‘

Прикарпатський національний університет імені Василя Стефаника

Кафедра інформаційних технологій

КУРСОВА РОБОТА

з дисципліни Об’єктно орієнотоване програмування

на тему: Розробка програми із використанням Java Spring для функціоналу кафедри ВНЗ.

Студента 2 курсу, групи ІПЗ-23
спеціальності 121 – Інженерія
програмного забезпечення
Совтус А. А.
Керівник Полатайко І. Б.

Національна шкала: _____

Університетська шкала: _____

Оцінка ECTS: _____

Члени комісії: _____ Іщеряков С.М.
(підпис) (прізвище та ініціали)

_____ Кузь М. В.
(підпис (прізвище та ініціали)

_____ Дутчак М. С.
(підпис (прізвище та ініціали)

м. Івано-Франківськ – 2023 рік

‘

Прикарпатський національний університет імені Василя Стефаника

ЗАВДАННЯ НА КУРСОВИЙ ПРОЄКТ

Совтус Андрій Андрійович

(прізвище, ім’я, по батькові студента)

Кафедра інформаційних технологій Дисципліна «ООП»

Спеціальність 121 Інженерія програмного забезпечення

Курс 2 Група ІПЗ-23 Семестр 2

1. Тема роботи (проєкту) Розробка програми із використанням Java Spring для функціоналу кафедри ВНЗ.

2. Рекомендована література Bruce Eckel Thinking in Java, Richard Warburton Java 8 Lambdas, Mala Gupta OCA Java SE 7, Руденко В. Д. Жугастров А. А. Вивчаємо Java у школі.

3. Перелік питань, які підлягають розробці проектування класів та взаємодія об’єктів, успадкування класів, аналіз предметної області та створення програми для функціоналу веб-сайту, створення точок для запитів за допомогою REST архітектури.

4. Дата видачі завдання: 21.02.2023

Термін подачі до захисту: 09.062023

5. Студент: Совтус А.А. Керівник: Полатайко І.Б.

‘

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів підготовки курсової роботи (проекту)	Термін виконання	Форма контролю
1.	Затвердження теми роботи		Затвердження на засіданні кафедри
2.	Узгодження завдання та плану курсової роботи з науковим керівником		Сформовані завдання та план курсової роботи
3.	Обґрунтування актуальності, формулювання мети, завдання, предмету та об’єкту дослідження роботи		Узгодження з науковим керівником
4.	Опрацювання джерел з теми роботи		Підготовка огляду літератури
5.	Підготовка І розділу роботи		Звіт на наук. семінарі кафедри
6.	Підготовка ІІ розділу роботи		Звіт на наук. семінарі кафедри
7.	Підготовка ІІІ розділу роботи		Представлення наук. керівнику
8.	Виправлення зауважень. Підготовка вступу і висновків.		Представлення наук. керівнику
10.	Оформлення роботи згідно вимог.		Представлення наук. керівнику
12.	Виправлення зауважень керівника. Відправка електронних версій нормоконтролеру.		Представлення наук. керівнику та нормоконтролеру.
13.	Виправлення зауважень нормоконролера. Подача видрукуваної і підписаної роботи (завдання, календарний план, зміст) на кафедру.		Наявність роботи із усіма необхідної підписами на кафедрі
14.	Оцінювання роботи		Виставлення оцінки на титульній сторінці роботи і заповнення відомості

Студент _____ Совтус А.А.
(підпис) (ПІБ)

Керівник роботи _____ Полатайко І.Б.
(підпис) (ПІБ)

‘

РЕФЕРАТ

Пояснювальна записка: сторінок (у рефераті), рисунків, джерел, додатки на сторінках.

Ключові слова: ООП, Основні поняття ООП, REST API, Java, Java Spring, IntelliJ IDEA, Backend, КАФЕДРА.

Об`єктом дослідження є програма для функціоналу кафедри ВНЗ.

Мета роботи – спроектувати та розробити необхідні компоненти бекенду для функціоналу кафедри ВНЗ, використовуючи ООП.

