Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп’ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр
(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ СЕРВІСІВ ДЛЯ ПІДТРИМКИ КЛІЄНТСЬКИХ ЗАСТОСУВАНЬ ДЛЯ СТУДЕНТІВ УЧАСНИКІВ МІЖНАРОДНОЇ МОБІЛЬНОСТІ
DEVELOPMENT OF THE SERVICES FOR SUPPORTING CLIENT APPLICATIONS FOR THE STUDENTS PARTICIPATING IN INTERNATIONAL MOBILITY PROGRAMMES

Виконав: студент 4 курсу, групи КНТ-137
Спеціальності 121 Інженерія програмного забезпечення
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Соколовський Д.В.
(прізвище та ініціали)
Керівник Табунщик Г.В.
(прізвище та ініціали)
Рецензент Шитікова Л.В.
(прізвище та ініціали)

2021

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

(повне найменування закладу вищої освіти)

Інститут, факультет ІІРЕ, ФКНТ

Кафедра програмних засобів

Ступінь вищої освіти бакалавр

Спеціальність 121 Інженерія програмного забезпечення

(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення

(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.

________________**С.О. Субботін**

"____" _______________20___року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Соколовського Даніла Віталійовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розроблення сервісів для підтримки клієнтських застосувань для студентів учасників міжнародної мобільності. Development of the Services for Supporting Client Applications for the Students Participating in International Mobility Programmes

керівник проєкту (роботи) Табунщик Галина Володимирівна, професор,

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

затверджені наказом закладу вищої освіти від " 30 " березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) 14 травня 2020 року

3. Вихідні дані до проєкту (роботи) технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір та обґрунтування структури системи, яка проектується. 3. Основні рішення з реалізації компонентів системи. 4. Керівництво програміста.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Табунщик Г.В., професор		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання “ 26 ” лютого 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області, розробка та затвердження технічного завдання	1 тиждень	Технічне завдання
2	Проектування структури системи	2 тиждень	Структурна схема
3	Розробка схеми функціонування системи та модулів програми	3 тиждень	Функціональна схема. Модулі
4	Створення програмного коду системи	4-5 тиждень	Текст програми
5	Тестування та відналагодження програми	6 тиждень	Працездатна система
6	Розробка розділів пояснювальної записки	7 тиждень	Пояснювальна записка
7	Розробка графічного матеріалу та захист дипломної роботи	8 тиждень	Графічний матеріал та дипломна робота

Студент ________ Соколовський Д.В.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) ________ Табунщик Г.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 87с., 12 табл., 24 рис., 2 дод., 5 джерел.

ВЕБЗАСТОСУНОК, АКАДЕМІЧНА МОБІЛЬНІСТЬ, API, EXPRESS, MONGODB, NODE, VISUAL STUDIO.

Об’єкт дослідження – сервіси для міжнародної мобільності.

Предмет дослідження – програмні засоби реалізації сервісів для підтримки клієнтських застосувань для студентів учасників міжнародної мобільності.

Мета роботи – розробка сервісів для підтримки клієнтських застосувань для студентів учасників міжнародної мобільності.

Матеріали, методи та технічні засоби: вебпрограмування, платформа Node.js, фреймворк Express, база даних MongoDB, середа тестування Jest, персональний комп’ютер з процесором Intel Core 5 під управлінням операційної системи Ubuntu 20.04.

Результати. Створено API для підтримки клієнтських застосувань для міжнародної мобільності.

Висновки. Розроблено сервіси для клієнтських застосувань, які оптимізують подачу та обробку заявок на участь в міжнародній мобільності.

Галузь використання – клієнсткі застосування, розроблені для процесів міжнародної мобільності.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 87 pages, 24 figures, 12 tables, 2 appendixes, 5 sources.

WEB APPLICATION, ACADEMIC MOBILITY, API, EXPRESS, MONGODB, NODE, VISUAL STUDIO.

Object of study: services for international mobility.

Subject of study: software tools for the implementation of services to support customer applications for students participating in international mobility

The purpose of the work is to develop the services for supporting client applications for the students participating in international mobility programmes

Materials, methods, and tools: web programming, Node.js platform, Express framework, MongoDB database, Jest testing environment, personal computer with Intel Core 5 processor running Ubuntu 20.04.

Results. API to support client applications for international mobility was created.

Conclusions. Services for client applications that optimize the submission and processing of applications for participation in international mobility were created.

The field of use: customer applications designed for international mobility processes.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface;

IT – Information Technology;

HTTP – HyperText Transfer Protocol;

ПЗ – програмне забезпечення;

ПП – програмний продукт.

ВСТУП

У нашому столітті програми обміну для студентів є дуже популярними, бо навчання за кордоном - це незамінний досвід, престижний диплом і шанс побудувати міжнародну кар'єру. Передові наукові розробки та технологічні стартапи, робота в кращих компаніях і можливість подорожувати по всьому світу - освіта за кордоном відкриває безліч нових дверей. Освіта за кордоном потрібна не тільки для побудови успішної кар'єри. Вона вчить не боятися невдач, правильно розставляти пріоритети і ширше дивитися на світ.

Часто процес проходження етапів для участі у мобільності потребує немало часу. Бо студент повинен знайти необхідну інформацію для участі у мобільності у конкретному навчальному закладі у багатьох ресурсах.

Саме цьому треба розробити сервіси для підтримки клієнтських застосувань для студентів учасників міжнародної мобільності, додати можливість роботи з базою даних, порівняння можливих мобільностей, оптимізувати процеси подачи заявки на участь та обробки заявок адміністратором.

Ця задача буде вирішуватися за допомогою аналізу предметної області та service-oriented architecture (SOA) з використанням сучасних інструментів, які допоможуть вирішити завдання.

Програмні засоби, які реалізують компоненти бізнес-процесів, оформляються як сервіси. Сервіс - це програмний компонент, який реалізує закінчену функцію надання або обробки даних, переводячи їх з одного цілісного стану в інший. Основною відмінністю сервісу від звичайного компонента є стандартний і платформено-незалежний інтерфейс. Клієнт, який звертається до сервісу, не зобов'язаний нічого знати про подробиці реалізації сервісу: якою мовою і в якій моделі програмування він створений, на яких апаратних засобах, в якому операційному середовищі, на якій платформі проміжного програмного забезпечення він виконується. Сервісно-орієнтована архітектура, таким чином, дозволяє компонувати бізнес-процеси з компонентів, які виконуються на різних платформах, представляти у вигляді сервісів і повторно використовувати в нових бізнес-процесах успадковані компоненти. В останньому випадку самі успадковані компоненти не змінюються, але для них тільки робиться оболонка, що адаптує їх інтерфейси до стандартів сервісів.

За своєю суттю SOA не містить нових революційних ідей, а є узагальненням найкращих практик створення програмно-інформаційних систем рівня підприємства і вище, вона не привнесла нічого оригінального, але вона служить квітессенцією ведення інтеграційних проектів.

Основними причинами появи SOA є висока динаміка сучасного бізнесу і неухильно зростаючі вимоги до постійної адаптації інформаційних систем по відношенню до цієї динаміки. Уже недостатньо, щоб інформаційна система забезпечувала просту автоматизацію інформаційних і розрахункових завдань бізнесу. Необхідно прагнути до того, щоб швидко мінливі умови бізнесу, що виникають внаслідок посилення конкуренції, знаходили повне відображення в інформаційній системі, тобто корпоративна інформаційна система повинна змінюватися настільки ж швидко, наскільки швидко змінюються вимоги бізнесу і бізнес-процеси компанії.

Сервісна орієнтація являє собою таке ідеальне бачення світу, в якому ресурси чітко розділені і послідовно представлені в термінах сервісів. Концептуальна ідея SOA полягає в тому, щоб «завертати» діяльність в окремі керовані, слабкозв'язані елементи, - послуги. Завжди є постачальник і споживач послуги. Послуга має цінність для споживача і йому не важливо, яким чином вона створюється (які технології, які алгоритми, - важливий тільки результат). Таким чином, сервіс найкращим чином відображає «взаємодію» - поняття, яке стає все більш значущим для сучасних компаній з гнучкими ланцюжками і мережами створення цінності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Освіта за кордоном - перший крок до побудови яскравої міжнародної кар'єри. Зараз існує багато програм, що надають таку можливість. Але для участі у програмі необхідно пройти деякі етапи, а створення відповідних сервісів може значно полегшити цей процес (вчасне оформлення та подання необхідних документів, обробка важливої інформації про обраний начальний заклад та інше). Використовуючи спецільні сервіси, які будуть підключені до клієнтських застосувань, у студентів не буде необхідності шукати сайти університетів, які не завжди є «user-friendly», витрачати час на пошук розділу для міжнародної мобільності, перевіряти умови для участі. Саме цьому необхідно розробити Application Programming Interface (API) – це інтерфейс взаємодії між сайтом і сторонніми програмами і серверами. Програміст може скористатися API для отримання доступу до функціоналу сторонньої програми. API робить можливим роботу ресурсів, які використовують потенціал і потужність іншого сайту або програми. Використання API скорочує необхідність створювати самостійно складні програми або додатки. Замість цього можна використовувати готові частини існуючих ресурсів, у яких є доступ до потрібної вам інформації і даних. Отже, створення API для підтримки клієнтських застосувань для студентів учасників міжнародної мобільності є актуальною задачею.

Ця система матиме сервіс-орієнтовану архітектуру, що будується за рахунок проектування та розробки сервісів і засобів їх підключення. Сервіс є певною роботою або бізнес-функцією, яка призначена для забезпечення узгодженої роботи додатків. SOA не залежить від мов програмування, платформ або протокольних специфікацій, за допомогою яких сервіси розробляються. Зокрема принципами SOA є:

- архітектура не прив'язана до певної технології;
- організація системи не залежить від використовуваних платформ;
- організація системи не залежить від вживаних мов програмування;

— використовуються сервіси, незалежні від конкретних додатків, з однаковими інтерфейсами доступу до них;

— сервіси організовуються як слабозв’язані компоненти для побудови систем.

Одним із головних аспектів SOA, який потрібен нам у системі сервісів, що розроблюється, є висока гнучкість, яку можна досягнути за рахунок можливості швидкого коректування бізнес-логіки - зміна, що вноситься в бізнес-функцію, в результаті торкнеться всіх необхідних додатків [1].

Іншою архітектурною особливістю сервісів для підтримки клієнтських застосувань для студентів учасників міжнародної мобільності є CRUD. CRUD - це акронім від CREATE, READ, UPDATE, DELETE. Це форма стандартних команд баз даних, які засновують CRUD. У даному застосунку CRUD використовується для користувачів, університетів, факультетів, спеціальностей, програм обміну, навчальних планів, Сілабусу. CRUD операції проектовані для поліпшення постійного сховища - з записами баз даних, наприклад. Як випливає з назви, постійне сховище переживає процеси, які його створили.

CRUD пропонує безліч переваг, включаючи:

— він спрощує та полегшує проектування додатків, роблячи їх більш масштабованими;

— полегшує контроль безпеки, задовольняючи різні вимоги доступу;

— краща продуктивність.

Отже, сервіси для підтримки клієнтських застосувань для студентів учасників міжнародної мобільності значно оптимізують процес аналізу пропозицій для мобільності та подачі заявки на участь наступним чином:

— економія часу завдяки користуванню єдиною системою;

— реєстрація, авторизація, зберігання особистої інформації;

— порівнювання відкритих для студентів програм мобільності, фільтрація, пошук;

— можливість подавати заявки черех єдину систему;

— система розроблена для використання студентами та координаторами.

На жаль, зараз існує досить невелика кількість систем, схожих на запропоновану вище. Приклади будуть розглянуті у наступному розділі.

1.2 Огляд існуючих методів вирішення завдання

1.2.1 Сервіс від Дортмундського університету

Дортмундський університет реалізував власну систему для учасників міжнародної мобільності (рис. 1.1).

Перевагами даної програми є присутність локалізації та мінімалістичність, тобто використовуються лише потрібні дані, немає купи зайвих даних.

Із недоліків варто зазначити те, що обробка особистої інформації не зовсім раціональна. Бо зараз при кожному створенні заявки на участь потрібно вводити особисту інформацію, а це потребує немало часу. Це має бути оптимізовано наступним чином: реалізувати зберігання інформації у приватному кабінеті, отже, її треба буде вводити лише один раз, а потім ці дані автоматично будуть додаватись до заявки.

Також є помилки в роботі локалізації. При виборі конкретної мови не весь текст сайту перекладається.

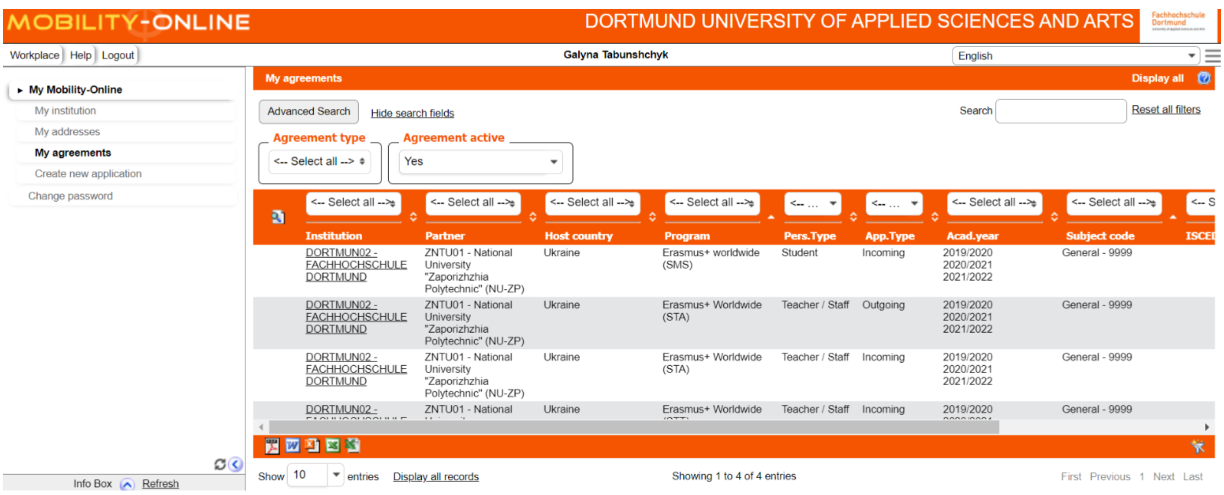

Рисунок 1.1 — Сторінка сайту від Дортмундського університету

1.2.2 Сервіс Київського національного університета

Відділ міжнародної мобільності Київського національного університету ім. Тараса Шевченка має власний сервіс. Із переваг можна виділити те, що є локалізація (українська та англійська мови).

На жаль, на сайті замало функцій для користувачів. Наприклад, не працює мапа та також вміст сторінки «Про університет» (рис. 1.2).

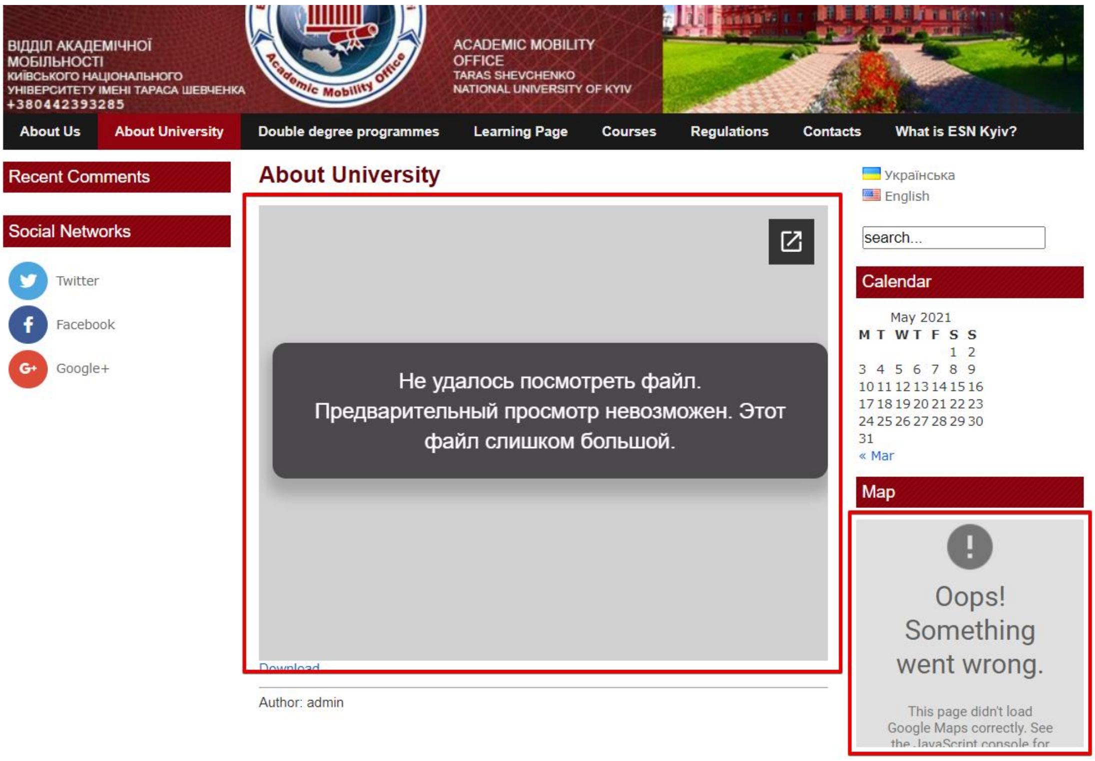

Рисунок 1.2 – Непрацюючі елементи сайту

На сайті немає можливості порівнювання поточного навчального плану студента з навчальним планом від університета-партнера, це можна буде оптимізувати, щоб взаємодія студентів з сайтом була максимально ефективною.

