезпеченням» підкреслюється прагнення задовольнити користувачів, підкреслюючи орієнтований на користувача підхід до дизайну, функціональності та загальної зручності використання програмного забезпечення. Цей фокус спрямований на розширення можливостей користувачів, сприяння співпраці та сприяння позитивному та продуктивному академічному середовищу.

5 РОЗРОБЛЕННЯ СТАРТАПУ ДЛЯ СПІЛЬНОЇ РОЗРОБКИ ТА ІНТЕГРАЦІЇ ОКРЕМИХ ЧАСТИН ПРОГРАМНИХ ПРОДУКТІВ

У цьому розділі ми розглянемо ключовий етап створення стартапу для спільної розробки та інтеграції окремих частин програмних продуктів. Цей етап є критичним у життєвому циклі проекту, оскільки визначає напрямок подальших дій, стратегії та технічні аспекти співпраці між різними розробниками та їхніми продуктами.

Розглянемо процеси формування команди для спільної розробки, визначення ролей та відповідальностей учасників. Також ми дослідимо методи інтеграції окремих частин програмних продуктів, щоб забезпечити сумісність, ефективність та стабільність у великих та складних проектах.

Цей розділ буде спрямований на розкриття ключових відомостей та кроків, необхідних для успішного запуску стартапу, який сприятиме спільній розробці та інтеграції програмних продуктів.

5.1 Опис ідеї стартапу

Розглянемо таблицю 5.1, у якій представлено опис ідеї стартапу, а саме зміст ідеї, напрямки застосування та вигоди для користувача.

Таблиця 5.1 — Опис ідеї стартапу

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Спільна розробка та інтеграція окремих частин програмних продуктів на прикладі розробки кабінету аспіранта кафедри	1. Освітній напрямок	Навчання в умовах реального світу, розвиток навичок, оволодіння новими навичками, покращення майстерності, шанс реалізувати свої інтереси та захоплення, навчання в атмосфері творчості та інновацій, рівний доступ до практичного навчання для всіх студентів
	2. Технологічні стартапи	Пошук і залучення перспективних кадрів, навчання та розвиток майбутніх співробітників, створення потужної команди для реалізації амбітних цілей, досягнення мети швидше та ефективніше, створення конкурентоспроможної команди.

Продовження таблиці 5.1.

	3. IT-спільноти	Реалізація спільних ідей та проектів, взаємонавчання та професійний розвиток.
	4. Інноваційні проекти	Співпраця між студентами та вченими для спільної роботи над науковими дослідженнями, розширення можливостей для студентів брати участь у наукових дослідженнях
	5. Колективні інновації	Створення мостів між людьми та ідеями, пошук нових шляхів для спільної роботи

Стартап пропонує платформу для спільної розробки та інтеграції окремих частин програмних продуктів. Платформа дозволяє розробникам з різних компаній та організацій об'єднуватися для створення нових програмних продуктів або для вдосконалення існуючих.

Розробка програмних продуктів є складним і трудомістким завданням, яке вимагає від розробників різноманітних навичок та знань. Часто розробникам доводиться працювати над різними частинами одного програмного продукту, що може призвести до проблем з інтеграцією цих частин.

Платформа стартапу пропонує ряд функцій, які допомагають розробникам вирішити проблему інтеграції:

- репозиторій коду: платформа надає репозиторій коду, де розробники можуть зберігати свій код і спільно працювати над ним;
- інструменти для управління проектами: платформа надає інструменти для управління проектами, які допомагають розробникам планувати свою роботу та стежити за її виконанням;
- інструменти для відстеження помилок: платформа надає інструменти для відстеження помилок, які допомагають розробникам виявляти та вирішувати проблеми з кодом.

Цей інструмент створений з метою сприяти зростанню активності та розвитку у сфері освіти та технологій, надаючи користувачам широкий спектр можливостей для співпраці, навчання та реалізації ідей у вибраних напрямках застосування.

Для визначення характеристик ідеї проекту розглянемо таблицю 5.2, що представлена нижче.

Таблиця 5.2 — Визначення характеристик ідеї проекту

№ з/п	Економічні характеристики	Концепція (стратегія) конкурентів			W	N	S
		Власний проект	eCampus	Moodle			
1.	Продуктивність	Можливість адаптувати продуктивність до індивідуальних особливостей організації.	Продуктивність бажає кращого, доволі часто немає можливості навіть увійти у обліковий запис.	Продуктивність залежить від обсягу та складності встановленої системи.			+

Продовження таблиці 5.2.

2.	Гнучкість та налаштовуваність	Широка сфера застосування і можливість індивідуальної настройки.	Можливості налаштування не дозволяють повністю адаптувати рішення до конкретних потреб.	Рішення пропонує певний простір для маневру, але не дозволяє повністю задовольнити потреби організації.			+
3.	Ризики та надійність	Рішення може бути менш стабільним і надійним, ніж рішення, розроблене та підтримуване внутрішньою командою. Однак воно може запропонувати більше можливостей для адаптації та інновацій.	Рішення може запропонувати задовільний користувацький досвід, але воно може бути менш інтуїтивним і ефективним, ніж сучасні рішення.	Рішення може бути надійним, але його надійність залежить від якості налаштування та обслуговування. Воно може бути менш стійким до змін, ніж стандартизовані рішення.		+	

Продовження таблиці 5.2.