Стислий опис тексту пояснювальної записки:

У даному курсовому проєкті описано основні етапи проектування та розробки бекенду з використанням ООП на фреймворку Java Spring Boot, такі як: визначення переваг та недоліків середовищ розробки Java, проектування REST API ендпоїнтів, використання переваг ООП, приклади роботи готової програми.

ABSTRACT

Explanatory note: pages (in abstract), figures, references, appendix on pages.

Key words: OOP, Basic concepts of OOP, REST API, Java, Java Spring, IntelliJ IDEA, Back-end, Chair.

Object of stud**y** – program for the function chair of the university.

Designing and developing the necessary backend components for the functionality of the university chair, using OOP.

This coursework project describes the main stages of designing and developing a backend using Object-Oriented Programming (OOP) on the Java Spring framework. These stages include: determining the advantages and disadvantages of the Java development environment, designing REST API endpoints, leveraging the benefits of OOP, and providing examples of a functioning program.

`

ЗМІСТ

`

ВСТУП

Об'єктно-орієнтоване програмування (ООП) є одним з найважливіших підходів до розробки програмного забезпечення, який дозволяє створювати комплексні, модульні та легко зрозумілі програми. Застосування ООП надає можливість створювати програми, що взаємодіють з об'єктами та виконують дії, базуючись на концепції об'єктів та класів.

Ця курсова робота присвячена вивченню ООП та його застосування в розробці програмного забезпечення. Метою даної роботи є детальний аналіз та опис принципів, концепцій та основних ідеї ООП, а також дослідження його переваг і можливостей в контексті сучасного програмування.

Тут будуть розглянуті основні принципи ООП, такі як інкапсуляція, успадкування та поліморфізм. Буде проведений огляд мови програмування, що підтримує ООП. Також розглядатимуться практичні приклади реалізації ООП-програм, включаючи створення класів, об'єктів, методів та взаємодію між ними.

Результати дослідження та практичних прикладів, наданих у цій курсовій роботі, стануть важливим внеском у розвиток програмування та сприятимуть подальшій популяризації об'єктно-орієнтованого підходу.

Предметною областю курсової роботи було проектування та розробка програми для контролю кафедри ВНЗ. Кафедра є важливою структурною одиницею в закладі вищої освіти та даний проєкт буде спрямований на менеджмент інфракструктури кафедри.

Метою курсової роботи є створення програми для кафедр ВНЗ за допомогою ООП.

Завдання курсової роботи:

- З’ясувати поняття ООП;
- Засвоїти методику ООП;
- Навчитися розробляти ER-діаграми;
- Навчитися розробляти програму на основі ER-діаграми;

`

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Визначення предметної області проекту

Визначення предметної області проекту є одним із важливих кроків у розробці програмного забезпечення з використанням об'єктно-орієнтованого програмування (ООП). Предметна область визначає контекст, в якому працює програма, і визначає об'єкти, які будуть моделюватися у програмі.

Під час визначення предметної області необхідно провести аналіз реального світу, в якому функціонуватиме програмне забезпечення, і ідентифікувати основні сутності, з якими воно буде працювати. Це можуть бути об'єкти, які відображають реальні або абстрактні елементи, процеси, взаємодії, атрибути тощо.

Після ідентифікації сутностей потрібно встановити їх характеристики, взаємозв'язки та функціональні вимоги до них. Це включає визначення атрибутів об'єктів, їхніх поведінок (методів), взаємодій між об'єктами та інших важливих аспектів. Наприклад, якщо ми розробляємо систему управління бібліотекою, сутностями можуть бути книги, користувачі, бібліотекарі, взаємодія між ними (видача, повернення книги), атрибути книги (назва, автор, жанр) тощо.

Важливим аспектом визначення предметної області є врахування потреб і вимог користувачів. Необхідно аналізувати їхні бізнес-процеси та взаємодії з системою, щоб впевнитися, що програмне забезпечення відповідає їхнім потребам і сприяє покращенню їхньої продуктивності.

У контексті ООП визначення предметної області визначає набір класів і об'єктів, які будуть використовуватися для моделювання предметної області. Класи відображають типи об'єктів, а об'єкти - конкретні екземпляри цих класів.

`

Вони визначають атрибути та методи, які відповідають характеристикам та поведінці об'єктів у реальному світі.