Форма для підписки на новини не має валідацію (рис. 1.3). Це може призвести до помилок.

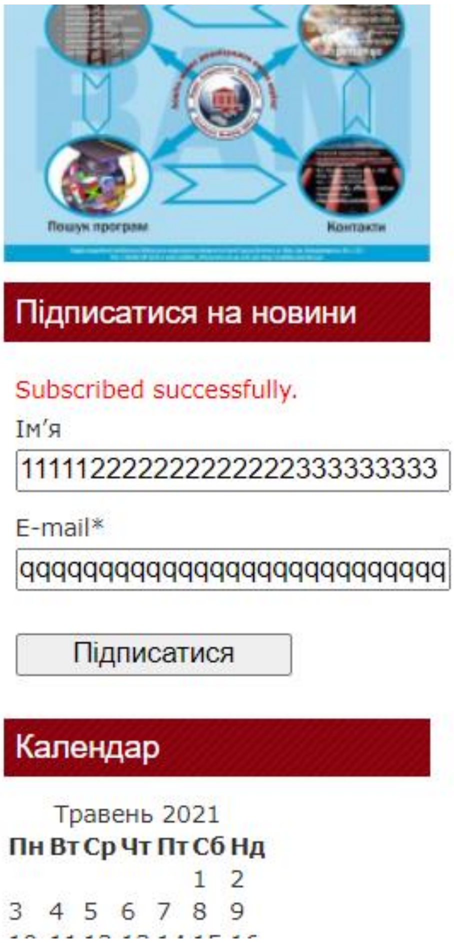

Рисунок 1.3 – Відсутність валідації

1.3 Технічне завдання

Програма призначена використання наступних функцій:

— ендпоінти для авторизації користувача (логін, реєстрація, забув пароль, відновити пароль);

— запит на отримання даних користувача і присутність обмежень (інші користувачі не мають доступу до особистих даних інших користувачів, виняток - адміністратор);

— запит на отримання списку університетів мобілізації з фільтрацією;

— запит на отримання даних про університет (факультети, спеціальності, програми мобільності);

— запит на створення заявки на участь в мобільності;

— CRUD для наступних сутностей: університет, факультет, спеціальність, програма мобільності, навчальна програма, спеціальність, Сілабус, предмет.

1.4 Висновок до розділу 1

Після визначення методів та програмних засобів реалізації системи можна сказати, що наразі не існує зручного програмного засобу для підтримки клієнтських застосувань для студентів учасників міжнародної мобільності. Саме цьому наш програмний продукт має мати вдосконалений функціонал, який є у існуючих програм, без тих недоліків, які є в існуючих системах підтримки.

2 ВІБІР ТА ОБГРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЕКТУЄТЬСЯ

2.1 Вибір інструментарію розробки програмного забезпечення

2.1.1 Вибір платформи

Node.js - це програмна платформа, яка робить JavaScript мовою загального призначення. Її також називають середовищем виконання JS. Вона вміє зв'язуватися з зовнішніми бібліотеками, викликати команди з коду і виконувати роль веб-сервера. Іншими словами, цей інструмент додає до повністю фронтендової мови бекендову частину, дозволяючи створювати з його допомогою не тільки веб-сайти, а й повноцінні програми, без задіяння браузера. Для розробки програмного забезпечення було обрано саме JS, тому що ця мова програмування задовольнювала всім потребам більше, ніж інші. Докладне порівнювання мов програмування наведено у таблицях 2.1 – 2.4.

Найбільш значимий плюс Node.js - неблокуюча модель введення-виведення. Ця система керована подіями і працює асинхронно, вибудовуючи чергу за пріоритетністю. Коли до сервера одночасно підключаються тисячі людей, їй легше впоратися з навантаженням, так як немає потреби створювати окремий потік для кожного підключення. Грамотний розподіл ресурсів допомагає витримати більшу кількість з'єднань.

Також перевагою Node.js над іншими платформами є те, що він розроблявся спеціально для web. Тому вільно взаємодіє з найбільш популярними базами даних, допомагає отримати низькорівневий доступ (http, udp, https, tcp). Кількість готових модулів і зовнішніх бібліотек Node.js безперервно зростає, чому сприяє використання npm.

Швидкість – це про Node.js. Бо створення робочого прототипу, що справляється з навантаженням, не відніме багато часу при використанні цієї платформи.

Технологія стрімко поліпшується, над цим працюють сотні програмістів всього світу. Процес координує фонд Node. Складно недооцінити роль ком'юніті, адже це допомога зі складнощами, велика кількість навчальних матеріалів, оперативно знайдені помилки у функціонуванні системи, можливість обговорити ідеї або знайти команду. З "живою" спільнотою розвиток технології відбувається набагато швидше, і все завдяки відкритому коду. Популярність і переваги Node js не залишилися непоміченими, його вже вибрали такі компанії, як Google, Ebay / PayPal, Netflix, Microsoft, Uber, Linkedin, Yahoo, Walmart і інші [2].

Таблиця 2.1 – Парадигми

1	Можливість	Мова					
		C#	Go	Java	JavaScript	PHP	Python
2	Імперативна	+	+	+	+	+	+
3	Об’єктно-орієнтована	+	+/-	+	+	+	+
4	Функціональна	+/-	+/-	+/-	+/-	+/-	+

Таблиця 2.2 – Типізація

1	Можливість	Мова					
		C#	Go	Java	JavaScript	PHP	Python
2	Статистична типізація	+	+	+	-	-	-
3	Динамічна типізація	+	-	-	+	+	+
4	Явна типізація	+	+	+	-	+/-	+/-
5	Неявна типізація	+/-	-	-	+	+	+

Таблиця 2.3 – Функціональні можливості

1	Можливість	Мова					
		C#	Go	Java	JavaScript	PHP	Python
2	First class functions	+	-	-	+	+	+
3	Анонімні функції	+	-	+	+	+	+

Продовження таблиці 2.3

4	Можливість	Мова					
		C#	Go	Java	JavaScript	PHP	Python
5	Лексичні замикання	+	-	+	+	+	+
6	Часткове застосування	-	-	-	+	-	+
7	Карирування	+	-	-	+	+	+

Таблиця 2.4 – Інше

1	Можливість	Мова					
		C#	Go	Java	JavaScript	PHP	Python
2	Пітримка Unicode в ідентифікаторах	+	+	+	+	+/-	+
3	Значення параметрі за замовчуванням	+	+	-	+	+	+
4	Наявність бібліотек для роботи з графікою і мультимедіа	+	-	+	+	+/-	+

2.1.2 Вибір середовища розробки

Visual Studio Code – обране середовище розробки. Visual Studio Code - це один з найбільш популярних редакторів коду, розроблений корпорацією Microsoft. Він поширюється в безкоштовному доступі і підтримується всіма актуальними операційними системами: Windows, Linux і macOS. VS Code є звичайним текстовим редактор з можливістю підключення різних плагінів, що дає можливість працювати з різними мовами програмування для розробки будь-якого IT-продукту.

Переваги Visual Studio Code:

— VS Code дозволяє розробляти консольні додатки та додатки з графічним інтерфейсом (з підтримкою технології Windows Forms), а також веб-сайти, веб-додатки та веб-служби;

— у редакторі є вбудований відладчик, інструменти для роботи з Git і засоби рефакторинга, навігація по коду, автодоповнення типових конструкцій і контекстної підказки;

— продукт підтримує розробку для платформ ASP.NET і Node.js;

— підтримка великої кількості мов, таких як C ++, C #, Python, PHP, JavaScript та інших;

— Visual Studio Team Services для швидкого тестування та розгортання різних типів додатків;

— зручність роботи з Unity-проектами;

— робота з Mono і Node.js за допомогою вбудованого відладчика;

— підтримка TypeScript;

— публікація створених додатків в Microsoft Azure через сервіс Visual Studio Team Services;

— велика бібліотека шаблонів, готових фрагментів коду і сніпетів з можливістю додавання своїх елементів;

— одночасна робота з декількома проектами (в декількох вікнах) [3].

Отже, Visual Studio Code має необхідні інструменти для комфортного написання коду і усунення дефектів.

2.1.3 Огляд особливостей обраної системи управління базами даних

Для роботи з даними потрібно було обрати базу даних. Було два варіанти: використання Structured Query Language (SQL) чи NoSQL. Обидві моделі підходили для вирішення поставленої задачі. Але було виявлено недолік моделі SQL – це важкість реалізації локалізації. Для реалізації локалізації в моделі SQL потрібно створювати додаткові таблиці для кожної мови. Отже, чим більше мов, тим більше таблиць та зв’язків між ними, що значно зменшить швидкість пошуку та повернення даних. Щодо NoSQL – у цій моделі не потрібні додаткові таблиці для локалізації. Для її реалізації потрібно лише змінити тип поля, якому потрібна локалізація, на json. А у випадку додання нової мови потрібно буде додати властивість в json об’єкт, який буде відповідати новій мови. При цьому швидкість пошуку та повернення даних ніяк не зміниться.

Отже, було обрано нереляційну СУБД – MongoDB. Нереляційна база даних - це база даних, в якій на відміну від більшості традиційних систем баз даних не використовується таблична схема рядків і стовпців. У цих базах даних застосовується модель зберігання, оптимізована під конкретні вимоги типу збережених даних. Наприклад, дані можуть зберігатися як прості пари "ключ - значення", документи JSON.

Бази даних NoSQL спеціально створені для певних моделей даних і мають гнучкі схеми, що дозволяють розробляти сучасні програми. Бази даних NoSQL набули широкого поширення в зв'язку з простотою розробки, функціональністю і продуктивністю при будь-яких масштабах [4].

Для визначення схеми зі строго-типізованими даними треба бажано Mongoose. Це Object-Relational Mapping, тобто об'єктно-реляційне відображення.

Відразу після визначення схеми Mongoose дає можливість створити Model (модель), засновану на певній схемі. Потім модель синхронізується з документом MongoDB за допомогою визначення схеми моделі. А потім можна користуватися різними функціями Mongoose для перевірки, збереження, видалення і запиту ваших даних, використовуючи звичайні функції MongoDB.

2.1.4 Вибір фреймворку

Для платформи Node.js було обрано фреймворк Express. Основна особливість цього фреймворка полягає в тому, що для нього характерний невеликий обсяг базового функціоналу. Всі інші необхідні вам функції потрібно буде добирати за рахунок зовнішніх модулів. По суті, Express в чистому вигляді - це сервер і у нього може не бути жодного модуля. Завдяки такому мінімалізму розробник спочатку отримує в своє розпорядження легкий і швидкий інструмент, який він може розширювати і розвивати. В результаті, цей фреймворк забезпечує розробнику можливість вирішувати будь-які завдання, не обмежуючи його при цьому у виборі засобів.

2.1.5 Вибір середи тестування

Для тестування було обрано фреймворк Jest. Це фреймворк для тестування JavaScript, розроблений для забезпечення впевненості в правильній роботі будь-якого JavaScript коду. Jest добре документований, вимагає мінімальної настройки і може бути розширений, щоб відповідати вашим вимогам. Забезпечуючи тестам унікальний глобальний стан, Jest безпомилково запускає тести в паралельних потоках. Jest розподіляє тести між робочими процесами, щоб максимізувати продуктивність. Консольні повідомлення об'єднуються і виводяться разом з результатами тестування. Файли, що містять тести, ізолюються, а глобальний стан скидається для кожного тесту - це значить, що жоден тест не буде конфліктувати з іншими. Щоб робити все це швидко, Jest в першу чергу запускає провалені раніше тести, і змінює черговість їх запуску, відштовхуючись від того, як довго виконується кожен тест.

2.1.6 Вибір платформи для розгортання проекту

Docker - це програмна платформа для швидкої розробки, тестування і розгортання додатків. Docker упаковує ПЗ в стандартизовані блоки, які називаються контейнерами. Кожен контейнер включає все необхідне для роботи програми: бібліотеки, системні інструменти, код і середу виконання. Завдяки Docker можна швидко розгортати і масштабувати додатки в будь-якому середовищі і зберігати впевненість в тому, що код буде працювати.

В основі роботи Docker лежить стандартизований спосіб виконання коду. Docker - це операційна система для контейнерів. Подібно до того, як віртуальна машина створює віртуальне уявлення апаратного забезпечення сервера (тобто усуває необхідність безпосередньо управляти таким), контейнери створюють віртуальне уявлення серверної операційної системи. Після установки на кожен сервер Docker надає доступ до простих команд, необхідних для збірки, запуску або зупинки контейнерів.

Використання Docker дозволяє швидше і ефективніше доставляти або переміщати код, стандартизує операції, що виконуються додатками, і в цілому економить кошти, оптимізуючи використання ресурсів. Завдяки Docker користувачі отримують об'єкт, який з високою надійністю можна запускати на будь-якій платформі. Простий і зрозумілий синтаксис Docker забезпечує повний контроль над виконуваними операціями. Повсюдне впровадження контейнерів має на увазі доступ до різноманітних інструментів і готовим додатків, які можна використовувати з Docker.

У цьому проекті Docker також корисний тим, що дозволяє розгорнути локально базу даних MongoDB [5].

2.2 Структурна схема програми

На рис. 2.1 зображено структурну схему програмного забезпечення.

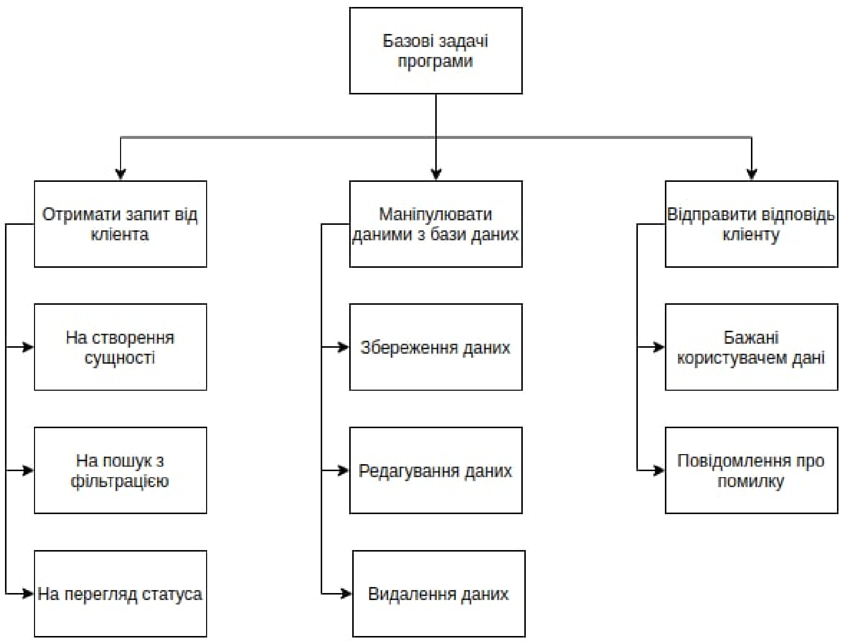

Рисунок 2.1 – Структурна схема програмного забезпечення

2.3 Висновок до розділу 2

Для створення сервісів для підтримки клієнтських застосувань для студентів учасників міжнародної мобільності було обрано якісний та сучасний набір інструментів. Фреймворком було обрано Vue.js, середовищем розробки Visual Studio Code. Також було описано обрану систему управління базами даних – MongoDB 4.4.4, ORM – Mongoose, фреймворк – Express, середу тестування – Jest, платформу для розгортання проекту – Docker..

3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Основні розроблені алгоритми

Основні алгоритми розробленої системи:

— вхід в адміністративну панель з попередньою валідацією (перевірка існування користувача та перевірка хеш-пароля);

— відправка повідомлень на електронну пошту користувача при реєстрації та при відновленні пароля;

— сортування даних при локаліації;

— маніпуляції даними за допомогою адміністративної панелі;

— реєстрація користувача з перевіркою існування елекронної пошти, вказаної користувачем навчального закладу, з хешуванням пароля;

— авторизація користувача з валідацією;

— відновлення пароля;

— пошук навчальних закладів з фільтрацією;

— пагінація даних;

— подача заявок на участь у програмі.

Наприклад, відправка повідомлень на пошту при реєстрації працює наступним чином. Якщо користувач успішно ввів всі необхідні для реєстрації дані та пройшов валідацію, то на вказану ним пошту відправляється лист для підтвердження пошти. Для реалізації відправки було використано спеціальний пакет, який називається «nodemailer». При роботі з ним було використано сервіс gmail, попередньо було створено gmail-акаунт, від імені якого відправляються повідомлення користувачам системи. Для того, щоб «nodemailer» мав доступ до головного gmail-акаунта, йому було надано електронну адресу цього акаунта та двофакторно захищений хеш-пароль. Для того, щоб повідомлення для підтвердження акаунта було відправлено на акаунт нового користувача, потрібна його адреса та також шаблон листа, який буде йому відправлено. Лист було попередньо реалізовано. Після того, як користувач отримує повідомлення на пошту, він натискає на відповідну кнопку, потім акаунт підтверджується. Після цього сервер отримує токен користувача, який потім використовується для роботи.