4.	Вартість впровадження	Впровадження власної платформи для управління проектами може бути складним завданням, яке вимагає значних ресурсів.	Рішення буде прийнятним для невеликих організацій з обмеженим бюджетом, але воно може не відповідати потребам більших організацій.	Відкриті системи для управління проектами можуть бути доступними для організацій з обмеженим бюджетом.	+		
5.	Вартість підтримки	Підтримка, розширення функціоналу та виправлення помилок у відкритих системах для управління проектами можуть бути витратними, оскільки вимагають залучення кваліфікованих фахівців.	Відкриті системи для управління проектами можуть бути доступними для організацій з обмеженим бюджетом, але вимагають наявності кваліфікованих фахівців у штаті.	Платформа для управління проектами може потребувати постійного обслуговування, щоб забезпечити її безперебійну роботу та відповідність потребам користувачів.	+		

Продовження таблиці 5.2.

6.	Оновлення та розвиток	Платформа для управління проектами потребує постійних інвестицій у її вдосконалення, щоб відповідати змінним потребам користувачів та залишатися конкурентоспроможною.	Невизначеність щодо майбутніх оновлень та нових функцій може викликати у користувачів відчуття незадоволення та розчарування.	Вартість адаптації платформи для управління проектами до нових стандартів може бути непередбачуваною, оскільки вона залежить від того, як ці стандарти вплинуть на роботу організації.	+		
----	-----------------------	--	---	--	---	--	--

Порівнюючи економічні характеристики власний проект є більш функціональним і потужним, ніж eCampus та Moodle. Він пропонує широкий набір функцій, які можуть бути корисними для організацій, що потребують комплексного рішення для управління проектами.

Moodle має більш зручний інтерфейс, ніж eCampus. Він може бути хорошим вибором для організацій, які цінують простоту використання.

Moodle має більш гнучку систему налаштування, ніж eCampus. Він може бути хорошим вибором для організацій, які потребують можливості адаптації платформи під свої конкретні потреби.

Особливий акцент у своєму проекті ми зробили на наступних аспектах:

- функціональне охоплення: ми включили в проект широкий набір функцій, які можуть бути корисними для організацій, що потребують комплексного рішення для управління проектами. Це може бути конкурентним перевагою;
- зручність використання: ми прагнули створити проект з зручним інтерфейсом, який буде простим у використанні для користувачів. Це також може бути конкурентним перевагою;
- Гнучкість: ми надали користувачам можливість налаштовувати проект під свої конкретні потреби. Це може бути корисним для організацій, які потребують адаптації платформи під свої конкретні вимоги.

5.2 Технологічний аудит ідеї проекту

Технологічний аудит ідеї проекту — це інструмент, який допомагає авторам проекту оцінити, чи можна реалізувати їхню ідею з технологічної точки зору. Аудит включає в себе аналіз доступних технологій, архітектури проекту, безпеки даних та інфраструктури.

Аудит проводиться на ранньому етапі розробки проекту, коли ще є можливість внести зміни в його концептуальну основу. Результати аудиту дозволяють авторам проекту прийняти обґрунтоване рішення про те, чи варто продовжувати розробку ідеї, чи необхідно внести зміни в її архітектуру чи технології. Розглянемо таблицю 5.3, на якій представлено технологічний аудит ідеї проекту.

Таблиця 5.3 — Визначення технологічної здійсненності ідеї проекту

№ з/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Інтерфейс користувача	JavaScript, Next.js	Наявна	Доступна безкоштовно
2.	Веб-сервер	TypeScript, Node.js, Nest.js	Наявна	Доступна безкоштовно
3.	База даних	PostgreSQL	Наявна	Доступна безкоштовно
4.	Хостинг	Heroku(Купують AWS)	Наявна	Доступна безкоштовно

Висновок: проект реалізувати можливо.

Обрана технологія реалізації: інтерфейс користувача, веб-сервер, база даних, хостинг.

Технології, які будуть використовуватися в проекті, є безкоштовними і мають усі необхідні функції для виконання завдань проекту.

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Аналіз ринкових можливостей - це спосіб побачити майбутнє свого стартапу. Він дозволяє оцінити потенційні можливості і ризики, з якими ви можете зіткнутися.

Розглянемо таблиці 5.4 та 5.5, у яких звітується про труднощі та підтримку, які впливають на введення проекту на ринок.

Таблиця 5.4 — Фактори загроз

№ з/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Сильна конкуренція	Наявність конкурентів може стати серйозною перешкодою для будь-якого бізнесу. Щоб успішно конкурувати, необхідно мати унікальні переваги, які виділять вас серед інших.	Конкуренти - це не вороги, а джерело інформації. Ретельне вивчення їхніх продуктів, послуг, стратегій та цілей допоможе вам зрозуміти, як вони працюють і що вони роблять добре, а що погано. Ця інформація допоможе вам знайти можливості для покращення власного продукту чи послуги та зробити його більш конкурентоспроможним.
2.	Зміни в попиті	Уподобання споживачів можуть змінюватися дуже швидко. Щоб продукт або послуга залишалися популярними, вам потрібно постійно стежити за змінами в потребах і бажаннях	Споживачі - це головний голос ринку, що диктує правила гри. Якщо компанія хоче залишатися успішною, вона повинна слухати своїх клієнтів і відповідати на потреби.

		ваших клієнтів.	
--	--	-----------------	--

Продовження таблиці 5.4.