В результаті визначення предметної області проекту ми отримуємо основну структуру програми, яка базується на об'єктах та класах, що відображають реальний світ. Це дозволяє розробникам ефективно працювати з програмним забезпеченням, орієнтуючись на конкретні сутності і взаємодіючи з ними за допомогою методів та інших механізмів ООП.

Отже, визначення предметної області є початковим кроком у розробці програмного забезпечення з використанням ООП, який допомагає створити модель системи, що відображає реальний світ та задовольняє потреби користувачів.

Кафедра – структурний підрозділ факультету, яка несе відповідальність за викладання навчальних дисциплін, організацію виховної роботи, проведення виробничих практик, підготовку та підвищення кваліфікації кадрів. Кафедра здійснює необхідний обсяг навчально-методичної роботи, який забезпечує проведення навчального процесу на високому науково-методичному рівні а також провадить науково-дослідну роботу відповідно до її профілю, і науково-дослідну роботу в галузі теорії й методики освіти.

Дана робота зосереджена на функціонуванні кафедри, а саме на контролю внутрішніх підрозділів.

Таким чином у базі даних мусять бути такі сутності:

- Викладачі;
- Дисципліни (предмети);
- Спеціальності;
- Кафедри;

1.2 Вивчення потреб і вимог користувачів

`

Вивчення потреб і вимог користувачів є важливим етапом у розробці програмного забезпечення з використанням об'єктно-орієнтованого програмування (ООП). Цей етап спрямований на збір і аналіз вимог, які стосуються функціональності, характеристик та властивостей програмного продукту з точки зору користувачів.

Під час вивчення потреб і вимог користувачів розробник взаємодіє з потенційними або фактичними користувачами системи, бізнес-аналітиками, продуктовими власниками та іншими стейкхолдерами. Основними завданнями цього етапу є:

- *Збір вимог*: Розробник збирає вимоги щодо того, як повинна функціонувати програма, які функції вона повинна виконувати, які задачі вона має вирішувати. Це можуть бути функціональні вимоги (наприклад, можливість створення, редагування та видалення об'єктів) або нефункціональні вимоги (наприклад, продуктивність, безпека, надійність).
- *Аналіз вимог*: Розробник аналізує та уточнює вимоги, розуміючи контекст і потреби користувачів. Важливо зрозуміти, як програмне забезпечення буде використовуватися, які сценарії взаємодії мають бути підтримані, які функції є найважливішими та при яких умовах система має працювати.
- *Формулювання вимог*: Розробник формулює вимоги у вигляді чітких, зрозумілих і вимірюваних характеристик. Вимоги повинні бути абсолютно зрозумілими для всіх зацікавлених сторін, включаючи розробників, тестувальників та користувачів.
- *Валідація вимог*: Перевірка та підтвердження вимог зі стейкхолдерами, щоб переконатися, що вони відповідають потребам користувачів і відображають реальні вимоги бізнесу.

`

Вивчення потреб і вимог користувачів дозволяє розробникам зрозуміти цільову аудиторію та створити програмне забезпечення, яке задовольняє їхні потреби та вимоги. Цей процес є важливим для успішної розробки програмного продукту з використанням ООП, оскільки він допомагає забезпечити, що програма буде ефективно працювати відповідно до очікувань користувачів та бізнесу.

Для менеджменту кафедри ВНЗ користувачу потрібно надати змогу виконувати операції створення, видалення, редагування, перегляд над такими одиницями кафедри як:

- Викладачі;
- Спеціальності;
- Предмети;
- Кафедри;

1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення

Визначення функціональних та нефункціональних вимог до програмного забезпечення є важливим етапом у розробці системи з використанням об'єктно-орієнтованого програмування (ООП). Ці вимоги визначають функціональність, характеристики та властивості програмного продукту.