Процес хешування виконується за допомогою бібліотеки bcryptjs. Для хешування пароля використовується хеш-функція із вхідними параметрами (пароль та saltRounds). SaltRounds є «вартістю» хеша, тобто часом, який було затрачено на створення хеша; чим вище saltRounds, тим довше пароль буде хешуватись і тим важче буде його зламати. Після того, як користувач вводить дані для реєстрації, спеціальна функція хешує введений пароль. Потім функція повертає захешований пароль, і він зберігається в базі даних. Це дає гарантію того, що при зломі бази даних, зловмисники не зможуть отримати доступ до акаунтів користувачів. Потім при авторизації користувач вводить адресу пошти та пароль. Спочатку перевіряється наявність поштової адреси користувача в базі даних. Якщо користувач існує, виконується функція compare, яка приймає 2 параметри (пароль, введений при авторизації, та хеш-пароль, який було попередньо збережено в базі даних). І у випадку, якщо паролі зівпадають, користувачу повертається токен для подальшої роботи.

3.1.1 Авторизація до системи

Неавторизований користувач має обмежений доступ до системи. Для отримання повного доступу необхідно зареєструватись (та ввести особисті дані). Потім для авторизації до системи треба буде використовувати логін та пароль.

3.1.2 Вхід до системи як студент

Студент має доступ до таких функцій:

- авторизація;
- відновлення пароля;
- реєстрація та додання особистих даних;
- редагування налаштувань в особистому акаунті;

— доступ до інформації про навчальні заклади;

— пошук університетів з фільтрацією (мова, факукльтет, спеціальність, локація, період, тип, сезон);

— доступ до інформації про окремий навчальний заклад;

— подача заявки на участь у академічній мобільності.

3.1.3 Вхід до системи як адміністратор

Адміністратор має наступні привілеї:

— доступ до списку корстувачів та їх заявок на участь у мобільності;

— можливість додати дані про спеціальності, факультети, університети;

— можливість видалити дані про спеціальності, факультети, університети;

— можливість редагувати дані про спеціальності, факультети, університети;

— обробка заявок учасників (змінювати статус);

— обробка даних користувачів системи;

— пошук університетів;

— використання фільтрації для пошуку університетів.

3.2 Основні рішення щодо модульного уявлення системи

Проект структуровано з використанням всіх правил для розробки АПІ. Папка database містить в собі папки models (моделі-сутності бази даних), schema (зберігає фрагменти схеми для моделей), view та файл:

— index.js – підключення до бази даних та ініціалізація моделей.

Папка /database/models:

— Application.js – модель сутності подачі заявки на участь;

— Curriculum.js – модель сутності навчальних планів;

— ExchProgram.js – модель сутності навчальних планів для мобільності;

— Faculty.js – модель сутності факультетів;

— Specialty.js – модель сутності степіальностей;

— Syllabus.js – модель сутності предмету з навчальною програмою;

— University.js – модель сутності університетів;

— User.js – модель сутності користувача.

Папка /database/models:

— Language.js – частина схеми для моделей, яка реалізує локалізацію.

Папка /database/view:

— UniversitiesView.js – відображення університетів.

Файли папки seeders:

— curriculum.js – тестові дані навчальних планів;

— drop.js – скрипт для видалення даних з бази даних;

— exchProgram.js – тестові дані навчальних планів академічної мобільності;

— faculty.js – тестові дані факультетів;

— index.js – скрипт для заповлнення бази даних тестовими даними;

— specialty.js – тестові дані спеціальностей;

— syllabus.js – тестові дані предметів;

— university.js – тестові дані університетів.

Папка src містить наступні папки: controllers, dto, helpers, middlewares, routers, services, validators.

Файли папки controllers:

— application.controller.js –контролер для сутності подачі заявки;

— auth.controller.js – контролер для поточного користувача;

— curriculum.controller.js – контролер для навчальних планів;

— faculty.controller.js – контролер для факультетів;

— filterFields.controller.js – контролер для фільтрації;

— specialty.controller.js – контролер для спеціальностей;

— university.controller.js – контролер для університетів.

Файли папки helpers:

- checkLocalization.js – перевірка поточної локалізації;
- constants.js – константні значення;
- errorHandler.js – перехоплювач помилок;
- index.js – основний файл з допоміжними функціями;
- nodemailer.js – скрипт для реалізації електронного повідомлення.

Файли папки middlewares:

- auth.middleware.js – проміжний обробник для даних користувача;
- getWithSearchAndPages.middleware.js – проміжний обробник для валідації даних для пагінації;
- uploadFile.middleware.js – проміжний обробник для збереження файлу;
- verifyToken.middleware.js – проміжний обробник для перевірки існування токена.

Папка routes містить один файл:

- index.js – головний файл з роутами.

Папка services:

- application.service.js – сервіс для роботи з заявками;
- auth.service.js – сервіс для поточного користувача;
- curriculum.service.js – сервіс для навчальних планів;
- faculty.service.js – сервіс для факультетів;
- filterFields.service.js – сервіс для фільтрації;
- specialty.service.js – сервіс для спеціальностей;
- university.service.js – сервіс для університетів.

Папка validators:

- auth.validator.js – валідація даних користувача;
- Validator.js – клас, який реалізує валідацію.

Папка test містить наступні папки: auth, curriculum, faculty, filterFields, specialty, university. Файли папки auth:

- forgot.test.js – тести для кінцевої точки процесу «Забув пароль»;
- login.test.js – тести для кінцевої точки авторизації;
- register.test.js – тести для кінцевої точки реєстрації;

— reset.test.js – тести для кінцевої точки відновлення пароля;

— verify.test.js – тести для кінцевої точки підтвердження електронної пошти.

В папці curriculum є один файл:

— getCurriculumOfUFS.test.js – тести для кінцевої точки отримання навчальних планів.

Папка faculty:

— universityFaculty.test.js – тести для кінцевої точки отримання факультетів.

В папці filterFields є:

— universityOfMobility.test.js – тести для кінцевої точки отримання університетів, що беруть участь у мобільності за заданими фільтрами.

Папка specialty:

— universityAndFacultySpecialty.test.js – тести для кінцевої точки отримання спеціальностей.

Папка university:

— get.test.js – тести для кінцевої точки отримання університетів;

— getById.test.js – тести для кінцевої точки отримання університету за ID;

— getOfMobility.test.js – тести для кінцевої точки отримання університетів, що беруть участь у мобільності.

Також є файли:

— docker-compose.yml – правила для установки компонентів для докера;

— Dockerfile – правила створення контейнера;

— package.json – файл, який реалізує структуру проекту;

— server.js – головний файл, який запускає сервер.

3.3 Організація даних

Для маніпулювання даними використовується Mongoose, який відповідає за отримання даних з колекцій MongoDB. Кожна колекція відповідає за свою власну таблицю.

3.3.1 Структура таблиць бази даних

В табл. 3.1 описані основні поля таблиці Users. Вона призначена для зберігання особистих даниих користувачів (логін, пароль, ім’я, день народження, університет, заявки), токенів та інформації щодо верифікації пошти.

Таблиця 3.1 — Таблиця Users

1	Назва поля	Тип поля	Опис
2	ID	Integer	Ключове поле таблиці
3	fullName	Text	Ім’я та прізвище
4	email	Text	Пошта
5	password	Text	Пароль
6	birthday	Date	Дата народження
7	token	Text	
8	isEmailVerified	Boolean	Перевірка верифікації пошти
9	university	Json	Університет
10	applications	Array	Заявки

В табл. 3.2 описані основні поля таблиці Universities. Ця таблиця слугує для опису детальної інформації про університети (назва, опис, локація, сайт, факультети).

Таблиця 3.2 — Таблиця Universities

1	Назва поля	Тип поля	Опис поля
2	ID	Integer	Ключове поле таблиці
3	name	Text	Назва університету
4	description	Text	Опис університету
5	location	Text	Місцезнаходження
6	website	Text	Веб-сайт
7	faculties	Array	Факультети

В табл. 3.3 описані основні поля таблиці Faculties. Вона призначена для зберігання спеціальностей факультетів.

Таблиця 3.3 — Таблиця Faculties

1	Назва поля	Тип поля	Опис
2	ID	Integer	Ключове поле таблиці
3	name	Text	Назва факультету
4	specialty	Array	Спеціальність

В табл. 3.4 описані основні поля таблиці Specialty. В цій таблиці зберігається основна інформація щодо спеціальності, а саме: навчальний план, програма мобільності.

Таблиця 3.4 — Таблиця Specialty

1	Назва поля	Тип поля	Опис поля
2	ID	Integer	Ключове поле таблиці
3	name	Text	Назва спеціальності
4	curriculums	Array	Навчальний план
5	exchProgrammes	Array	Програма мобільності

В табл. 3.5 описані основні поля таблиці Applications. В цій таблиці зберігається інформація щодо заявки на участь: навчальний план та угода про навчання.

Таблиця 3.5 — Таблиця Applications

1	Назва поля	Тип поля	Опис
2	ID	Integer	Ключове поле таблиці
3	curriculum	Json	Навчальний план
4	learningAgreement	Json	Угода про навчання

В табл. 3.6 описані основні поля таблиці ExchProgrammes. В цій таблиці зберігається основна інформація щодо програм мобільності, а саме: назва, мова, тип навчання, сезон, період, кількість кредитів, ім'я та пошта координатора, предмети.

Таблиця 3.6 — Таблиця ExchProgrammes

1	Назва поля	Тип поля	Опис поля
2	ID	Integer	Ключове поле таблиці
3	name	Text	Назва програми
4	language	Text	Мова навчання
5	type	Text	Віртуальне/фактичне навчання
6	season	Text	Весна/осінь/осінь-весна
7	period	Text	Семестр/декілька тижнів
8	ectsCredits	Integer	Кількість кредитів
9	coordinatorFullName	Text	Ім'я координатора
10	coordinatorEmail	Text	Пошта координатора
11	syllabuses	Array	Предмети

В табл. 3.7 описані основні поля таблиці Curriculums. В цій таблиці зберігається основна інформація щодо навчальних планів, а саме: семестр та предмети.

Таблиця 3.7 — Таблиця Curriculums

1	Назва поля	Тип поля	Опис поля
2	ID	Integer	Ключове поле таблиці

Продовження таблиці 3.7

3	semester	Integer	Семестр
4	syllabuses	Array	Предмети

В табл. 3.8 описані основні поля таблиці Syllabuses. В цій таблиці зберігається інформація щодо предметів: назва, опис, код, кількість кредитів, семестр.

Таблиця 3.8 — Таблиця Syllabuses

1	Назва поля	Тип поля	Опис поля
2	ID	Integer	Ключове поле таблиці
3	name	Text	Назва
4	description	Text	Опис
5	code	Text	Код
6	ects	Text	Кількість кредитів
7	semester	Integer	Семестр

Для повноцінної роботи програми, була розроблена БД, ER діаграма якої зображена на рис 3.1.

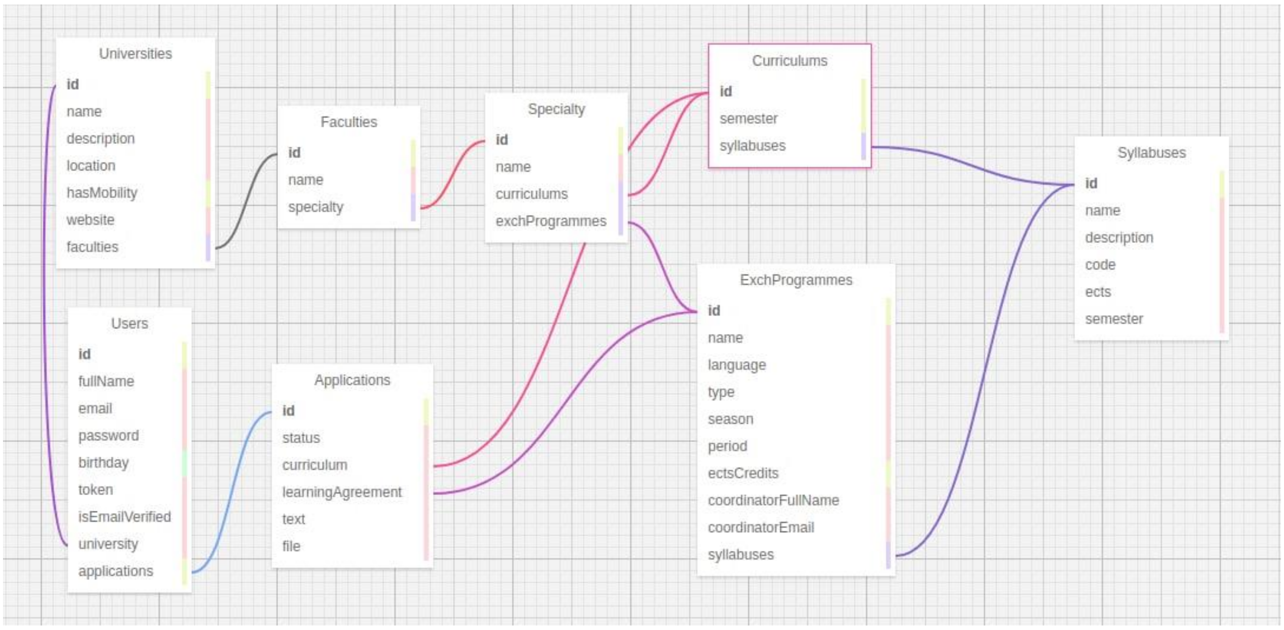

Рисунок 3.1 – ER діаграма БД

Виходячи з рисунку, було створено вісім таблиць, серед яких:

- Universities – включає в себе список існуючих університетів;
- Users – список всіх користувачів;
- Faculties – список всіх факультетів;
- Applications – таблиця з заявками користувачів;
- Specialty – список спеціальностей;
- Curriculums – список навчальних планів;
- ExchProgrammes – список програм мобільності;
- Syllabuses – список предметів з навчальною програмою.

3.3.2 Введення та виведення даних

Введення даних виконується за допомогою загальних елементів інтерфейсу, комп'ютеної миші та клавіатури. Виведення даних виконується за допомогою монітору комп'ютера.

3.4 Захищення даних

Для захищення даних кожен, хто має доступ до системи, матиме свій логін і пароль. Пароль має бути не менше 6 символів та містити цифри та літери.

Після введення даних користувачем при реєстрації пароль хешується, а потім отриманий хеш-пароль зберігається в базі даних. Для реалізації цього було використано бібліотеку bcryptjs. Незареєстрований користувач має обмежений доступ до системи.

Усі паролі зберігаються у базі даних, до яких матиме доступ тільки адміністратор системи.

3.5 Висновок до розділу 3

Було розроблено основні алгоритми програми, які будуть використовуватись для реалізації усіх поставлених задач. Основними алгоритмами є сортування даних при локалізації, хешування пароля, процес відправки повідомлень на електронну пошту користувача та маніпуляції даними за допомогою адміністративної панелі.

Було описано та заповнено всі таблиці, які потрібні для працездатності системи. Згенеровано ER діаграму даних, яка поєднує всі таблиці і реалізує логіку системи.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення й умови застосування програми

Розроблена система призначена для забезпечення гнучкого механізму подачі та обробки заявок на участь в академічній мобільності.

Розроблена адміністративна панель дозволяє виконувати маніпуляції з даними університетів та заявок.

Система надає студенту наступні можливості:

- авторизація;
- відновлення пароля;
- реєстрація та додання особистих даних;
- редагування налаштувань в особистому профілю;
- доступ до інформації про окремий навчальний заклад;
- пошук університетів з фільтрацією;
- подача заявки на участь у програмі академнічної мобільності.

4.2 Характеристики програми

До складу технічних засобів має входити персональна електронна обчислювальна машина (ПЕОМ), яка виконує роль сервера, який містить:

- процесор Intel Core i3 1.4Hz, не менше;
- оперативну пам'ять об'ємом 8 Гігабайт, не менше;
- HDD 100 Гігабайт, не менше;
- операційну систему Ubuntu 18.xx;
- Node 14.x;
- Npm 6.14.x;
- платформу для розгортання проекту Docker 19.3.x;
- базу даних MongoDB 4.4.x.

4.3 Звернення до програми

Для звернення до програми потрібно виконати наступні кроки. Із кореневої папки необхідно виконати команду npm install для встановлення необхідних пакетів бекенду. Потім потрібно заповнити даними для налаштувань файл «.env». Далі необхідно запустити Docker Container, який ініціалізує базу даних. Потім за допомогою консолі запустити АПІ.

4.4 Виняткові ситуації

Були розроблені спеціальні функції для обробки виняткових ситуацій. Наприклад, при реєстрації користувач має пройти валідацію, на рис. 4.1 зображено результат роботи кінцевої точки для реєстрації, де видно, що кожне поле має власне правило валідації. Для перевірки використовуються middleware, які перевіряють валідність тих даних, для перевірки яких не потрібне явне звернення до бази даних. У випадку, якщо валідація middleware пройшла успішно, переходимо до наступного етапу валідації, який знаходиться в сервісній частині програми. Ця валідація пов’язана з базою даних, вона перевіряє наявність наступних значень в базі даних: існування користувача, університетів, факультетів, спеціальностей, навчальних планів.