3.	Технологічні зміни	Технології - це рушійна сила сучасного світу. Вони постійно змінюються, і ті, хто не може їх освоїти, ризикують залишитися позаду. Якщо компанія хоче залишатися конкурентоспроможною, вона повинна постійно інвестувати в дослідження та розробки, щоб створювати продукти та послуги, які відповідають найновішим технологічним стандартам.	Технології розвиваються стрімко, і ті, хто не йде в ногу з часом, ризикують відстати від конкурентів. Компанії, які швидко реагують на технологічні зміни, можуть отримати конкурентні переваги, пропонуючи продукти та послуги, які відповідають найновішим стандартам.
4.	Політичні чи регуляторні обмеження	Зміни в законодавстві, правилах або стандартах можуть змінити правила гри для ведення бізнесу. Компанії повинні бути готові до того, що їм доведеться адаптуватися до нових вимог або обмежень, щоб залишатися в рамках закону.	Компанії не повинні бути пасивними спостерігачами, коли справа доходить до формування законодавства. Активна участь у цьому процесі може допомогти їм забезпечити, щоб законодавство було

			справедливим для бізнесу.
--	--	--	---------------------------

Продовження таблиці 5.4.

5.	Нестабільність економічної ситуації	Негативні зміни на ринку та в економіці можуть бути серйозною загрозою для бізнесу. Вони можуть призвести до того, що компанії втратить клієнтів і доходи, а це, в свою чергу, може призвести до скорочення бюджетів на закупівлі.	Розширення бізнесу в нові сегменти ринку або створення нових продуктів допоможе компанії диверсифікувати свій портфель і зменшити ризик, пов'язаний з негативними змінами в одному сегменті.
----	-------------------------------------	--	--

Таблиця 5.5 — Фактори можливостей

№ з/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Зростання попиту	Спостереження за трендами на ринку та задоволення зростаючої потреби споживачів у певних продуктах чи послугах є запорукою успіху для бізнесу.	Компанії, які хочуть бути успішними, повинні розуміти потреби своїх споживачів та розробляти продукти, які відповідають цим потребам.
2.	Нішева спеціалізація	Концентрація на вузькій, але перспективній ніші	Щоб бути успішними, компанії повинні знайти

		ринку дозволяє компанії стати експертом у цій сфері та завоювати довіру потенційних клієнтів.	те, що відрізняє їх від конкурентів, і зробити це очевидним для споживачів.
--	--	---	---

Продовження таблиці 5.5.

3.	Технологічні інновації	Інновації – це ключ до успіху в сучасному світі. Компанії, які використовують передові технології та інновації, можуть запропонувати своїм клієнтам кращі продукти та послуги, ніж їхні конкуренти.	У сучасному світі, де технології розвиваються стрімкими темпами, компанії, які не інновують, швидко втрачають свою конкурентоспроможність.
4.	Тенденції ринку	Компанії, які хочуть бути попереду своїх конкурентів, повинні бути в курсі останніх тенденцій у споживчому поведінці та уподобаннях клієнтів.	Для компаній, які хочуть досягти успіху, не достатньо бути лідером на локальному ринку. Вони повинні прагнути до глобального успіху.
5.	Глобальна доступність	Використання можливостей глобального ринку, якщо проект має потенціал для міжнародного розширення.	Глобальний ринок - це величезна можливість для компаній, які мають унікальний продукт або послугу.

Стратегічне планування на основі аналізу ринкових можливостей - це

фундамент успіху будь-якого стартапу. Важливо не лише розуміти загрози та можливості ринку, але й бути готовим до змін.

Гнучкість у реагуванні на зміни є ключовим фактором для успішного стартапу. Необхідно постійно моніторити ринок та адаптувати свою стратегію відповідно до змін.

Отримана інформація про ринок допоможе розробити стратегію введення на ринок, яка буде сприяти успішному старту та подальшому розвитку проекту.

Висновки до розділу 5

Розроблення стартапу для спільної розробки та інтеграції окремих частин програмних продуктів - це складний і багатогранний проект, який вимагає від команди глибокого розуміння ринку, технологій та бізнес-моделі. Для успішного запуску і розвитку такого стартапу необхідно дотримуватися таких факторів і рекомендацій.

Успішний запуск і розвиток стартапу в цій галузі залежить від таких факторів:

- вибір правильної ринкової ніші: стартап повинен запропонувати дійсно унікальний продукт або послугу, яка задовольняє реальні потреби ринку;
- використання передових технологій: стартап повинен використовувати найсучасніші технології, щоб забезпечити конкурентоспроможність свого продукту або послуги;
- розроблення ефективної бізнес-моделі: стартап повинен розробити бізнес-модель, яка дозволить йому генерувати прибуток і розвиватися.

ВИСНОВКИ

На завершення, дослідження та розробки, проведені для даної магістерської дисертації, були спрямовані на дослідження та впровадження інноваційних моделей і методів для комплексної розробки та інтеграції програмних продуктів. Вибраний набір технологій, зокрема JavaScript, Node.js, Nest.js, Swagger, GitHub Actions, Docker, Heroku та PostgreSQL, відіграв ключову роль у створенні надійної веб-програми, адаптованої для офісу аспіранта.

Завдяки ретельному аналізу та створенню розробленої системи стало очевидним, що вибрані технології ефективно взаємодіють, сприяючи безперебійному процесу розробки та розгортання. Процес автоматизації CI/CD, сприяний GitHub Workflow і GitHub Actions, оптимізував робочі процеси, підвищивши загальну ефективність життєвого циклу розробки.

Підрозділи, що детально розглядають залучення програмних засобів, розкривають значення кожного компонента у спільній розробці проекту. Від JavaScript і TypeScript до Node.js і Nest.js, кожна технологія внесла вагомий внесок у успіх проекту, сприяючи масштабованості, зручності обслуговування та загальній надійності системи.