- *Функціональні вимоги:*

Функціональні вимоги описують, які функції та операції повинна виконувати програма. Вони визначають, як програмне забезпечення має взаємодіяти з користувачем, обробляти дані, виконувати розрахунки та забезпечувати необхідну функціональність. Приклади функціональних вимог включають:

`

- Реєстрація та авторизація користувачів.
- Створення, читання, оновлення та видалення даних.
- Виконання певних операцій та обчислень.
- Взаємодія з іншими системами або компонентами.
- Забезпечення доступу до функцій відповідно до рівнів привілеїв користувачів

Ось кілька прикладів функціональних вимог, які я склав для програми:

- Як користувач, я хочу мати змогу видаляти спеціальності;
- Як користувач, я хочу мати змогу додавати спеціальності;
- Як користувач, я хочу мати змогу переглядати спеціальності;
- Як користувач, я хочу мати змогу оновлювати наявні спеціальності;
- Як користувач, я хочу мати змогу видаляти викладачів;
- Як користувач, я хочу мати змогу додавати викладачів;
- Як користувач, я хочу мати змогу переглядати викладачів;
- Як користувач, я хочу мати змогу оновлювати наявних викладачів;

- *Нефункціональні вимоги:*

Нефункціональні вимоги визначають характеристики та властивості програмного продукту, які не стосуються конкретної функціональності. Вони визначають якість, продуктивність, надійність та інші аспекти системи. Приклади нефункціональних вимог включають:

- *Продуктивність*: швидкість реакції системи, обсяг обробки даних, час відповіді.
- *Надійність*: стабільність системи, відновлюваність після збоїв, обробка помилок.
- *Безпека*: захист даних, контроль доступу, шифрування.
- *Сумісність*: здатність взаємодіяти з іншими системами або компонентами.

`

- *Удосконаленість:* здатність до розширення та модифікації.
- *Зручність використання*: інтерфейс користувача, документація, підтримка.

Визначення функціональних та нефункціональних вимог до програмного забезпечення допомагає уточнити, які функції має виконувати програма та які вимоги повинні бути враховані для досягнення бажаної якості та задоволення потреб користувачів. Це дозволяє розробникам визначити обсяг робіт, створити план розробки та оцінити успішність системи під час її реалізації та впровадження.

Ось кілька прикладів нефункціональних вимог, які я склав для програми:

- Як користувач, я хочу мати забезпечену надійність системи;
- Як користувач, я хочу мати забезпечену стабільність та неперервність працездатності системи без виникнення частих помилок або збоїв;
- Як користувач, я хочу мати забезпечення швидкого доступу до інформації про спеціальності, викладачів та предмети.
- Як користувач, я хочу мати забезпечення швидкого доступу до інформації про спеціальності, викладачів та предмети.

`

2. АРХІТЕКТУРНИЙ ДИЗАЙН. ПРОЕКТУВАННЯ: ERD, ODD

2.1 Визначення об’єктно-орієнтованої доменної моделі системи

Об'єктно-орієнтована доменна модель системи є концептуальним зображенням структури та поведінки системи, яке використовує об'єктно-орієнтовану парадигму програмування. Вона допомагає розуміти взаємозв'язки між різними елементами системи, моделювати їх властивості та поведінку, ідентифікувати ключові сутності та їх взаємодію.

Основні складові об'єктно-орієнтованої доменної моделі системи:

- *Класи*: Класи є основними будівельними блоками об'єктно-орієнтованої доменної моделі. Вони представляють сутності або об'єкти, які мають спільні властивості, поведінку та стосунки з іншими класами. Класи описують структуру об'єктів, включаючи атрибути (властивості) та методи (операції).
- *Об'єкти*: Об'єкти є конкретними інстанціями класів і представляють конкретні сутності або екземпляри. Кожен об'єкт має свій унікальний стан (значення атрибутів) та поведінку (методи, які він може виконувати).
- *Відношення між класами*: Доменна модель відображає відношення між класами, такі як агрегація, композиція, спадкування (унаслідування), асоціація та залежність. Ці відношення визначають, як класи взаємодіють один з одним, обмінюються даними та спільно працюють.
- *Атрибути та методи*: Класи мають атрибути (властивості), які відображають стан об'єктів, і методи (операції), які визначають їх поведінку. Атрибути можуть бути простими типами даних або посиланнями на інші класи. Методи визначають, як об'єкти можуть виконувати операції та взаємодіяти з іншими об'єктами.
- *Успадкування*: Успадкування дозволяє класам успадковувати властивості та методи від батьківських класів. Це сприяє поліпшенню використання

`

коду, реорганізації спільного функціоналу та забезпеченню більшої зрозумілості структури моделі.