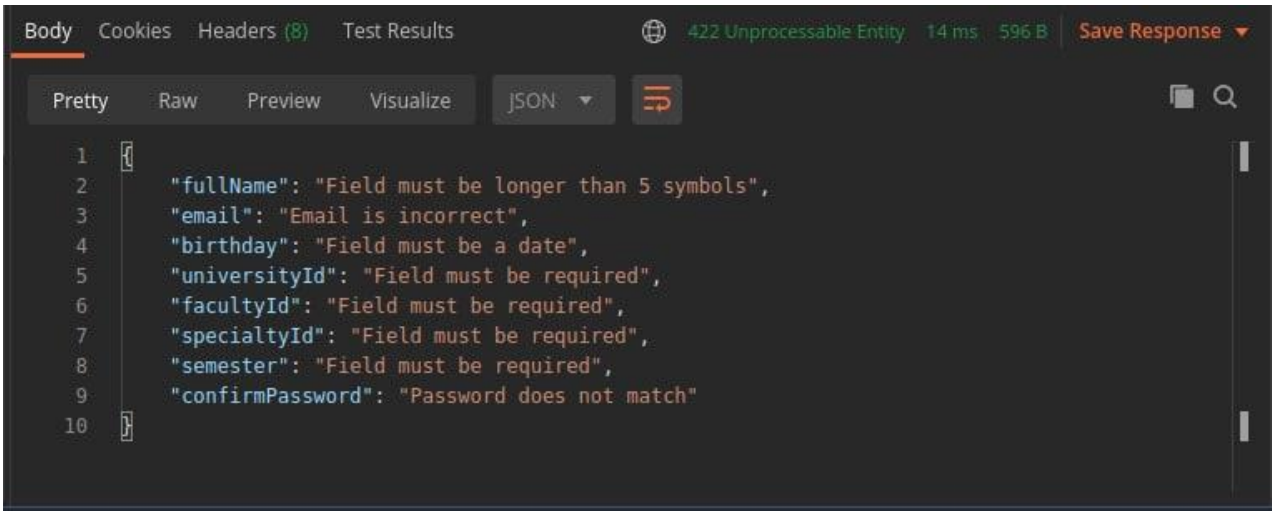

Рисунок 4.1 – Повідомлення про помилки при реєстрації

Інший middleware перевіряє наявність та справжність токену (рис. 4.2). У випадку, якщо токен коректний, токен розшифровується, та контролеру передається ID користувача для подальшої взаємодії із системою. А якзо він некоректний, то middleware відправляє повідомлення про помилку.

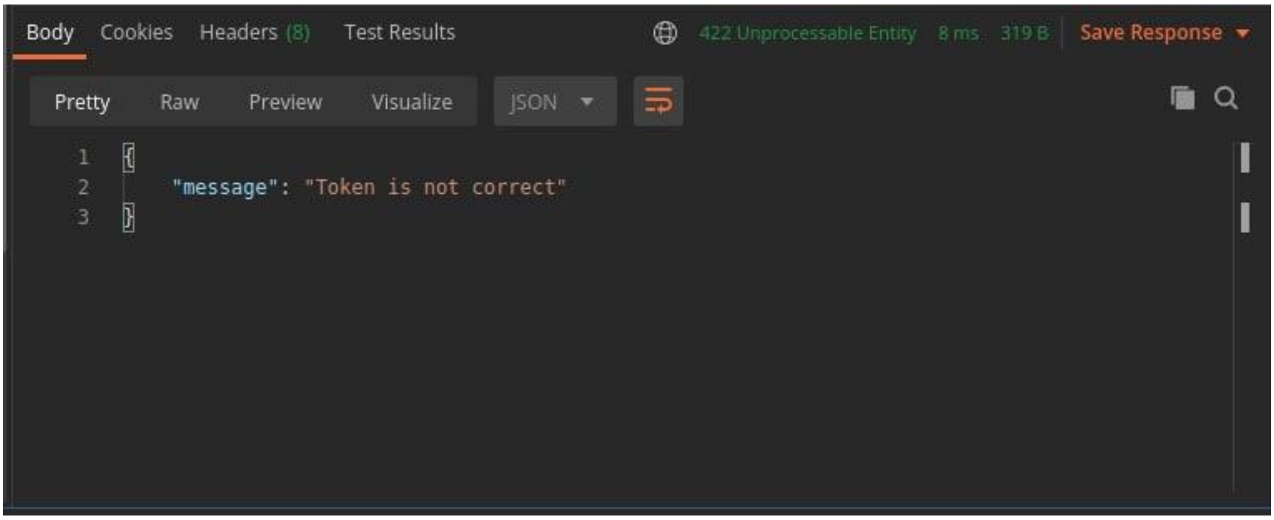

Рисунок 4.2 – Повідомлення про токен

У ендпоінтів, які реалізують функціонал пагінації, є проміжний обробник, який перевіряє значення параметріві пагінації (поточна сторінка та кількість записів на одній сторінці). У middleware прописано правило, що поточна сторінка не може бути негативною та не має бути символом в полі запиту. А у кількості елементів на одній сторінці такі ж обмеження, як і у значення поточної сторінки, є тільки одна відмінність – кількість елементів на одній сторінці має діапазон від 1 до 50 (рис. 4.3). Це зроблено для того, щоб убезпечити роботу АПІ у випадку, якщо користувач захоче одразу отримать дуже велику кількість даних, що потім може зламати роботу АПІ.

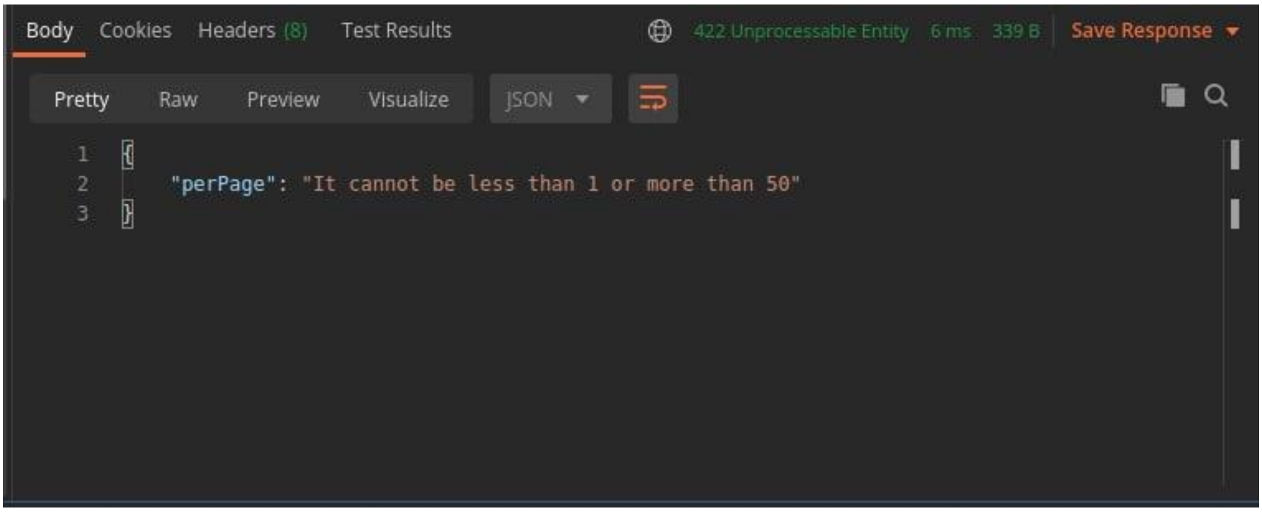

Рисунок 4.3 – Повідомлення про кількість елементів на одній сторінці

4.5 Висновок до розділу 4

У даному розділі було розглянуто основні призначення та умови застосування програми. Були визначені характеристики програми, яка була розроблена за допомогою платформи Node.js та фреймворку Express.

Також були розглянуті виняткові ситуації які можуть виникнути при взаємодії з системою.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи бакалавра, було проведено аналіз існуючих систем для підримки клієнтських застосувань для міжнародної мобільності. Проведений аналіз показав, що подібні існуючі системи мають недоліки, які впливають на ефективність та продуктивність. Тому було розроблено сервіси, які усувають головні недоліки існуючих систем.

Було складено технічне завдання, в якому зазначена мета розробки програми, призначення, основні вимоги до програми, порядок функціонування системи.

Розроблена система дозволює виконувати:

— авторизацію адміністратора в адміністративну панель;
— обробку заявок учасників адміністратором;
— доступ до списку користувачів для адміністратора;
— маніпуляції даними про навчальні заклади адміністратором;
— авторизацію користувача;
— відновлення пароля;
— реєстрацію та додання особистих даних;
— редагування налаштувань в особистому акаунті;
— доступ інформації про навчальні заклади;
— пошук навчальних закладів з фільтрацією;
— подачу заявки на участь у мобільності.

Система може бути задіяна в будь-якому клієнтському застосунку для студентів учасників міжнародної мобільності. Програма повністю дороблена та готова до використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Service Oriented Architechture [Electronic resource]. – Access mode: https://piter-soft.ru/knowledge/glossary/process/soa-system.html

2. Преимущества Node.js. [Электрон. ресурс]. – Режим доступа: https://artjoker.ua/ru/blog/v-chem-preimushchestva-nodejs/

3. Редактор Visual Studio Code. [Электрон. ресурс]. – Режим доступа: https://timeweb.com/ru/community/articles/chto-takoe-visual-studio-code

4. Базы данных NoSQL. [Электрон. ресурс]. – Режим доступа: https://aws.amazon.com/ru/nosql/

5. Docker. [Электрон. ресурс]. – Режим доступа: https://aws.amazon.com/ru/docker/

ДОДАТОК А

Текст програми

А.1 Файл /database/models/Application.js

```
const { Schema, model, Types } = require("mongoose");
const Language = require("../schema/Language");

const schema = new Schema({

  status: {
    type: Language.schema,
    required: true,
  },
  curriculum: {
    type: Object,
    required: true,
  },
  learningAgreement: {
    type: Object,
    required: true,
  },
  text: {
    type: String,
    required: false,
  },
  file: {
    type: String,
    required: false,
  },
});

module.exports = model("Application", schema);
```

А.2 Файл /database/models/Curriculum.js

```
const { Schema, model, Types } = require("mongoose");

const schema = new Schema({

  semester: {
    type: Number,
    required: true,
  },

  syllabuses: [{ type: Types.ObjectId, ref: "Syllabus" }],
});

module.exports = model("Curriculum", schema);
```

А.3 Файл /database/models/ExchProgram.js

```
const { Schema, model, Types } = require("mongoose");
const Language = require("../schema/Language");

const schema = new Schema({
```

```
  name: {
    type: Language.schema,
    required: true,
  },
  language: {
    type: Language.schema,
    required: true,
  },
  type: {
    type: Language.schema,
    require: true,
  },
  season: {
    type: Language.schema,
    required: true,
  },
  period: {
    type: Language.schema,
    require: true,
  },
  ectsCredits: {
    type: Number,
    required: true,
  },
  coordinatorFullName: {
    type: String,
    required: false,
  },
  coordinatorEmail: {
    type: String,
    required: false,
  },
  syllabuses: [{ type: Types.ObjectId, ref: "Syllabus" }],
});

module.exports = model("ExchProgram", schema);
```

А.3 Файл /database/models/Faculty.js

```
const { Schema, model, Types } = require("mongoose");
const Language = require("../schema/Language");

const schema = new Schema({

  name: {
    type: Language.schema,
    required: true,
  },
  specialties: [{ type: Types.ObjectId, ref: "Specialty" }],

});

module.exports = model("Faculty", schema);
```

А.4 Файл /database/models/Specialty.js

```
const { Schema, model, Types } = require("mongoose");
```

```
const Language = require("../schema/Language");

const schema = new Schema({

  name: {
    type: Language.schema,
    required: true,
  },
  curriculums: [{ type: Types.ObjectId, ref: "Curriculum" }],
  exchProgrammes: [{ type: Types.ObjectId, ref: "ExchProgram" }],

});

module.exports = model("Specialty", schema);
```

А.5 Файл /database/models/Syllabus.js

```
const { Schema, model } = require("mongoose");
const Language = require("../schema/Language");

const schema = new Schema({

  name: {
    type: Language.schema,
    required: true,
  },
  description: {
    type: Language.schema,
    required: true,
  },
  code: {
    type: String,
    required: true,
  },
  ects: {
    type: Number,
    required: true,
  },
  semester: {
    type: Language.schema,
    required: true,
  },
});

module.exports = model("Syllabus", schema);
```

А.6 Файл /database/models/University.js

```
const { Schema, model, Types } = require("mongoose");
const Language = require("../schema/Language");

const schema = new Schema({

  name: {
    type: Language.schema,
```

```
    required: true,
  },

  description: {

    type: Language.schema,
    required: false,
  },
  location: {

    type: Language.schema,
    required: true,
  },
  hasMobility: {

    type: Boolean,
    require: true,
    default: false,
  },
  website: {
    type: String,
    required: true,
  },

  faculties: [{ type: Types.ObjectId, ref: "Faculty" }],
});

module.exports = model("University", schema);
```

А.7 Файл /database/models/User.js

```
const { Schema, model, Types } = require("mongoose");
const Language = require("../schema/Language");

const schema = new Schema({

  fullName: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  birthday: {
    type: Date,
    required: true,
  },
  token: {
    type: String,
    required: false,
  },
  isEmailVerified: {
    type: Boolean,
    required: true,
```

```
    default: false,
  },

  university: {
    type: Object,
    required: true,
  },
  applications: [{ type: Types.ObjectId, ref: "Application" }],
});

module.exports = model("User", schema);
```

А.8 Файл /database/schema/Language.js

```
const { Schema, model } = require("mongoose");

const schema = new Schema(

  {
    en: {
      type: {},
      required: true,
    },
    ua: {
      type: {},
    },
  },
  { _id: false }
);

module.exports = model("Language", schema);
```

А.9 Файл /database/view/UniversitiesView.js

```
const { Schema, model, Types } = require("mongoose");
const Language = require("../schema/Language");

const schema = new Schema(
  {
    name: {
      type: Language.schema,
      required: true,
      unique: true,
    },
    description: {
      type: Language.schema,
      required: false,
    },
    location: {
      type: Language.schema,
      required: true,
    },
    hasMobility: {
      type: Boolean,
      require: true,
      default: false,
    },
```

```
website: {
  type: String,
  required: true,
},
faculties: [

  {
    name: {

      type: Language.schema,
      required: true,
      unique: true,
    },
    specialties: [

      {
        name: {
          type: Language.schema,
          required: true,
          unique: true,
        },

        curriculums: [{ type: Types.ObjectId, ref: "Curriculum" }],
        exchProgrammes: [
          {
            name: {
              type: Language.schema,
              required: true,
            },
            language: {
              type: Language.schema,
              required: true,
            },
            type: {
              type: Language.schema,
              require: true,
            },
            season: {
              type: Language.schema,
              required: true,
            },
            period: {
              type: Language.schema,
              require: true,
            },
            ectsCredits: {
              type: Number,
              required: true,
            },
            coordinatorFullName: {
              type: String,
              required: false,
            },
            coordinatorEmail: {
              type: String,
              required: false,
            },
            syllabuses: [
              {
                name: {
                  type: Language.schema,
                  required: true,
                },
                code: {
```

```
                    type: String,
                    required: true,
                  },
                  ects: {
                    type: Number,
                    required: true,
                  },
                  semester: {
                    type: Language.schema,
                    required: true,
                  },
                },
              ],
            },
          ],
        },
      ],
    },
  ],
  },
  { collection: "universitiesView" }
);

module.exports = model("UniversitiesView", schema);
```

A.10 Файл /database/index.js

```
const mongoose = require("mongoose");
require("dotenv").config();

const Application = require("./models/Application");
const Curriculum = require("./models/Curriculum");
const ExchProgram = require("./models/ExchProgram");
const Faculty = require("./models/Faculty");
const Specialty = require("./models/Specialty");
const Syllabus = require("./models/Syllabus");
const University = require("./models/University");
const User = require("./models/User");

const UniversitiesView = require("./view/UniversitiesView");

const connect = () => {
  return mongoose.connect(

`mongodb://${process.env.MONGO_HOST}:${process.env.MONGO_PORT}/${process.env.MONGO
_DATABASE}`,
    {

      useUnifiedTopology: true,
      useNewUrlParser: true,
      useFindAndModify: false,
      useCreateIndex: true,

    }
  );
};

const disconnect = () => {
  return mongoose.disconnect();
};
```

```
module.exports = {
  mongoose,
  connect,
  disconnect,
  Application,
  Curriculum,
  ExchProgram,
  Faculty,
  Specialty,
  Syllabus,
  University,
  User,

  UniversitiesView,
};
```

А.11 Файл /src/controllers/application.controller.js

```
const applicationService = require("../services/application.service");
const { handleError } = require("../helpers/errorHandler");

const create = async (req, res) => {
  const localization = require(`../../client/localization/${req.local}`);

  try {
    await applicationService.create(req.userId, req.body, req.file);
    res.status(201).send({ m: "ok" });
  } catch (error) {
    handleError(error, res);
  }
};

module.exports = {
  create,
};
```