Дослідження таких платформ, як Docker і Heroku, стало ще одним прикладом прихильності сучасним методам розробки, уможливлуючи контейнеризацію та ефективне розгортання хмари. Інтеграція PostgreSQL забезпечила надійне та масштабоване рішення для баз даних, що має вирішальне значення для обробки даних у програмі.

На заключний етап цього технічного дослідження, аналіз eCampus і Moodle відкрив широкий ландшафт освітніх платформ та вказав на потенційні напрямки подальших вдосконалень. За використання цих платформ було реалізовано систему, що відповідає сучасним тенденціям освітніх технологій.

Підсумовуючи, реалізація веб-додатку, спеціально адаптованого для кабінету аспіранта, є необхідною ланкою в комплексному підході до розробки програмного

забезпечення. Успішне завершення цього етапу підкреслює важливість використання комплексної моделі розробки програмного забезпечення, яка враховує як окремі компоненти, так і їх інтеграцію в єдину систему.

У міру розвитку проекту постійне вдосконалення та адаптація до нових технологій будуть важливі. Досвід, отриманий під час цієї роботи, забезпечує міцну основу для майбутніх проектів, підкреслюючи важливість бути в курсі технологічних досягнень і застосовувати спільний та інтегрований підхід до розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Flanagan D. JavaScript: The Definitive Guide. O'Reilly Media, 2020, p. 1096.
2. LaLonde W. Mastering Node.js. Packt Publishing, 2013, p. 372.
3. Bell J., Magolan G. Nest.js: A Progressive Node.js Framework Kindle Edition. 2018 p. 350.
4. Swagger Documentation. URL: <https://swagger.io/>.
5. GitHub Actions Documentation. URL: <https://docs.github.com/en/actions>.
6. GitHub Secrets Documentation. URL: <https://docs.github.com/en/actions/security-guides/encrypted-secrets>.
7. Docker Documentation. URL: <https://docs.docker.com/>.
8. Heroku Documentation. URL: <https://devcenter.heroku.com/>.
9. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/>.
10. CI/CD Process Documentation. URL: <https://www.atlassian.com/continuous-delivery/ci-cd>.
11. GitHub Workflow Documentation. URL: <https://docs.github.com/en/actions/learn-github-actions/workflow-syntax-for-github-actions>.
12. Wilson J. Web Workers: Multithreaded Programs in JavaScript. O'Reilly Media, 2012, p. 56.
13. Freeman E., Robson E. Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software. O'Reilly Media, 2004, p. 694.
14. Fowler M. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 2018, p. 455.
15. Brown A., Cormack A. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010, p. 512.
16. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015, p. 280.
17. Cooper P., Love T., Hall D. Beginning Node.js. Apress, 2014, p. 372.
18. Schildt H. C: The Complete Reference. McGraw-Hill Education, 2017, p. 1056.

19. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002, p. 560.
20. O'Reilly T., Holzschlag M. HTML & XHTML: The Definitive Guide. O'Reilly Media, 2000, p. 672.
21. Chodorow K. MongoDB: The Definitive Guide. O'Reilly Media, 2010, p. 432.
22. Abelson H., Sussman G. J., Sussman J. Structure and Interpretation of Computer Programs. MIT Press, 1996, p. 657.
23. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017, p. 616.
24. Dewar R. B., Smosna M. Ada 95: The Lovelace Tutorial. Springer, 1996, p. 559.
25. Kernighan B. W., Ritchie D. M. The C Programming Language. Prentice Hall, 1988, p. 274.
26. Deitel H. M., Deitel P. J. C++ How to Program. Prentice Hall, 2017, p. 1080.
27. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994, p. 395.
28. Roberts E. S. Thinking Recursively with Java. John Wiley & Sons, 2006, p. 256.
29. Ousterhout J. Tcl and the Tk Toolkit. Addison-Wesley, 1994, p. 832.
30. Hunt A., Thomas D. The Pragmatic Programmer: Your Journey to Mastery. Addison-Wesley, 2019, p. 352.
31. McFarland D. JavaScript & jQuery: The Missing Manual. O'Reilly Media, 2014, p. 686.
32. Wilson J. Web Workers: Multithreaded Programs in JavaScript. O'Reilly Media, 2012, p. 56.
33. Freeman A., Robson E. Head First HTML and CSS: A Learner's Guide to Creating Standards-Based Web Pages. O'Reilly Media, 2012, p. 768.
34. Poth K. Pro Git. Apress, 2014, p. 456.
35. Git Documentation. URL: <https://git-scm.com/doc>.
36. Chacon S., Straub B. Pro Git. Apress, 2014, p. 456.
37. Node.js Documentation. URL: <https://nodejs.org/en/docs/>.

38. Docker Compose Documentation. URL: <https://docs.docker.com/compose/>.
39. Cloud-Native Applications in Java. URL: <https://www.oreilly.com/library/view/cloud-native-applications/9781491984307/>.
40. PostgreSQL Official Documentation. URL: <https://www.postgresql.org/docs/>.
41. GitHub Actions Marketplace. URL: <https://github.com/marketplace/actions>.
42. Heroku Dev Center. URL: <https://devcenter.heroku.com/>.
43. CI/CD Best Practices. URL: <https://cloud.google.com/solutions/devops/cicd-best-practices>.
44. GitHub Workflow Syntax. URL: <https://docs.github.com/en/actions/learn-github-actions/workflow-syntax-for-github-actions>.
45. React.js Documentation. URL: <https://reactjs.org/docs/getting-started.html>.
46. TypeScript Documentation. URL: <https://www.typescriptlang.org/docs/>.
47. Nest.js Documentation. URL: <https://docs.nestjs.com/>.
48. Next.js Documentation. URL: <https://nextjs.org/docs>.
49. Swagger Documentation. URL: <https://swagger.io/>.
50. GitHub Secrets Documentation. URL: <https://docs.github.com/en/actions/security-guides/encrypted-secrets>.
51. Continuous Integration Best Practices. URL: <https://circleci.com/best-practices/>.
52. Introduction to JavaScript. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction>.
53. Understanding CI/CD. URL: <https://www.redhat.com/en/topics/devops/what-is-ci-cd>.
54. Web Development with Docker. URL: <https://www.docker.com/solutions/web-development>.