- *Поліморфізм*: Поліморфізм дозволяє об'єктам класу використовувати методи, визначені в їх батьківських класах, з різною поведінкою. Це означає, що різні об'єкти можуть викликати одну й ту ж методу, але виконувати її по-різному.

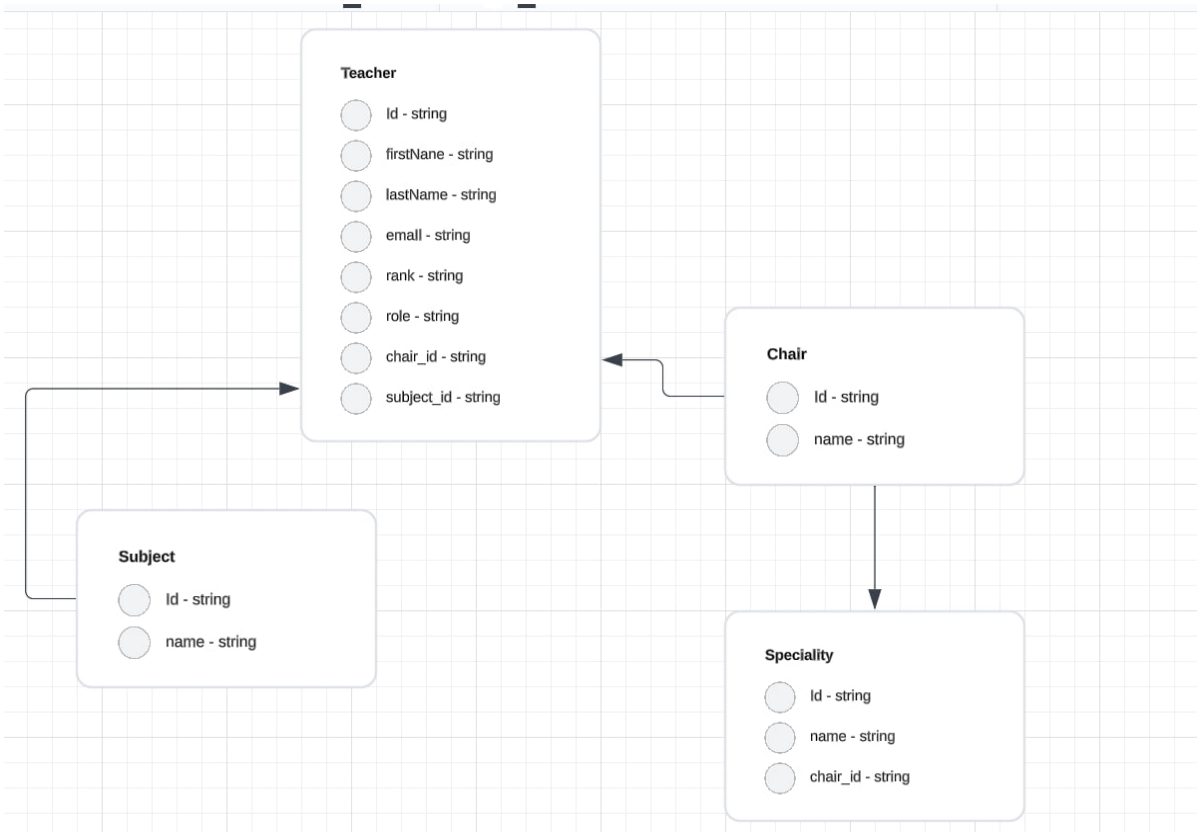

Рисунок 2.1 – «Доменна модель програми»

Доменна модель програми, складається з чотирьох класів: «Teacher», «Chair», «Subject», «Speciality». Клас «Teacher» є головний класом у програмі він звязаний з класом «Subject» по полю-ідентифікатору «subject_id» та з «Chair» по «chair_id». Даний трюк дозволяє прив’язувати конкретний предмет і конекретну кафедру до певного викладача. Клас «Speciality» пов’язаний з класом «Chair» за полем «chair_id», що дозволяє відносити спеціальності до кафедри ВНЗ і надає змогу діставати спеціальності за кафедрами.