А.12 Файл /src/controllers/auth.controller.js

```
const authService = require("../services/auth.service");
const { handleError } = require("../helpers/errorHandler");

const register = async (req, res) => {
  const localization = require(`../../client/localization/${req.local}`);

  try {

    await authService.register(req.body, req.local);
    res.status(201).send({
      message: localization.auth.register,
    });
  } catch (error) {
    handleError(error, res);
  }
};

const verify = async (req, res) => {

  try {
```

```
    const token = await authService.verify(req.params.token, req.local);
    res.status(200).send(token);

  } catch (error) {
    handleError(error, res);
  }
};

const login = async (req, res) => {
  try {
    const token = await authService.login(req.body, req.local);
    res.status(200).send(token);
  } catch (error) {
    handleError(error, res);
  }
};

const forgot = async (req, res) => {
  const localization = require(`../../client/localization/${req.local}`);

  try {

    await authService.forgot(req.body, req.local);
    res.status(200).send({
      message: localization.auth.forgot,
    });
  } catch (error) {
    handleError(error, res);

  }
};

const reset = async (req, res) => {
  const localization = require(`../../client/localization/${req.local}`);

  try {
    await authService.reset(req.params.token, req.body, req.local);
    res.status(200).send({
      message: localization.auth.reset,
    });

  } catch (error) {
    handleError(error, res);
  }
};

const me = async (req, res) => {
  try {
    const user = await authService.me(req.userId);
    res.status(200).send(user);
  } catch (error) {
    handleError(error, res);
  }
};

module.exports = {
  register,
  verify,
  login,
  forgot,
  reset,
  me,
};
```

А.13 Файл /src/controllers/curriculum.controller.js

```
const curriculumService = require("../services/curriculum.service");
const { handleError } = require("../helpers/errorHandler");

const getCurriculumOfUFS = async (req, res) => {

  try {

    const curriculum = await curriculumService.getCurriculumOfUFS(
      req.params.universityId,
      req.params.facultyId,
      req.params.specialtyId

    );
    res.status(200).send(curriculum);
  } catch (error) {
    handleError(error, res);
  }
};

module.exports = {
  getCurriculumOfUFS,
};
```

А.14 Файл /src/controllers/faculty.controller.js

```
const facultyService = require("../services/faculty.service");
const { handleError } = require("../helpers/errorHandler");

const getUniversitysFaculty = async (req, res) => {
  try {

    const faculty = await facultyService.getUniversitysFaculty(
      req.params.universityId,
      req.query.search,
      req.query.page,
      req.query.perPage,
      req.local
    );
    res.status(200).send(faculty);
  } catch (error) {
    handleError(error, res);
  }
};

module.exports = {

  getUniversitysFaculty,
};
```

А.15 Файл /src/controllers/filterFields.controller.js

```
const filterFieldsService = require("../services/filterFields.service");
```

```
const { handleError } = require("../helpers/errorHandler");

const universityOfMobility = async (req, res) => {

  try {
    res
      .status(200)
      .send(await filterFieldsService.getUniversityOfMobility(req.local));
  } catch (error) {
    handleError(error, res);
  }
};

module.exports = {
  universityOfMobility,
};
```

А.16 Файл /src/controllers/specialty.controller.js

```
const specialtyService = require("../services/specialty.service");
const { handleError } = require("../helpers/errorHandler");

const getSpecialtyOfFacultyAndUniversity = async (req, res) => {
  try {

    const specialty = await specialtyService.getSpecialtyOfFacultyAndUniversity(
      req.params.universityId,
      req.params.facultyId,
      req.query.search,
      req.query.page,
      req.query.perPage,
      req.local

    );
    res.status(200).send(specialty);
  } catch (error) {
    handleError(error, res);
  }
};

module.exports = {
  getSpecialtyOfFacultyAndUniversity,
};
```

А.17 Файл /src/controllers/university.controller.js

```
const universityService = require("../services/university.service");
const { handleError } = require("../helpers/errorHandler");

const get = async (req, res) => {
  try {
    const universities = await universityService.get(
      req.query.search,
      req.query.page,
      req.query.perPage,
```

```
      req.local
    );

    res.status(200).send(universities);
  } catch (error) {
    handleError(error, res);
  }
};

const getById = async (req, res) => {
  try {

    const university = await universityService.getById(
      req.params.universityId,
      req.local
    );
    res.status(200).send(university);
  } catch (error) {

    handleError(error, res);
  }
};

const getOfMobility = async (req, res) => {
  try {
    const universities = await universityService.getOfMobility(
      req.query.search,
      req.query.page,
      req.query.perPage,
      req.query,
      req.local
    );
    res.status(200).send(universities);
  } catch (error) {

    handleError(error, res);
  }
};

module.exports = {
  get,
  getById,
  getOfMobility,
};
```

А.18 Файл /src/middlewares/auth.middleware.js

```
const authValid = require("../validators/auth.validator");

const register = (req, res, next) => {
  const result = authValid.register(req.body, req.local);

  if (result) {
    res.status(422).send(result);
  } else {
    next();
  }
};

const login = (req, res, next) => {
```

```
  const result = authValid.login(req.body, req.local);

  if (result) {
    res.status(422).send(result);
  } else {
    next();
  }
};

const forgot = (req, res, next) => {

  const result = authValid.forgot(req.body, req.local);

  if (result) {
    res.status(422).send(result);
  } else {
    next();
  }
};

const reset = (req, res, next) => {
  const result = authValid.reset(req.body, req.local);

  if (result) {
    res.status(422).send(result);
  } else {
    next();
  }
};

module.exports = {
  register,
  login,
  forgot,
  reset,
};
```

А.19 Файл /src/middlewares/getWithSearchAndPages.middleware.js

```
const get = async (req, res, next) => {
  const localization = require(`../../client/localization/${req.local}`);

  if (
    req.query.page < 1 ||
    (typeof req.query.page !== "undefined" && isNaN(req.query.page))
  ) {
    return res.status(404).send();
  }

  if (typeof req.query.perPage !== "undefined" && isNaN(req.query.perPage)) {
    return res.status(404).send();
  }

  if (req.query.perPage < 1 || req.query.perPage > 50) {
    return res.status(422).send({ perPage: localization.validator.perPage });
  }

  req.query.page = req.query.page && +req.query.page;

  req.query.perPage = req.query.perPage && +req.query.perPage;
```

```
  next();
};

module.exports = {
  get,
};
```

А.20 Файл /src/middlewares/uploadFile.middleware.js

```
const multer = require("multer");

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "public/uploads/");
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + "-" + file.originalname);
  },
});

module.exports = multer({ storage: storage });
```

А.21 Файл /src/middlewares/verifyToken.middleware/.js

```
const jwt = require("jsonwebtoken");

module.exports = (req, res, next) => {
  const localization = require(`../../client/localization/${req.local}`);

  try {
    const token = req.headers.authorization?.split(" ")[1];

    const decoded = jwt.verify(token, process.env.SECRET);

    req.userId = decoded.id;

    next();
  } catch (error) {
    res.status(422).send({
      message: localization.validator.noVerifyToken,
    });
  }
};
```

А.22 Файл /src/routers/index.js

```
const express = require("express");

const router = express.Router();
```

```
const authMiddleware = require("../middlewares/auth.middleware");
const getWithSearchAndPagesMiddleware =
require("../middlewares/getWithSearchAndPages.middleware");
const auploadFileMiddleware = require("../middlewares/uploadFile.middleware");
const verifyTokenMiddleware = require("../middlewares/verifyToken.middleware");


const applicationController = require("../controllers/application.controller");
const authController = require("../controllers/auth.controller");
const universityController = require("../controllers/university.controller");
const facultyController = require("../controllers/faculty.controller");
const specialtyController = require("../controllers/specialty.controller");
const curriculumController = require("../controllers/curriculum.controller");
const filterFieldsController = require("../controllers/filterFields.controller");

router.post("/auth/register", authMiddleware.register, authController.register);
router.put("/auth/verify/:token", authController.verify);
router.post("/auth/login", authMiddleware.login, authController.login);
router.post("/auth/forgot", authMiddleware.forgot, authController.forgot);
router.post("/auth/reset/:token", authMiddleware.reset, authController.reset);
router.get("/auth/me", verifyTokenMiddleware, authController.me);

router.post(
  "/application",
  verifyTokenMiddleware,
  auploadFileMiddleware.single("file"),
  applicationController.create
);

router.get(
  "/university",
  getWithSearchAndPagesMiddleware.get,
  universityController.get
);
router.get("/university/:universityId", universityController.getById);
router.get(
  "/university-of-mobility",
  getWithSearchAndPagesMiddleware.get,
  universityController.getOfMobility
);

router.get(
  "/university/:universityId/faculty",
  getWithSearchAndPagesMiddleware.get,
  facultyController.getUniversitysFaculty
);

router.get(
  "/university/:universityId/faculty/:facultyId/specialty",
  getWithSearchAndPagesMiddleware.get,
  specialtyController.getSpecialtyOfFacultyAndUniversity
);

router.get(

"/university/:universityId/faculty/:facultyId/specialty/:specialtyId/curriculum",
  curriculumController.getCurriculumOfUFS
);

router.get(
  "/filter-fields/university",
  filterFieldsController.universityOfMobility
);
```

```
module.exports = router;
```

А.23 Файл /src/services/application.service.js

```
const {
  User,
  Application,
  University,
  Faculty,
  Specialty,
  ExchProgram,
  Syllabus,
} = require("../../database");
const { applicationStatus } = require("../helpers/constants");
const jwt = require("jsonwebtoken");

const create = async (userId, data, file) => {
  try {
    const user = await User.findById(userId);

    if (!user) {
      throw new ErrorHandler(422, {
        message: localization.validator.userNoFound,
      });
    }

    const university = await University.findById(
      data.idForApply.universityId
    ).lean();
    const faculty = await Faculty.findById(data.idForApply.facultyId).lean();
    const specialty = await Specialty.findById(
      data.idForApply.specialtyId
    ).lean();
    const exchProgramm = await ExchProgram.findById(
      data.idForApply.exchProgrammId
    ).lean();
    const syllabuses = await Syllabus.find({

      _id: {
        $in: data.chooseSyllabuses,
      },
    });

    delete university.faculties;
    delete faculty.specialties;
    delete specialty.curriculums;
    delete specialty.exchProgrammes;
    exchProgramm.syllabuses = [];

    university.faculty = faculty;
    university.faculty.specialty = specialty;
    university.faculty.specialty.exchProgramm = exchProgramm;
    university.faculty.specialty.exchProgramm.syllabuses = syllabuses;

    const application = await Application.create({
      status: applicationStatus.created,
      curriculum: user.university,
      learningAgreement: university,
      text: data?.text || null,
```

```
      file: file?.path || null,
    });

    user.applications.push(application);
    await user.save();
    return;
  } catch (error) {
    throw error;
  }
};

module.exports = {
  create,
};
```

А.24 Файл /src/services/auth.service.js

```
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");

const {
  User,
  University,
  Faculty,
  Specialty,
  Curriculum,
} = require("../../database");
const { ErrorHandler } = require("../helpers/errorHandler");
const mail = require("../helpers/nodemailer");
const createUserDto = require("../dto/createUser.dto");

const register = async (data, local) => {
  const localization = require(`../../client/localization/${local}`);

  try {
    const user = await User.findOne({ email: data.email });

    if (user) {
      throw new ErrorHandler(422, {
        email: localization.validator.emailNoFree,
      });
    }

    const university = await University.findById(data.universityId);

    if (!university) {
      throw new ErrorHandler(422, {
        universityId: localization.validator.notUniversity,
      });
    }

    const faculty = await University.findOne({
      _id: data.universityId,
      faculties: data.facultyId,
    });

    if (!faculty) {
      throw new ErrorHandler(422, {
        facultyId: localization.validator.notFaculty,
```

```
      });
    }

    const specialty = await Faculty.findOne({
      _id: data.facultyId,
      specialties: data.specialtyId,
    });

    if (!specialty) {
      throw new ErrorHandler(422, {
        specialtyId: localization.validator.notSpecialty,
      });
    }

    let curriculum = await Specialty.findOne({
      _id: data.specialtyId,
    })
      .populate("curriculums")
      .lean();

    curriculum = curriculum.curriculums.find(
      (el) => el.semester == data.semester
    );

    if (!curriculum) {
      throw new ErrorHandler(422, {
        semester: localization.validator.notSemester,
      });
    }

    data.password = await bcrypt.hash(data.password, 12);
    data.university = await getFullInfoForUniversity(data);

    const newUser = await User.create(createUserDto(data, local));
    const token = await jwt.sign({ id: newUser.id }, process.env.SECRET);
    newUser.token = token;
    await newUser.save();
    mail(
      data.email,
      localization.mail.title,
      localization.mail.verificationHTML(
        `${process.env.CLIENT_URL}/verify/${token}`
      )
    );
    return;
  } catch (error) {
    throw error;
  }
};

const getFullInfoForUniversity = async ({

  universityId,
  facultyId,
  specialtyId,
  semester,
}) => {
  const university = await University.findById(universityId).lean();
  const faculty = await Faculty.findById(facultyId).lean();
  const specialty = await Specialty.findById(specialtyId).lean();
  let curriculum = await Specialty.findOne({
    _id: specialtyId,
  })
    .populate("curriculums")
```

```
    .lean();

  curriculum = curriculum.curriculums.find((el) => el.semester == semester);

  curriculum = await Curriculum.findById(curriculum._id)
    .populate("syllabuses")
    .lean();

  delete university.faculties;
  delete faculty.specialties;
  delete specialty.curriculums;

  university.faculty = faculty;
  university.faculty.specialty = specialty;
  university.faculty.specialty.curriculum = curriculum;

  return university;
};

const verify = async (token, local) => {

  const localization = require(`../../client/localization/${local}`);

  try {
    const decoded = jwt.verify(token, process.env.SECRET);

    if (!decoded.id) {
      throw ErrorHandler(422, {
        message: localization.validator.noVerifyToken,
      });
    }

    const user = await User.findById(decoded.id);

    user.token = null;
    user.isEmailVerified = true;
    await user.save();

    return {
      token: await jwt.sign(
        {
          id: user._id,
        },
        process.env.SECRET
      ),
    };
  } catch (error) {
    throw error;
  }
};

const login = async (data, local) => {

  const localization = require(`../../client/localization/${local}`);

  try {
    const user = await User.findOne({ email: data.email });

    if (!user) {
      throw new ErrorHandler(422, {
        message: localization.validator.loginData,
      });
    }
```

```
    if (!user.isEmailVerified) {
      throw new ErrorHandler(422, {
        message: localization.validator.noVerifyEmail,
      });
    }

    const isMatch = await bcrypt.compare(data.password, user.password);

    if (!isMatch) {

      throw new ErrorHandler(422, {
        message: localization.validator.loginData,
      });
    }

    return {
      token: await jwt.sign(
        {
          id: user._id,
        },
        process.env.SECRET
      ),
    };
  } catch (error) {
    throw error;
  }
};

const forgot = async (data, local) => {

  const localization = require(`../../client/localization/${local}`);

  try {
    const user = await User.findOne({ email: data.email });

    if (!user) {
      throw new ErrorHandler(422, {
        message: localization.validator.userNoFound,
      });
    }

    if (!user.isEmailVerified) {
      throw new ErrorHandler(422, {
        message: localization.validator.noVerifyEmail,
      });
    }

    const token = await jwt.sign({ id: user._id }, process.env.SECRET);

    await user.update({
      token,
    });

    mail(
      data.email,
      localization.mail.title,
      localization.mail.forgotHTML(`${process.env.CLIENT_URL}/reset/${token}`)
    );

    return;
  } catch (error) {
    throw error;
  }
};
```

```
const reset = async (token, data, local) => {
  const localization = require(`../../client/localization/${local}`);

  try {

    const decoded = jwt.verify(token, process.env.SECRET);

    if (!decoded.id) {
      throw ErrorHandler(422, {
        message: localization.validator.noVerifyToken,
      });
    }

    await User.findByIdAndUpdate(decoded.id, {
      password: await bcrypt.hash(data.password, 12),
      token: "",
    });

    return;
  } catch (error) {
    throw error;
  }
};

const me = async (id) => {
  try {
    return await User.findById(id).select({ password: 0 });
  } catch (error) {
    throw error;
  }
};

module.exports = {
  register,
  verify,
  login,
  forgot,
  reset,
  me,
};
```

А.25 Файл /src/services/curriculum.service.js

```
const {
  University,
  Faculty,
  Specialty,
  Curriculum,
} = require("../../database");
const { ErrorHandler } = require("../helpers/errorHandler");

const getCurriculumOfUFS = async (universityId, facultyId, specialtyId) => {
  try {
    const university = await University.findOne({
      _id: universityId,
      faculties: facultyId,
    });
```

```
    if (!university) {
      throw new ErrorHandler(404);
    }

    const faculty = await Faculty.findOne({
      _id: facultyId,
      specialties: specialtyId,
    });

    if (!faculty) {
      throw new ErrorHandler(404);
    }

    const curriculumId = (await Specialty.findById(specialtyId)).curriculums;

    const curriculum = await Curriculum.find({
      _id: {
        $in: curriculumId,
      },
    }).sort("semester");

    return curriculum;
  } catch (error) {
    throw error;
  }
};

module.exports = {
  getCurriculumOfUFS,
};
```