ДОДАТОК А

Моделі та методи спільної розробки та інтеграції окремих частин програмних продуктів на прикладі розробки веб-додатку кабінету аспіранта кафедри

Лістинг програми
УКР.НТУУ «КПІ ім. Ігоря Сікорського»

Аркушів 4

Київ — 2024

backend.yml

name: Deploy to Heroku

on:

push:

branches:

- main

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Docker

uses: docker/setup-buildx-action@v3

- name: Build and push Docker image

run: |

IMAGE_NAME=my-docker-image-\${GITHUB_SHA::8}

Set the build context to the project root

docker build -t \$IMAGE_NAME -f dockerfile .

Log in to Heroku container registry

docker login --username=_ --password-stdin registry.heroku.com <<< \${secrets.HEROKU_API_KEY }}

```

    docker tag $IMAGE_NAME registry.heroku.com/${{
secrets.HEROKU_BACKEND_APP_NAME }}/web
    docker push registry.heroku.com/${{
secrets.HEROKU_BACKEND_APP_NAME }}/web
    echo "DOCKER_IMAGE_NAME=$IMAGE_NAME" >> $GITHUB_ENV

```

- name: Deploy to Heroku

env:

```

    HEROKU_API_KEY: ${ secrets.HEROKU_API_KEY }
    HEROKU_BACKEND_APP_NAME: ${
secrets.HEROKU_BACKEND_APP_NAME }

```

run: |

```

# Install Heroku CLI (if not already installed)
if ! command -v heroku &> /dev/null; then
    curl https://cli-assets.heroku.com/install.sh | sh
fi

```

```

# Deploy the Docker container to Heroku

```

```

heroku container:release -a $HEROKU_BACKEND_APP_NAME web

```

- name: Clean up

run: |

```

docker buildx prune -f

```

main.ts

```

import { NestFactory } from '@nestjs/core';
import { AppModule } from './modules/app/app.module';
import { ValidationPipe } from '@nestjs/common';

```



```

import { NestExpressApplication } from '@nestjs/platform-express';
import * as cookieParser from 'cookie-parser';
import { ConfigService } from '@nestjs/config';
import { setupSwagger } from './swaggerConfig'; // Import the Swagger configuration

async function bootstrap() {
  const app = await NestFactory.create<NestExpressApplication>(AppModule);

  const configService = app.get(ConfigService);
  app.enableCors({
    origin: '*',
    credentials: true,
  });

  app.setGlobalPrefix('api', { exclude: [''] });

  app.use(cookieParser());
  app.enableShutdownHooks();
  app.useGlobalPipes(
    new ValidationPipe({
      transform: true,
      transformOptions: { enableImplicitConversion: true },
    }),
  );

  // Set up Swagger documentation
  setupSwagger(app);

  await app.listen(configService.get('PORT') || 3303);
}

```

```
}
```

```
bootstrap();
```

swaggerConfig.ts

```
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';
```

```
import { INestApplication } from '@nestjs/common';
```

```
export function setupSwagger(app: INestApplication): void {
```

```
  const config = new DocumentBuilder()
```

```
    .setTitle('GraduateOfficeDiploma')
```

```
    .setDescription(
```

```
      'API documentation for GraduateOfficeDiploma back-end (Nest.js)',
```

```
    )
```

```
    .setVersion('1.0')
```

```
    .build();
```

```
const document = SwaggerModule.createDocument(app, config);
```

```
SwaggerModule.setup('api-docs', app, document);
```

```
}
```

ДОДАТОК Б

Моделі та методи спільної розробки та інтеграції окремих частин програмних продуктів на прикладі розробки веб-додатку кабінету аспіранта кафедри

Презентація
УКР.НТУУ «КПІ ім. Ігоря Сікорського»

Аркушів 7

Київ — 2024



Моделі та методи спільної розробки та інтеграції окремих частин програмних продуктів на прикладі розробки веб-додатку кабінету аспіранта кафедри

Виконав: студент магістерського рівня 2-го року навчання, групи ТВ-321мп

Лисик Юрій Юрійович

Керівник: доктор технічних наук

Недашківський Олексій Леонідович



АКТУАЛЬНІСТЬ ТЕМИ ДОСЛІДЖЕННЯ

В умовах функціонування академічної спільноти, що швидко розвивається, розробка веб-додатку, спеціально розробленого для навчання в аспірантурі, є не лише вимогою часу, але й стратегічним імперативом.

Проведене дослідження стосується гострої потреби в комплексних та інтегрованих рішеннях, підвищення адміністративної ефективності, академічних консультацій та комунікації під час навчання в аспірантурі та може бути використано в інших академічних задачах.

Актуальність полягає в значному потенціалі трансформувати традиційні практики, відкриваючи нову еру цифрової та орієнтованої на користувача підтримки аспірантів.



МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета: Підвищення ефективності інтеграції та автоматизації процесу розробки і розгортання частини програмних продуктів в середовищі експлуатації за допомогою розробки механізмів комплексної розробки та інтеграції окремих частин програмних продуктів.

Об'єкт дослідження: Веб-додаток кабінету аспіранта кафедри.

Предмет дослідження: Моделі та методи комплексної розробки та інтеграції окремих частин програмних продуктів.



ВИЗНАЧЕНІ ЗАВДАННЯ

Для досягнення вищезазначеної мети були визначені наступні завдання

1. Дослідити різні моделі та методи, що використовуються для комплексної розробки програмного забезпечення для навчальних закладів;
2. Проаналізувати системи контролю версій з метою забезпечення безперебійної, злагодженої та контрольованої співпраці членів команди розробників;
3. Розробити схему організації спільної роботи команди з елементами автоматизації процесів при створенні та розгортанні кабінету аспіранта кафедри.



ІСНУЮЧІ АНАЛОГИ КІНЦЕВИХ ПРОГРАМНИХ СИСТЕМ



Переваги:

- Наявність модулів для моніторингу успішності (журнал, атестації, сесія).
- Наявність навчальних матеріалів
- Можливість надсилати та переглядати повідомлення

Недоліки:

- Відсутність можливості створювати та оцінювати завдання
- Невідповідність користувацького інтерфейсу сучасним стандартам



Google Classroom

Переваги:

- Наявність механізму створення та оцінки завдань
- Відповідність стандартам сучасного користувацького інтерфейсу.

Недоліки:

- Відсутність модулів для спільної роботи над науковою роботою
- Складність інтеграції в існуючі системи підтримки навчального процесу



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

Слайд 5/14

РОЛІ ТА МОЖЛИВОСТІ КОРИСТУВАЧІВ СИСТЕМИ, ЩО СТВОРЮЄТЬСЯ В РАМКАХ СПІЛЬНОГО ПРОЄКТУ

Роль	Можливості
Викладач	Перегляд профілю аспірантів, які вивчаються під їх керівництвом, перегляд академічних досягнень своїх аспірантів, виставлення та оцінювання завдань та лекцій для аспірантів, обговорення та обмін повідомленнями з аспірантами, завантаження та редагування навчального матеріалу.
Аспірант	Перегляд свого профілю та особистої інформації, перегляд розкладу та завдань, які надаються викладачами, відправлення завдань та лекцій для оцінки викладачам, обговорення та обмін повідомленнями з викладачами та іншими аспірантами, доступ до бібліотеки навчального матеріалу.
Завідувач кафедри	Моніторинг академічної активності викладачів та аспірантів на кафедрі, затвердження курсів та навчальних планів.
Адміністратор	Управління доступом та ролями користувачів в системі, моніторинг та підтримка технічного функціонування системи, забезпечення безпеки та конфіденційності даних, резервне копіювання та відновлення даних, вирішення технічних питань та надання підтримки користувачам.