А.26 Файл /src/services/faculty.service.js

```
const { University, Faculty } = require("../../database");
const { ErrorHandler } = require("../helpers/errorHandler");

const getUniversitysFaculty = async (
  universityId,
  search = "",
  page = 1,
  perPage = 10,
  local
) => {
  try {
    const university = await University.findById(universityId);

    if (!university) {
      throw new ErrorHandler(404);
    }

    const facultyId = (await university.populate("faculties")).faculties;

    const faculty = await Faculty.find(
      {
        _id: {
          $in: facultyId,
        },
        [`name.${local}`]: {
          $regex: search,
```

```
          $options: "i",
        },
      },
      null,
      {
        skip: (page - 1) * perPage,
        limit: perPage,
      }
    ).sort(`name.${local}`);

    return faculty.map(item => ({
      ...item._doc,
      id: item._id,
      name: item.name[local],
    }));
  } catch (error) {
    throw error;
  }
};

module.exports = {
  getUniversitysFaculty,
};
```

А.27 Файл /src/services/filterFields.service.js

```
const {
  University,
  Faculty,
  Specialty,
  ExchProgram,
} = require("../../database");

const getUniversityOfMobility = async (local = "en") => {
  try {
    const location = new Set(
      (
        await University.find({
          hasMobility: true,
        })
          .select({ location: 1, _id: 0 })
          .sort(`location.${local}`)
      ).map((el) => el.location[local])
    );

    let facultyId = (
      await University.find({
        hasMobility: true,
      }).select({ faculties: 1, _id: 0 })
    ).map((el) => el.faculties);

    let temp = [];
    for (let i = 0; i < facultyId.length; i++) {
      temp.push(...facultyId[i]);
    }

    facultyId = new Set(temp);

    const faculty = new Set(
      (
```

```
    await Faculty.find({
      _id: {
        $in: Array.from(facultyId),
      },
    })
      .select({ name: 1, _id: 0 })
      .sort(`name.${local}`)
  ).map((el) => el.name[local])
);

let specialtyId = (
  await Faculty.find({
    _id: {
      $in: Array.from(facultyId),
    },
  }).select({ specialties: 1, _id: 0 })
).map((el) => el.specialties);

temp = [];
for (let i = 0; i < specialtyId.length; i++) {
  temp.push(...specialtyId[i]);
}

specialtyId = new Set(temp);

const specialty = new Set(
  (
    await Specialty.find({
      _id: {
        $in: Array.from(specialtyId),
      },
    })
      .select({ name: 1, _id: 0 })
      .sort(`name.${local}`)
  ).map((el) => el.name[local])
);

let exchProgrammesId = (
  await Specialty.find({
    _id: {
      $in: Array.from(specialtyId),
    },
  }).select({ exchProgrammes: 1, _id: 0 })
).map((el) => el.exchProgrammes);

temp = [];
for (let i = 0; i < exchProgrammesId.length; i++) {
  temp.push(...exchProgrammesId[i]);
}

exchProgrammesId = new Set(temp);

let language = (
  await ExchProgram.find({
    _id: {
      $in: Array.from(exchProgrammesId),
    },
  }).select({ language: 1, _id: 0 })
).map((el) => el.language[local]);

temp = [];
for (let i = 0; i < language.length; i++) {
  temp.push(...language[i]);
}
```

```
    language = new Set(temp);

    const type = new Set(
      (
        await ExchProgram.find({
          _id: {
            $in: Array.from(exchProgrammesId),
          },
        })
          .select({ type: 1, _id: 0 })
          .sort(`type.${local}`)
      ).map((el) => el.type[local])
    );

    let season = (
      await ExchProgram.find({
        _id: {
          $in: Array.from(exchProgrammesId),
        },
      })
        .select({ season: 1, _id: 0 })
        .sort(`season.${local}`)
    ).map((el) => el.season[local]);

    temp = [];
    for (let i = 0; i < season.length; i++) {
      temp.push(...season[i]);
    }

    season = new Set(temp);

    const period = new Set(
      (
        await ExchProgram.find({
          _id: {
            $in: Array.from(exchProgrammesId),
          },
        })
          .select({ period: 1, _id: 0 })
          .sort(`period.${local}`)
      )
        .map((el) => el.period[local])
    );

    return {
      location: Array.from(location),
      faculty: Array.from(faculty),
      specialty: Array.from(specialty),
      language: Array.from(language).sort(),
      type: Array.from(type),
      season: Array.from(season).sort(),
      period: Array.from(period),
    };
  } catch (error) {
    console.log(error);
    throw error;
  }
};

module.exports = {
  getUniversityOfMobility,
};
```

А.28 Файл /src/services/specialty.service.js

```
const { University, Faculty, Specialty } = require("../../database");
const { ErrorHandler } = require("../helpers/errorHandler");

const getSpecialtyOfFacultyAndUniversity = async (
  universityId,
  facultyId,
  search = "",
  page = 1,
  perPage = 10,
  local
) => {
  try {
    const university = await University.findOne({
      _id: universityId,
      faculties: facultyId,
    });

    if (!university) {
      throw new ErrorHandler(404);
    }

    const specialtyId = (await Faculty.findById(facultyId)).specialties;

    const specialty = await Specialty.find(
      {
        _id: {
          $in: specialtyId,
        },
        [`name.${local}`]: {
          $regex: search,
          $options: "i",
        },
      },
      null,
      {
        skip: (page - 1) * perPage,
        limit: perPage,
      }
    ).sort(`name.${local}`);

    return specialty.map((item) => ({
      ...item._doc,
      id: item._id,
      name: item.name[local],
    }));
  } catch (error) {
    throw error;
  }
};

module.exports = {
  getSpecialtyOfFacultyAndUniversity,
};
```

А.29 Файл /src/services/university.service.js

```
const { University, UniversitiesView } = require("../../database");
const { ErrorHandler } = require("../helpers/errorHandler");
const { universityChangeDataByLocal } = require("../helpers");

const get = async (search = "", page = 1, perPage = 10, local) => {
  try {
    const university = await University.find(
      {
        [`name.${local}`]: {
          $regex: search,
          $options: "i",
        },
      },
      null,
      {
        skip: (page - 1) * perPage,
        limit: perPage,
      }
    ).sort(`name.${local}`);

    return university.map((item) => ({
      ...item._doc,
      id: item._id,
      name: item.name[local],
      description: item.description[local],
      location: item.location[local],
    }));
  } catch (error) {
    throw error;
  }
};

const getById = async (id, local) => {
  try {
    const university = await UniversitiesView.findById(id);

    if (!university) {
      throw new ErrorHandler(404, {});
    }

    return universityChangeDataByLocal(university, local);
  } catch (error) {
    throw error;
  }
};

const getOfMobility = async (
  search = "",
  page = 1,
  perPage = 10,
  {
    location = "",
    faculty = "",
    specialty = "",
    language = "",
    type = "",
    season = "",
    period = "",
```

```
    },
    local
  ) => {
    try {
      const query = {
        hasMobility: true,
        [`name.${local}`]: {
          $regex: search,
          $options: "i",
        },
        [`location.${local}`]: {
          $regex: location,
          $options: "i",
        },
        faculties: {
          $elemMatch: {
            [`name.${local}`]: {
              $regex: faculty,
              $options: "i",
            },
            specialties: {
              $elemMatch: {
                [`name.${local}`]: {
                  $regex: specialty,
                  $options: "i",
                },
                exchProgrammes: {
                  $elemMatch: {
                    [`language.${local}`]: {
                      $regex: language,
                      $options: "i",
                    },
                    [`type.${local}`]: {
                      $regex: type,
                      $options: "i",
                    },
                    [`season.${local}`]: {
                      $regex: season,
                      $options: "i",
                    },
                    [`period.${local}`]: {
                      $regex: period,
                      $options: "i",
                    },
                  },
                },
              },
            },
          },
        },
      };

      const universities = await UniversitiesView.find(query, null, {
        skip: (page - 1) * perPage,
        limit: perPage,
      }).sort(`name.${local}`);

      const count = await UniversitiesView.countDocuments(query);

      const res = universities.map((university) =>
        universityChangeDataByLocal(university, local)
      );

      return {
```

```
      items: res,
      count: count,
    };
  } catch (error) {
    throw error;
  }
};

module.exports = {
  get,
  getById,
  getOfMobility,
};
```

А.30 Файл /test/auth/forgot.test.js

```
const request = require("supertest");
const app = require("../../server");
const db = require("../../database");

const testData = {
  fullName: "First name Last name",
  email: "test-forgot@exapmle.com",
  birthday: "1999-01-01",
  universityId: null,
  facultyId: null,
  specialtyId: null,
  semester: 7,
  password: "123456",
  confirmPassword: "123456",
};

describe("Forgot endpoint", () => {
  let token;

  beforeAll(async () => {
    const university = await db.University.findOne({});
    const faculty = await db.Faculty.findOne({});
    const specialty = await db.Specialty.findOne({});
    testData.universityId = university._id;
    testData.facultyId = faculty._id;
    testData.specialtyId = specialty._id;

    await request(app).post("/api/auth/register").send(testData);

    const user = await db.User.findOne({
      email: testData.email,
    });

    token = user.token;
  });

  it("should return a validation error", async () => {
    const res = await request(app).post("/api/auth/forgot");

    expect(res.statusCode).toEqual(422);
    expect(res.body.email).toBe("Field must be required");
  });

  it("should return an error that the user does not exist", async () => {
    const res = await request(app).post("/api/auth/forgot").send({
      email: "test@example.com",
```

```
    });

    expect(res.statusCode).toEqual(422);
    expect(res.body.message).toBe("This user does not exist");
  });

  it("should return an error that the mail was not confirmed", async () => {
    const res = await request(app).post("/api/auth/forgot").send({
      email: testData.email,
    });

    expect(res.statusCode).toEqual(422);
    expect(res.body.message).toBe("Mail has not been confirmed");

    await request(app).put(`/api/auth/verify/${token}`);
  });

  it("should create a password recovery token", async () => {
    const res = await request(app).post("/api/auth/forgot").send({
      email: testData.email,
    });

    expect(res.statusCode).toEqual(200);
    expect(res.body.message).toBe(
      "A message was sent to the mail with password recovery"
    );
  });

  afterAll(async () => {
    await db.User.findOneAndRemove({
      email: testData.email
    });

    await app.close();
  });
});
```

А.31 Файл /test/auth/login.test.js

```
const request = require("supertest");
const app = require("../../server");
const db = require("../../database");

const testData = {
  fullName: "First name Last name",
  email: "test-login@exapmle.com",
  birthday: "1999-01-01",
  universityId: null,
  facultyId: null,
  specialtyId: null,
  semester: 7,
  password: "123456",
  confirmPassword: "123456",
};

describe("Login endpoint", () => {
  let token;

  beforeAll(async () => {
    const university = await db.University.findOne({});
    const faculty = await db.Faculty.findOne({});
    const specialty = await db.Specialty.findOne({});
```

```
    testData.universityId = university._id;
    testData.facultyId = faculty._id;
    testData.specialtyId = specialty._id;

    await request(app).post("/api/auth/register").send(testData);

    const user = await db.User.findOne({
      email: testData.email,
    });

    token = user.token;
  });

  it("should return a validation error", async () => {
    const res = await request(app).post("/api/auth/login");

    expect(res.statusCode).toEqual(422);
    expect(res.body.email).toBe("Field must be required");
    expect(res.body.password).toBe("Field must be required");
  });

  it("should return an error indicating that the login information was incorrect",
async () => {
    const res = await request(app).post("/api/auth/login").send({
      email: "test@example.com",
      password: "123456",
    });

    expect(res.statusCode).toEqual(422);
    expect(res.body.message).toBe("Incorrect login information entered");
  });

  it("should return an error that the mail was not confirmed", async () => {

    const res = await request(app).post("/api/auth/login").send({
      email: testData.email,
      password: testData.password,
    });

    expect(res.statusCode).toEqual(422);
    expect(res.body.message).toBe("Mail has not been confirmed");

    await request(app).put(`/api/auth/verify/${token}`);
  });

  it("must authorize the user", async () => {

    const res = await request(app).post("/api/auth/login").send({
      email: testData.email,
      password: testData.password,
    });

    expect(res.statusCode).toEqual(200);
    expect(res.body.token).not.toBeUndefined();
    expect(res.body.user.id).not.toBeUndefined();
    expect(res.body.user.fullName).not.toBeUndefined();
    expect(res.body.user.email).toEqual("test-login@exapmle.com");
    expect(res.body.user.birthday).not.toBeUndefined();
    expect(res.body.user.university).not.toBeUndefined();
    expect(res.body.user.applications).not.toBeUndefined();
  });

  afterAll(async () => {
```

```
    await db.User.findOneAndRemove({
      email: testData.email,
    });

    await app.close();
  });
});
```

А.32 Файл /test/auth/register.test.js

```
const request = require("supertest");
const app = require("../../server");
const db = require("../../database");

const testData = {

  fullName: "First name Last name",
  email: "test-register@exapmle.com",
  birthday: "1999-01-01",
  universityId: null,
  facultyId: null,
  specialtyId: null,
  semester: 7,
  password: "123456",
  confirmPassword: "123456",
};

describe("Register endpoint", () => {

  beforeAll(async () => {

    const university = await db.University.findOne({});
    const faculty = await db.Faculty.findOne({});
    const specialty = await db.Specialty.findOne({});
    testData.universityId = university._id;
    testData.facultyId = faculty._id;
    testData.specialtyId = specialty._id;
  });

  it("should create a new user", async () => {

    const res = await request(app).post("/api/auth/register").send(testData);

    expect(res.statusCode).toEqual(201);
    expect(res.body.message).toBe("User has been registered!");
  });

    it("should return an error that email is not free", async () => {
      const res = await request(app).post("/api/auth/register").send(testData);

      expect(res.statusCode).toEqual(422);
      expect(res.body.email).toBe("This email is not free");
    });

    it("should return an error that the validation failed", async () => {
      const res = await request(app).post("/api/auth/register");

      expect(res.statusCode).toEqual(422);
      expect(res.body.fullName).toBe("Field must be required");
      expect(res.body.email).toBe("Field must be required");
      expect(res.body.birthday).toBe("Field must be required");
```

```
      expect(res.body.universityId).toBe("Field must be required");
      expect(res.body.facultyId).toBe("Field must be required");
      expect(res.body.specialtyId).toBe("Field must be required");
      expect(res.body.semester).toBe("Field must be required");
      expect(res.body.password).toBe("Field must be required");
      expect(res.body.confirmPassword).toBe("Field must be required");
    });

    it("should return an error that the selected university does not exist", async
() => {
      const res = await request(app).post("/api/auth/register").send({
        fullName: "First name Last name",
        email: "test-register-1@exapmle.com",
        birthday: "1999-01-01",
        universityId: db.mongoose.Types.ObjectId(),
        facultyId: db.mongoose.Types.ObjectId(),
        specialtyId: db.mongoose.Types.ObjectId(),
        semester: 8,
        password: "123456",
        confirmPassword: "123456",
      });

      expect(res.statusCode).toEqual(422);
      expect(res.body.universityId).toBe("Selected university does not exist");
    });

    it("should return the error that the selected university does not have the
selected faculty", async () => {

      const res = await request(app).post("/api/auth/register").send({

        fullName: "First name Last name",
        email: "test-register-1@exapmle.com",
        birthday: "1999-01-01",
        universityId: testData.universityId,
        facultyId: db.mongoose.Types.ObjectId(),
        specialtyId: db.mongoose.Types.ObjectId(),
        semester: 8,
        password: "123456",
        confirmPassword: "123456",
      });

      expect(res.statusCode).toEqual(422);
      expect(res.body.facultyId).toBe(
        "Selected university does not have a selected faculty"
      );
    });

    it("should return the error that the selected faculty does not have the
selected specialty", async () => {

      const res = await request(app).post("/api/auth/register").send({
        fullName: "First name Last name",
        email: "test-register-1@exapmle.com",
        birthday: "1999-01-01",
        universityId: testData.universityId,
        facultyId: testData.facultyId,
        specialtyId: db.mongoose.Types.ObjectId(),
        semester: 8,
        password: "123456",
        confirmPassword: "123456",
      });

      expect(res.statusCode).toEqual(422);
```

```
      expect(res.body.specialtyId).toBe(
        "Selected faculty does not have the a selected specialty"
      );
    });

    it("should return an error stating that the selected specialty does not have
this semester", async () => {

      const res = await request(app).post("/api/auth/register").send({
        fullName: "First name Last name",
        email: "test-register-1@exapmle.com",
        birthday: "1999-01-01",
        universityId: testData.universityId,
        facultyId: testData.facultyId,
        specialtyId: testData.specialtyId,
        semester: -1,
        password: "123456",
        confirmPassword: "123456",
      });

      expect(res.statusCode).toEqual(422);
      expect(res.body.semester).toBe(
        "Selected specialty does not have a selected semester"
      );
    });

    it("should return an error that the passwords do not match", async () => {
      const res = await request(app).post("/api/auth/register").send({
        fullName: "First name Last name",
        email: "test-register-1@exapmle.com",
        birthday: "1999-01-01",
        universityId: testData.universityId,
        facultyId: testData.facultyId,
        specialtyId: testData.specialtyId,
        semester: 8,
        password: "123456",
        confirmPassword: "1234567",
      });

      expect(res.statusCode).toEqual(422);
      expect(res.body.confirmPassword).toBe("Password does not match");
    });

  afterAll(async () => {
    await db.User.findOneAndRemove({
      email: testData.email
    });

    await app.close();
  });
});
```

А.33 Файл /test/auth/reset.test.js

```
const request = require("supertest");
const app = require("../../server");
const db = require("../../database");

const testData = {

  fullName: "First name Last name",
  email: "test-reset@exapmle.com",
```

```
  birthday: "1999-01-01",
  universityId: null,
  facultyId: null,
  specialtyId: null,
  semester: 7,
  password: "123456",
  confirmPassword: "123456",
};

describe("Reset endpoint", () => {

  let token;

  beforeAll(async () => {

    const university = await db.University.findOne({});
    const faculty = await db.Faculty.findOne({});
    const specialty = await db.Specialty.findOne({});
    testData.universityId = university._id;
    testData.facultyId = faculty._id;
    testData.specialtyId = specialty._id;

    await request(app).post("/api/auth/register").send(testData);

    let user = await db.User.findOne({
      email: testData.email,
    });

    await request(app).get(`/api/auth/verify/${user.token}`);

    await request(app).post("/api/auth/forgot").send({
      email: testData.email,
    });

    user = await db.User.findOne({
      email: testData.email,
    });

    token = user.token;
  });

  it("should return a validation error", async () => {

    const res = await request(app).post(`/api/auth/reset/${token}`);

    expect(res.statusCode).toEqual(422);
    expect(res.body.password).toBe("Field must be required");
    expect(res.body.confirmPassword).toBe("Field must be required");
  });

  it("should recover the password", async () => {

    const res = await request(app).post(`/api/auth/reset/${token}`).send({
      password: "123456",
      confirmPassword: "123456",
    });

    expect(res.statusCode).toEqual(200);
    expect(res.body.message).toBe("Password changed successfully");
  });

  afterAll(async () => {

    await db.User.findOneAndRemove({
```

```
      email: testData.email
    });

    await app.close();
  });
});
```