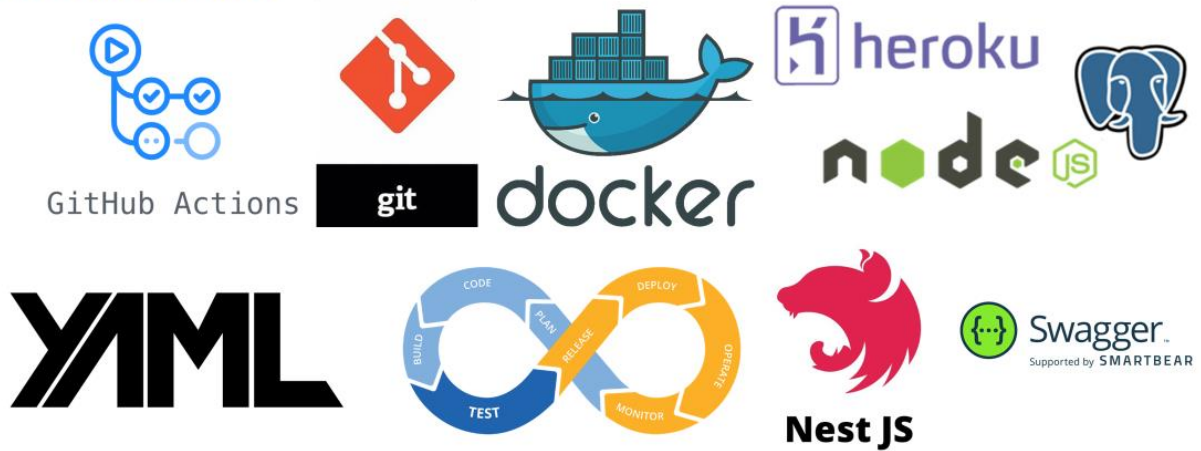


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Слайд 6/14

ВИКОРИСТАНІ ПРОГРАМНІ ЗАСОБИ

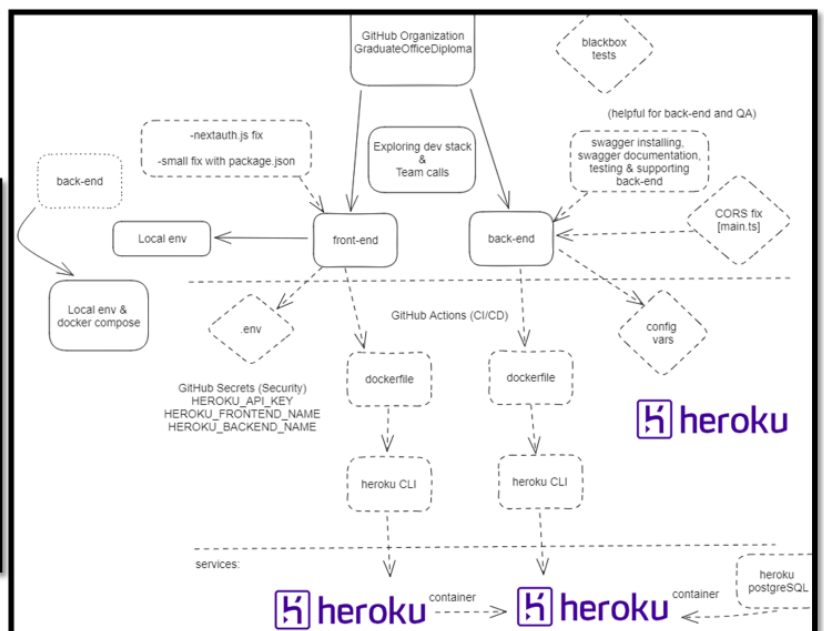
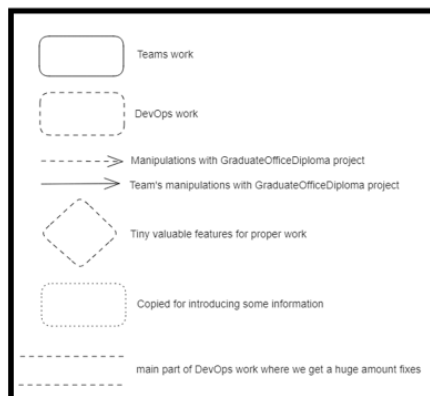
ЯКІ ПЕРЕДБАЧАЮТЬСЯ ЗАЛУЧАТИСЯ УЧАСНИКАМИ РОЗРОБКИ В РАМКАХ СПІЛЬНОГО ПРОЄКТУ



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

Слайд 7/14

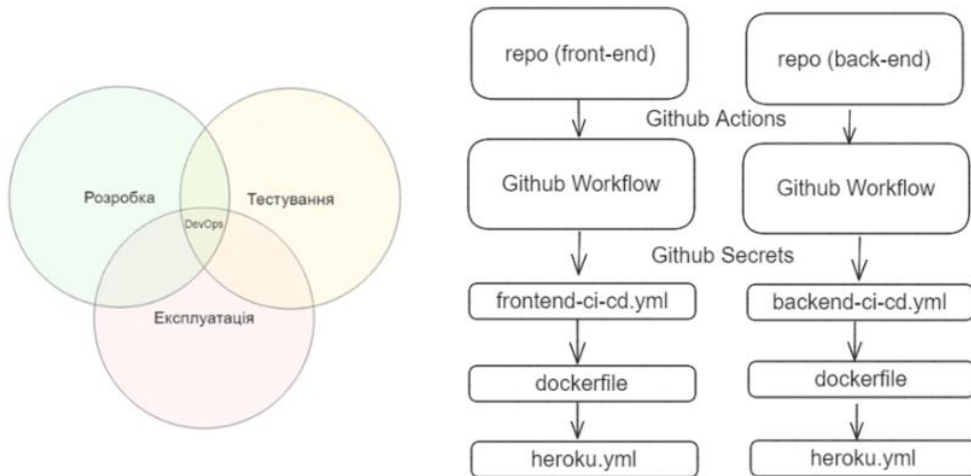
РОЗРОБЛЕНА СХЕМА ОРГАНІЗАЦІЇ СПІЛЬНОЇ РОБОТИ



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

Слайд 8/14

ОСНОВНИЙ ФУНКЦІОНАЛ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ РОЗРОБЛЕНОЇ СХЕМИ ОРГАНІЗАЦІЇ СПІЛЬНОЇ РОБОТИ



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

Слайд 9/14

ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБЛЕНОЇ СХЕМИ ОРГАНІЗАЦІЇ СПІЛЬНОЇ РОБОТИ

```
1 name: Deploy to heroku
2
3 on:
4   push:
5     branches:
6       - main
7
8 jobs:
9   deploy:
10    runs-on: ubuntu-latest
11
12    steps:
13      - name: Checkout code
14        uses: actions/checkout@v2
15
16      - name: Set up Docker
17        uses: docker/setup-buildx-action@v3
18
19      - name: Build and push Docker image
20        run: |
21          IMAGE_NAME=my-docker-image-${GITHUB_SHA:0:8}
22          # Set the build context to the project root
23          docker build -t $IMAGE_NAME -f dockerfile .
24          # Log in to Heroku container registry
25          docker login --username=_ --password=${HEROKU_API_KEY} registry.heroku.com
26          docker tag $IMAGE_NAME registry.heroku.com/${HEROKU_BACKEND_APP_NAME}/web
27          docker push registry.heroku.com/${HEROKU_BACKEND_APP_NAME}/web
28          echo "DOCKER_IMAGE_NAME=$IMAGE_NAME" >> $GITHUB_ENV
```