ДОДАТОК Б

Слайди презентації

Рисунок Б.1 – Слайд 1

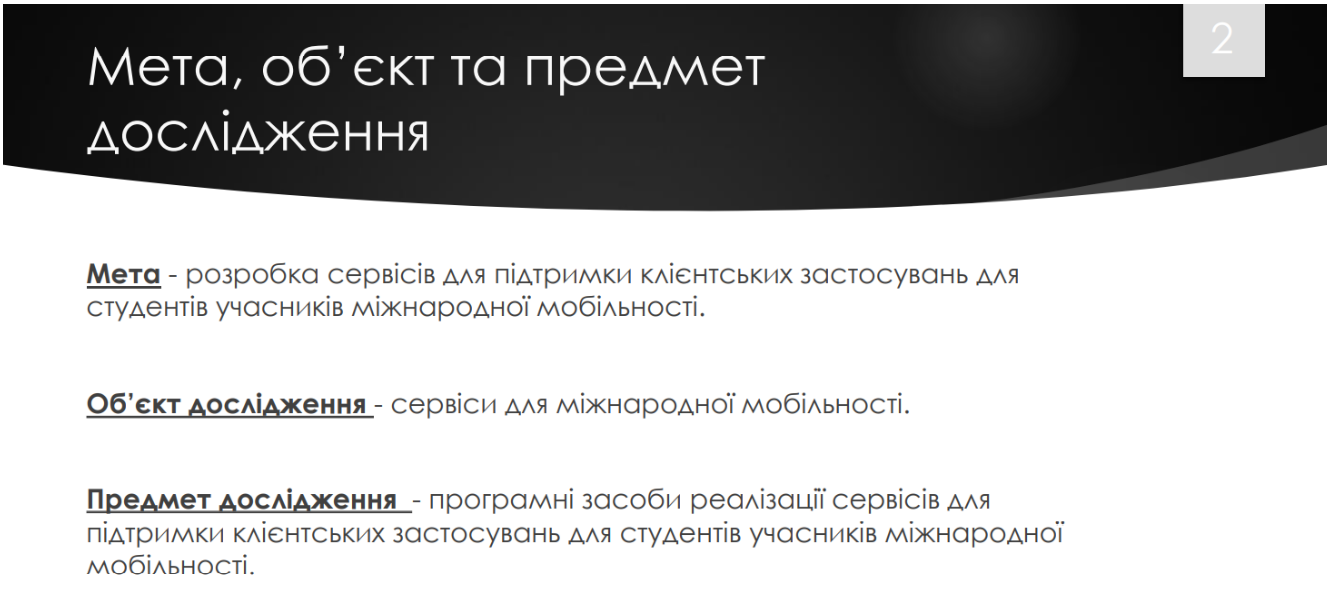

Рисунок Б.2 – Слайд 2

Рисунок Б.3 – Слайд 3

Рисунок Б.4 – Слайд 4

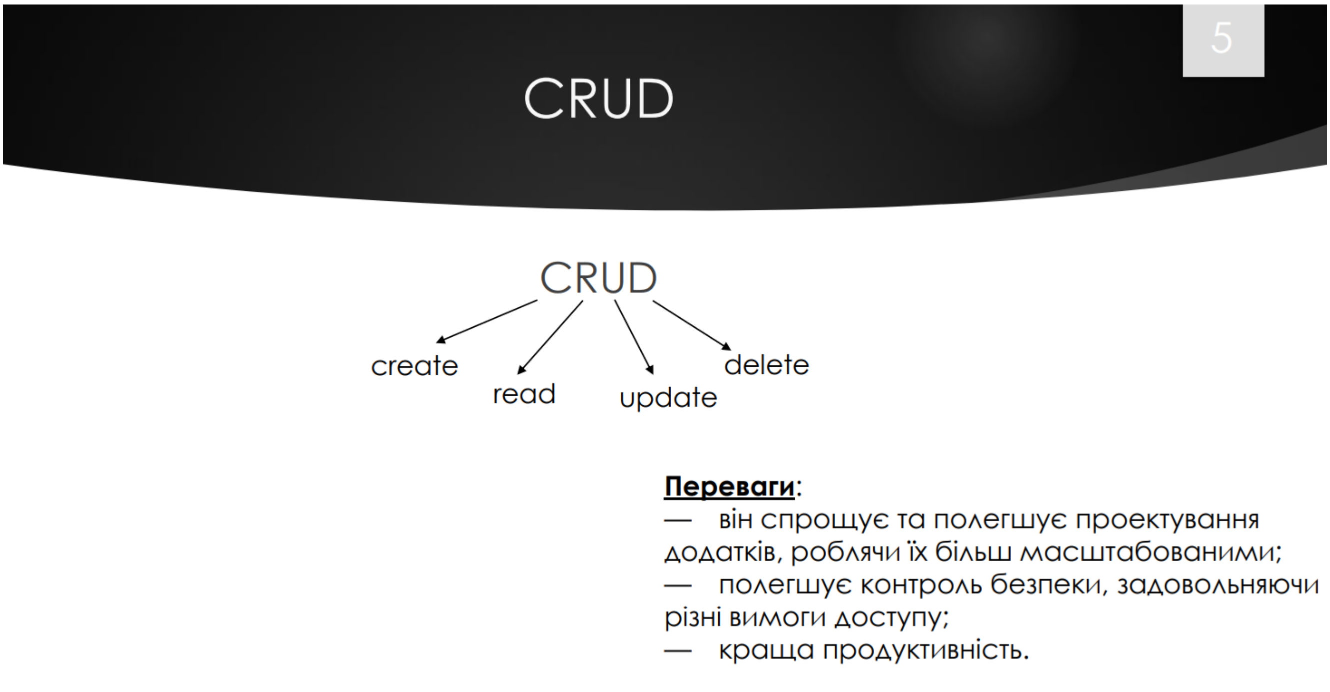

Рисунок Б.5 – Слайд 5

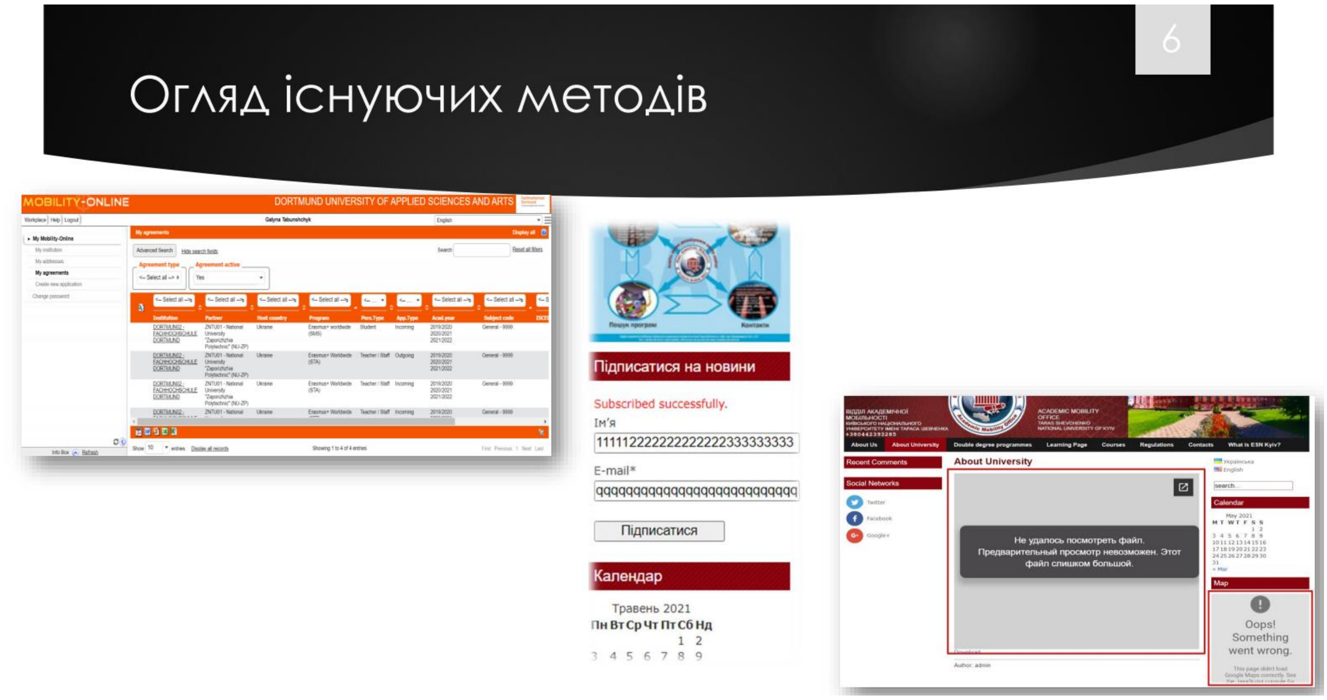

Рисунок Б.6 – Слайд 6

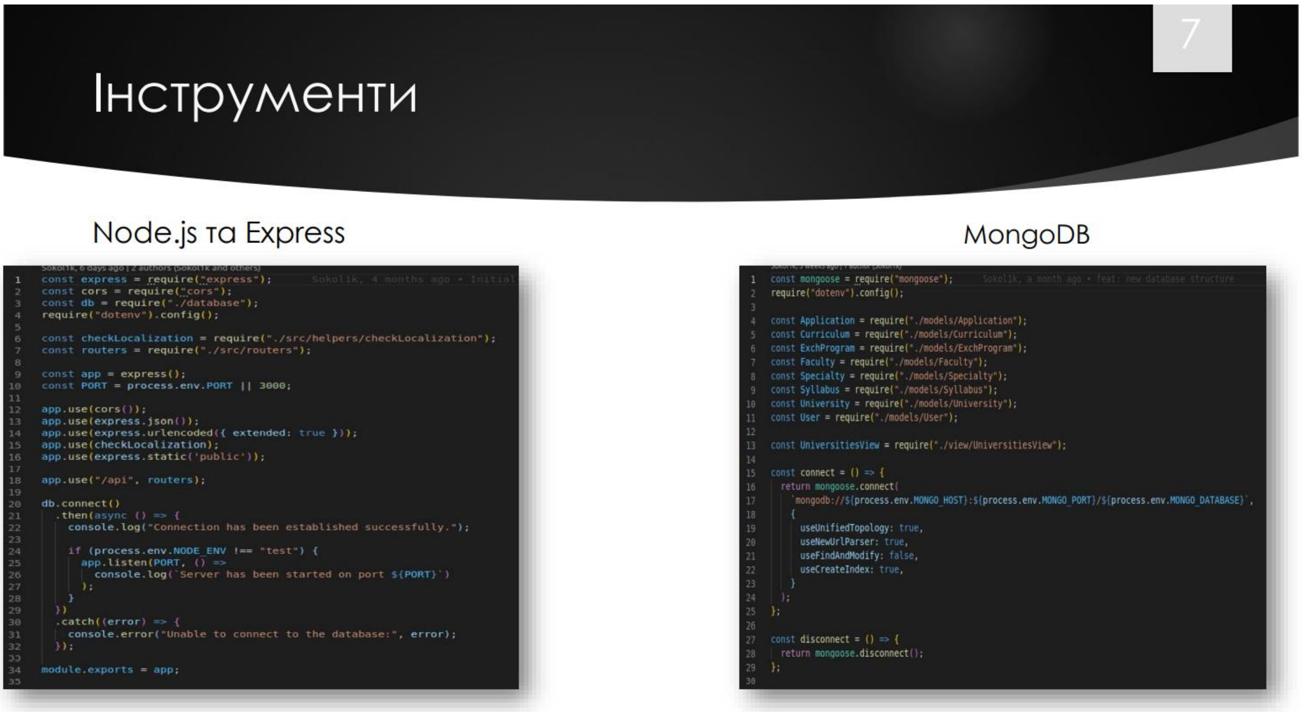

Рисунок Б.7 – Слайд 7

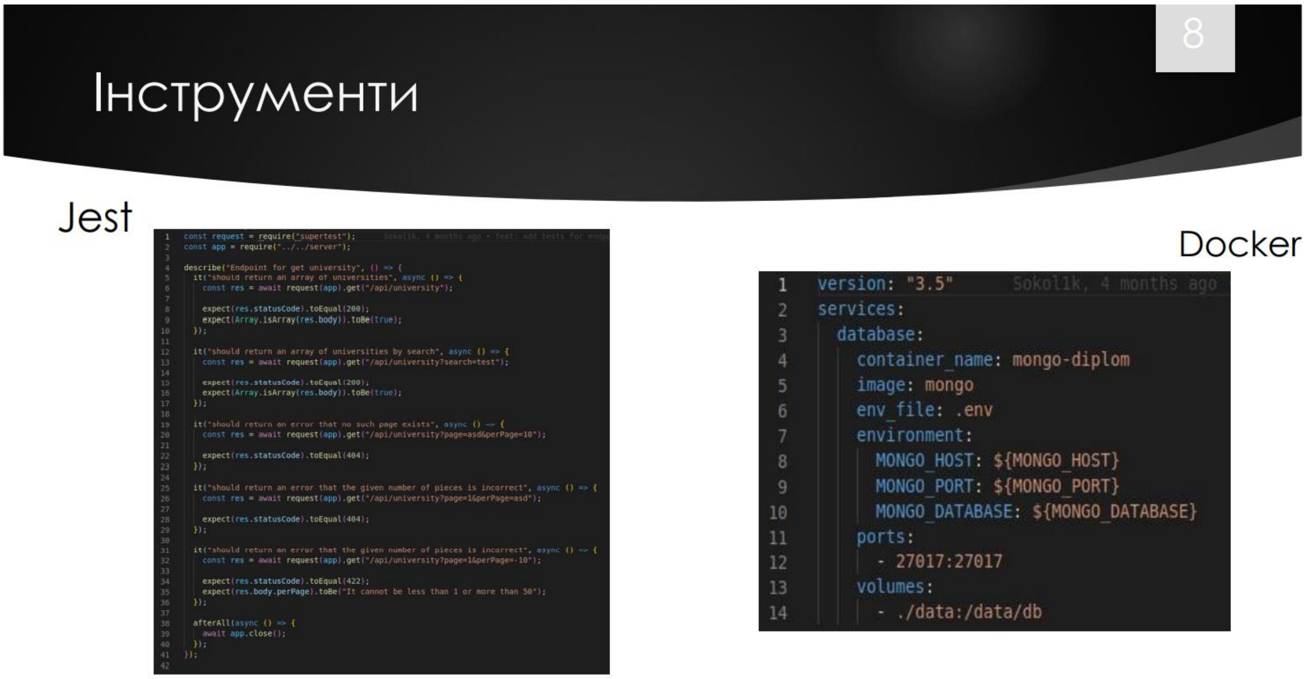

Рисунок Б.8 – Слайд 8

Рисунок Б.9 – Слайд 9

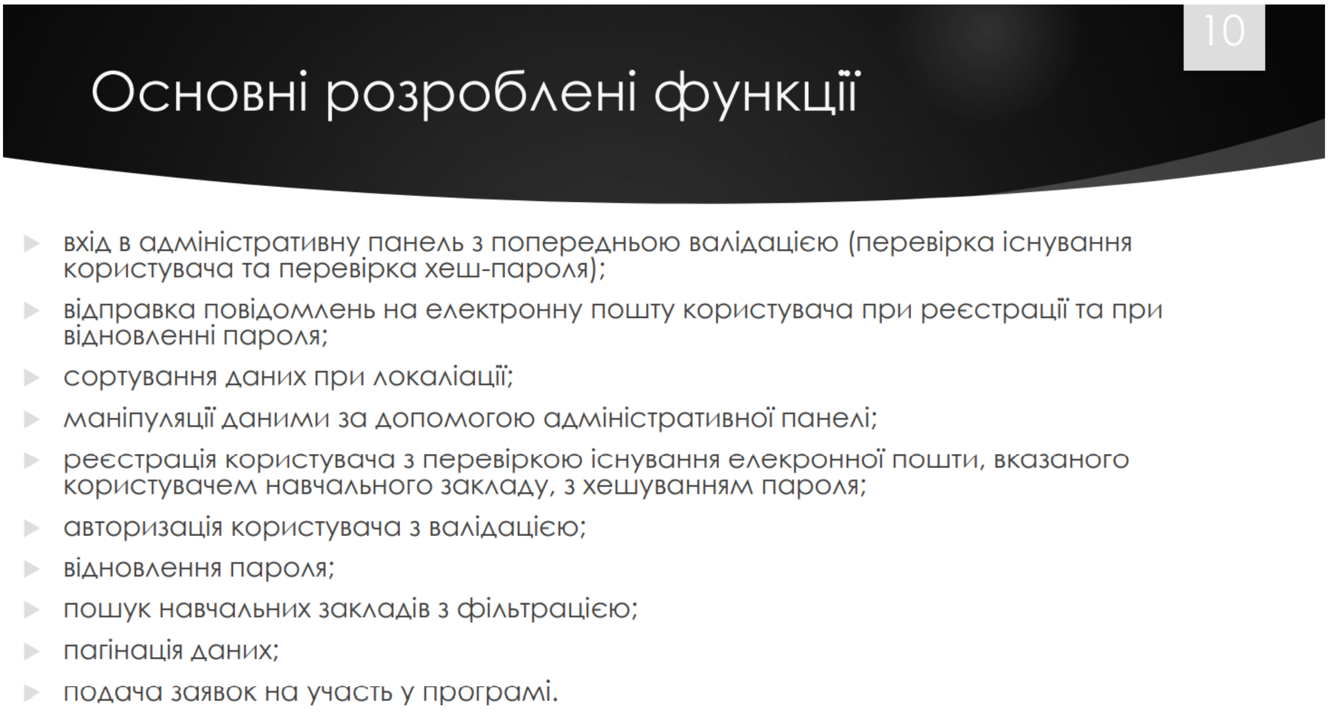

Рисунок Б.10 – Слайд 10

Таблиці БД

11

Users

1	Назва поля	Тип поля	Опис
2	ID	Integer	Ключове поле таблиці
3	fullName	Text	Ім'я та прізвище
4	email	Text	Пошта
5	password	Text	Пароль
6	birthday	Date	Дата народження
7	token	Text	
8	isEmailVerified	Boolean	Перевірка верифікації пошти
9	university	Json	Університет
10	applications	Array	Заявки

ExchProgrammes

1	Назва поля	Тип поля	Опис поля
2	ID	Integer	Ключове поле таблиці
3	name	Text	Назва програми
4	language	Text	Мова навчання
5	type	Text	Віртуальне/фактичне навчання
6	season	Text	Весна/осінь/осінь-весна
7	period	Text	Семестр/декілька тижнів
8	ectsCredits	Integer	Кількість кредитів
9	coordinatorFullName	Text	Ім'я координатора
10	coordinatorEmail	Text	Пошта координатора
11	syllabuses	Array	Предмети

Рисунок Б.11 – Слайд 11

Діаграма БД

12

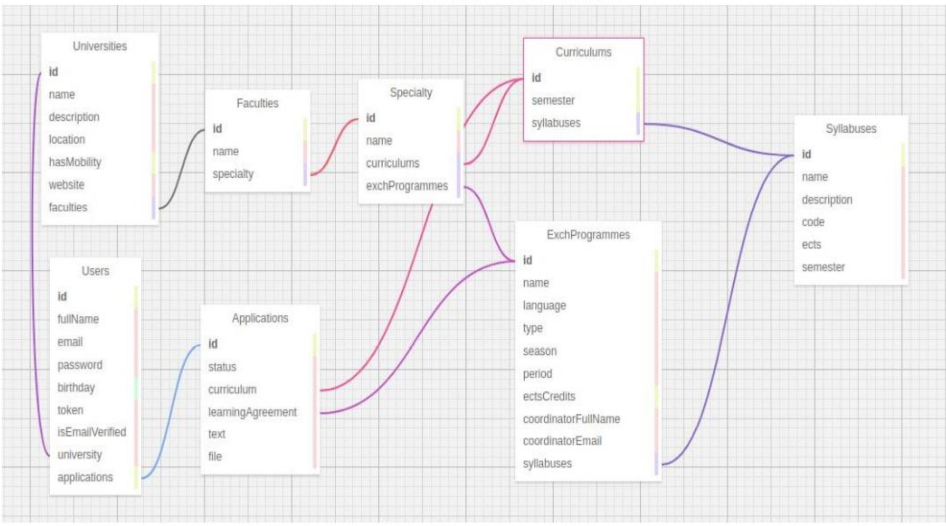

Рисунок Б.12 – Слайд 12

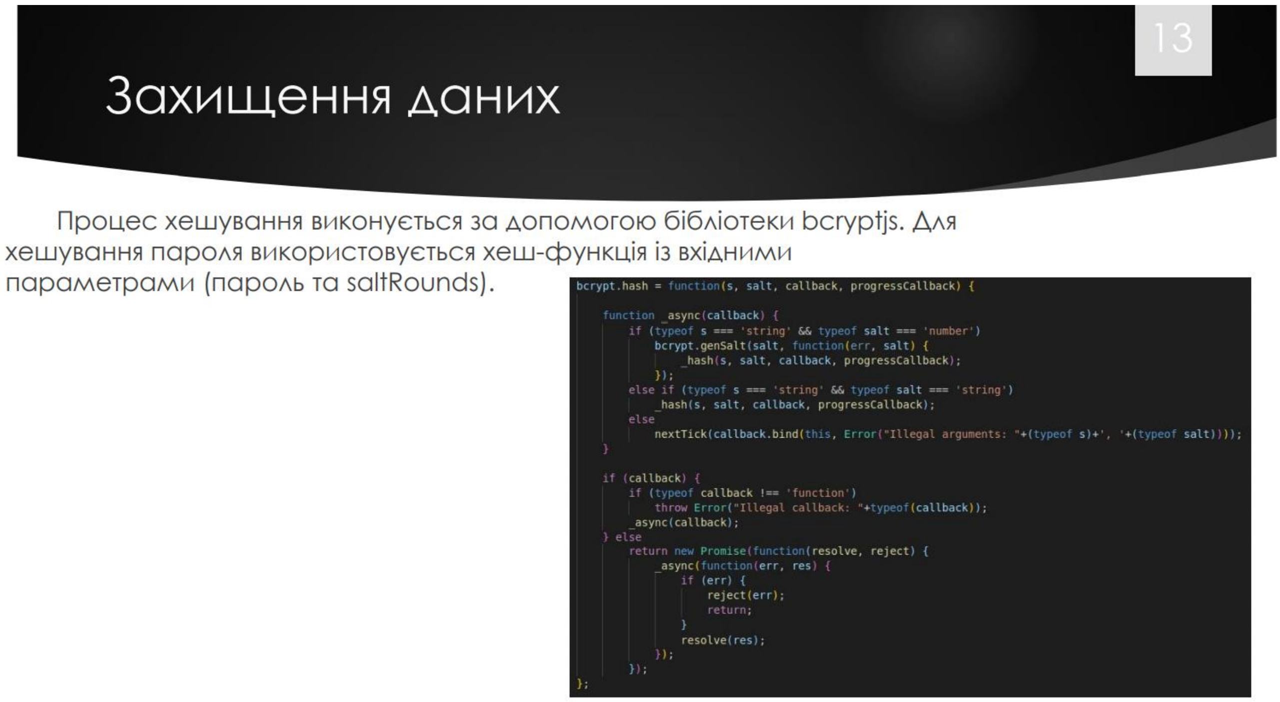

Рисунок Б.13 – Слайд 13

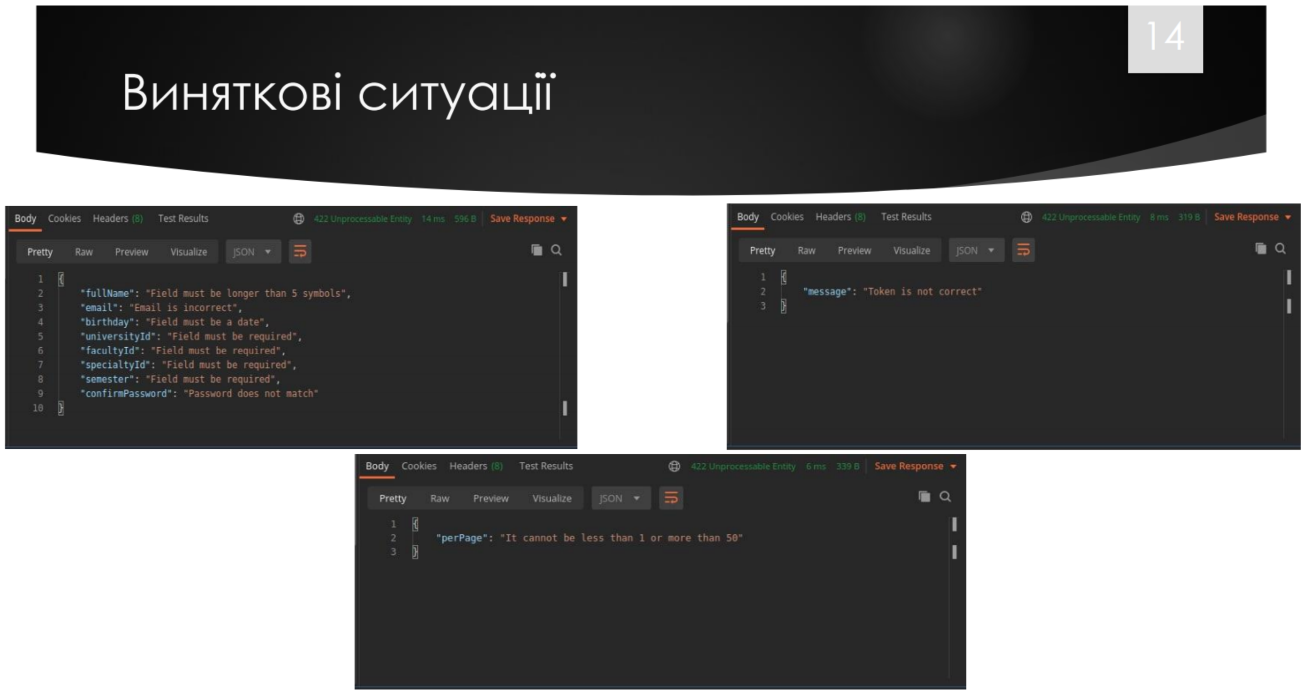

Рисунок Б.14 – Слайд 14

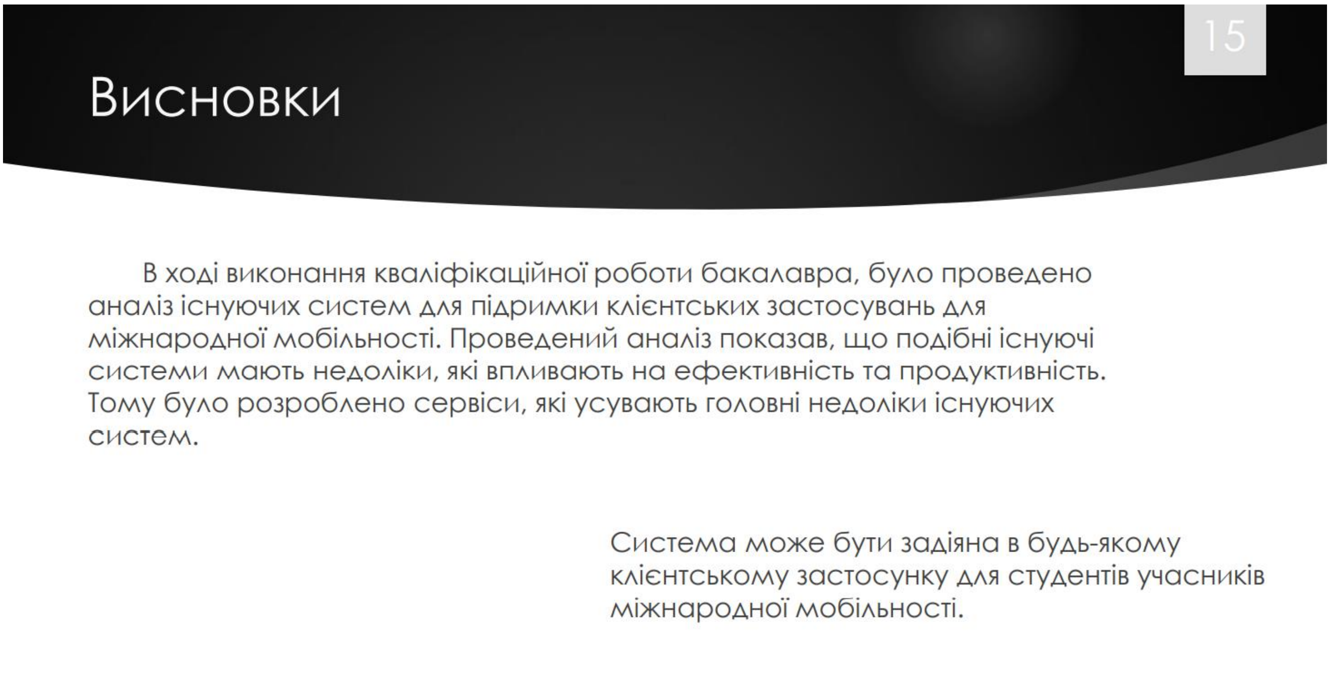

Рисунок Б.15 – Слайд 15

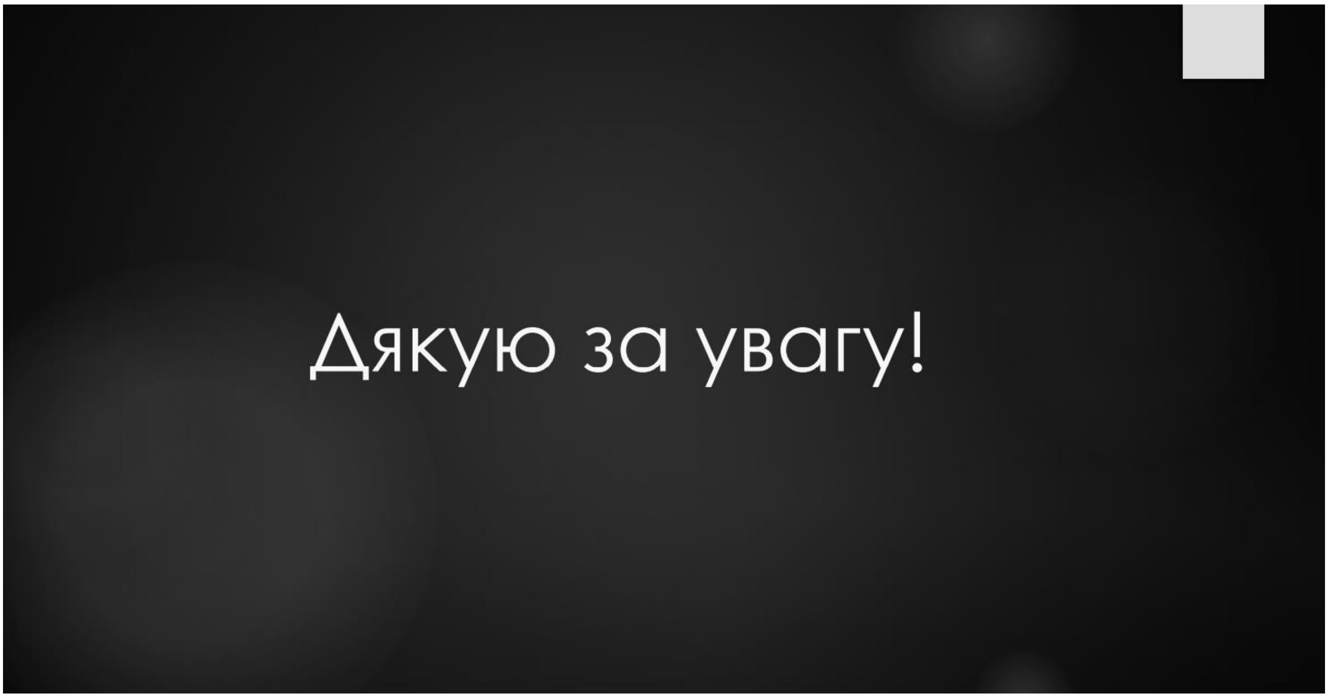

Рисунок Б.16 – Слайд 16