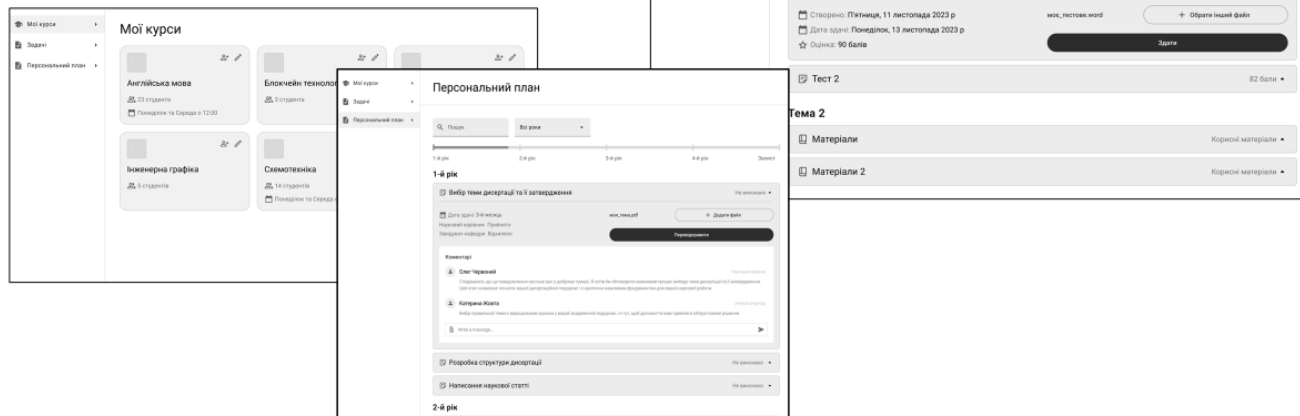
```
30 - name: Deploy to Heroku
31   env:
32     HEROKU_API_KEY: ${ secrets.HEROKU_API_KEY }
33     HEROKU_BACKEND_APP_NAME: ${ secrets.HEROKU_BACKEND_APP_NAME }
34   run: |
35     # Install Heroku CLI (if not already installed)
36     if ! command -v heroku && /dev/null; then
37       curl https://cli-assets.heroku.com/install.sh | sh
38     fi
39
40     # Deploy the Docker container to Heroku
41     heroku container:release -a $HEROKU_BACKEND_APP_NAME web
42
43 - name: Clean up
44   run: |
45     docker buildx prune -f
```



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

Слайд 10/14

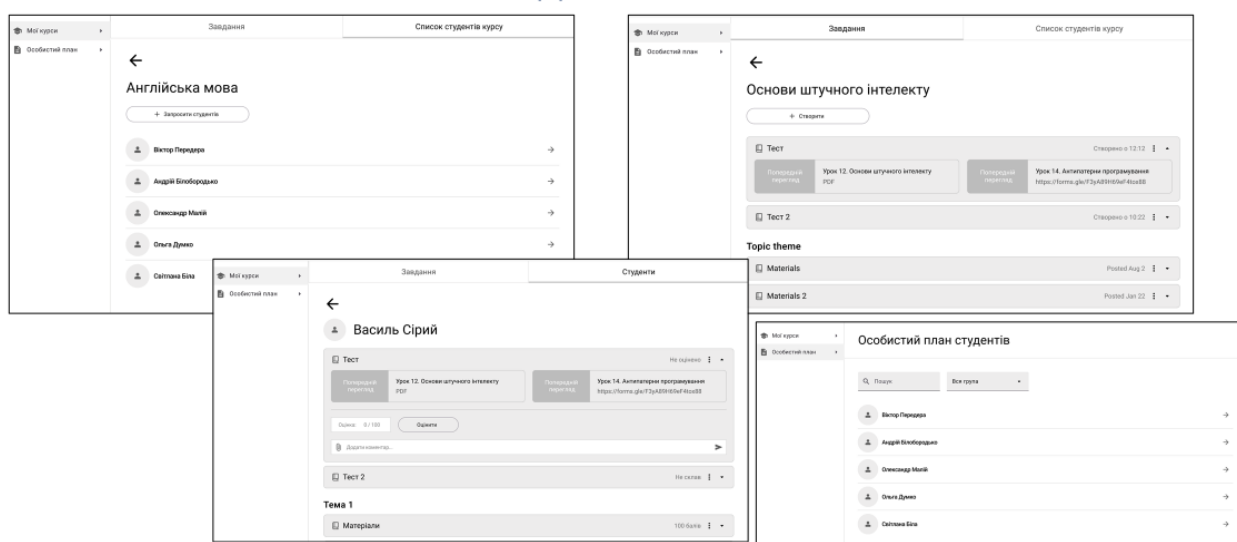
ПРЕДСТАВЛЕННЯ УСПІШНОГО РОЗГОРТАННЯ ПРОЕКТУ ЗІ СТОРОНИ АСПІРАНТА



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

Слайд 11/14

ПРЕДСТАВЛЕННЯ УСПІШНОГО РОЗГОРТАННЯ ПРОЕКТУ ЗІ СТОРОНИ ВИКЛАДАЧА



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

Слайд 12/14

ВИСНОВКИ

1. Програмний продукт дозволяє організувати спільну розробку та інтеграцію окремих частин програмних продуктів при створенні та додаванні нового функціоналу до кабінету аспіранта кафедри.
2. Підібраний стек технологій надає сучасні, масштабовані та безпечні рішення. Основна з яких - контейнеризація, що прискорює розгортку на хмарному сховищі, та дає ефективну утилізацію ресурсів та сприяє масштабованості застосунку.
3. Розроблена схема організації спільної роботи команди з елементами автоматизації процесів при створенні та розгортанні кабінету аспіранта кафедри, на основі використання систем контролю версій.
4. Результат спільної роботи команди над «Веб-додатком кабінету аспіранта кафедри» підтверджує працездатність запропонованих рішень.



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

Слайд 13/14



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

National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"