`

2.2 Розробка ERD для моделювання схеми збережених даних

Розробка ERD (Entity-Relationship Diagram) для моделювання схеми збережених даних є важливою частиною курсової роботи з ООП. Цей процес допомагає вам визначити структуру та взаємозв'язки між сутностями вашої системи збереження даних. Нижче наведено кроки, які можна виконати для розробки ERD:

- *Визначення сутностей:*
  - Спочатку ідентифікуйте основні сутності вашої системи. Наприклад, якщо ви моделюєте систему управління кафедрою, можливими сутностями можуть бути «Викладач», «Кафедра», «Предмет» і «Спеціальність».
- *Визначення атрибутів:*
  - Для кожної сутності визначте її атрибути (характеристики). Наприклад, для сутності «Викладач» атрибути можуть включати «Прізвище», «Ім’я», «Ранг» та «Емейл».
- *Визначення взаємозв'язків*:
  - Встановіть взаємозв'язки між сутностями. Наприклад, зв'язок між «Викладачем» та «Предметом» може бути «один до багатьох», оскільки кожен викладач може викладати кілька предметів.
- *Визначення первинних ключів:*
  - Для кожної сутності виберіть атрибут(и), які є унікальним ідентифікатором цієї сутності. Цей атрибут(и) буде первинним ключем. Наприклад, у сутності «Викладач» атрибут «Id», «Email» може бути первинним ключем.
- *Побудова ERD:*
  - Використовуйте спеціальні символи та нотацію для побудови діаграми ERD, яка відображає сутності, атрибути, взаємозв'язки та ключі вашої системи збереження даних.

`

- *Валідація та оптимізація ERD*:
  - Перегляньте ERD, щоб переконатися, що всі взаємозв'язки та атрибути правильно відображені. Виконайте оптимізацію ERD, якщо необхідно, для покращення продуктивності та ефективності вашої схеми збереження даних.

Цей процес допомагає вам розуміти структуру та зв'язки між сутностями вашої системи збереження даних і є важливим кроком у розробці програмного забезпечення з використанням ООП.

Ось приклад ERD діаграми для функіонування кафедри ВНЗ:

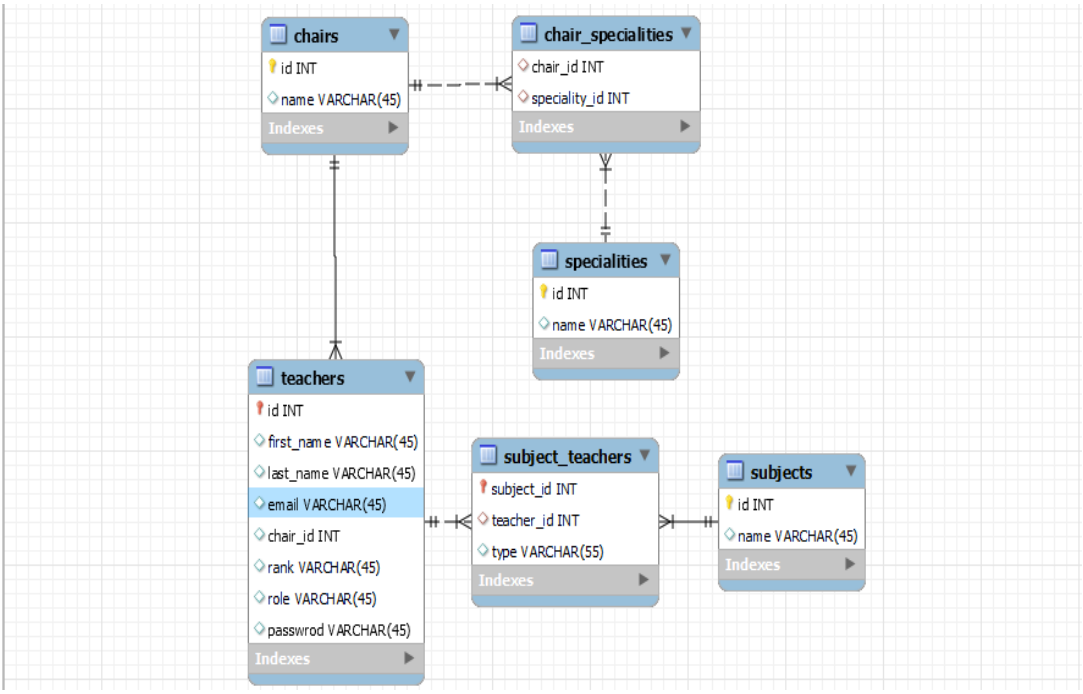

Рисунок 2.2 – «ERD діаграма»

`

2.3 Визначення основних класів, об'єктів та їх взаємодії в системі

Визначення основних класів, об'єктів та їх взаємодії в системі є ключовим аспектом об'єктно-орієнтованого програмування. У цій парадигмі програмування система моделюється як набір взаємодіючих об'єктів, кожен з яких має свої властивості і поведінку.

Основні поняття в ООП:

- *Клас*: Клас є шаблоном або описом, який визначає структуру об'єкта. Він визначає, які дані можуть міститися в об'єкті (властивості або поля) та які операції можуть бути виконані над цим об'єктом (методи). Наприклад, якщо ви пишете програму для керування кафедрами, можливо, у вас є клас «Викладач», який має властивості, такі як прізвище, ім’я, ранг тощо.
- *Об'єкт*: Об'єкт є конкретною реалізацією класу. Він є екземпляром класу, створеним з використанням конструктора класу. Кожен об'єкт має свій стан (значення властивостей) і поведінку (реалізовану методами). У нашому прикладі з кафедрами, об'єкт «Викладач» може бути конкретним викдадачем з унікальними властивостями, такими як електронна скринька, ідентифікатор тощо.
- *Взаємодія*: Об'єкти взаємодіють один з одним, виконуючи методи та обмінюючи інформацію. Це може бути досягнуто за допомогою виклику методів одного об'єкта з іншого об'єкта або шляхом обміну повідомленнями. Наприклад, у кафдері об'єкт «Викладач» може взаємодіяти з об'єктом «Кафедра» через методи прив’язки книги.

Проектуючи систему або програму, ви повинні розглянути, які класи будуть присутні в системі, які об'єкти ці класи створюють і як вони взаємодіють один з одним для досягнення поставлених цілей. Це допоможе вам

`
організувати код в логічні блоки і полегшити розробку та підтримку вашої програми.

У програмі для менеджменту кафедр ВНЗ потрібно 4 головних класи:

- Teacher;
- Speciality;
- Subject;
- Chair;

Ці класи мають головну позицію у функціонуванні програми, без них програма не матиме сенсу, оскільки кожен із них є невід’ємною частиною кафедри.

2.4 Розробка діаграм класів, яка показує структуру класів і їх взаємозв'язки

Діаграма класів є графічним інструментом для візуалізації структури класів і їх взаємозв'язків у системі. Вона дозволяє лаконічно представити класи, їх властивості (поля) і методи, а також зв'язки між класами, такі як спадкування, агрегація або асоціація.

Основні елементи діаграми класів:

- *Класи:* Вони представлені у діаграмі прямокутниками, в яких зазначається назва класу.
- *Атрибути (поля)*: Це властивості класу, які відображаються в прямокутнику класу або над ним. Атрибути можуть мати назву, тип даних та модифікатор доступу (наприклад, приватний, захищений або публічний).
- *Методи*: Вони представлені у діаграмі так само, як і атрибути, але зазвичай вказуються в дужках під назвою класу. Методи описують поведінку класу і можуть приймати аргументи та повертати значення.

`

- *Зв'язки між класами*: Вони показують взаємозв'язки між класами і можуть мати різні типи, такі як спадкування, агрегація, асоціація або залежність.
- *Модифікатори доступу*: Вони вказують рівень доступу до класів, атрибутів та методів, такі як публічний (+), приватний (-) або захищений (#).

Діаграма класів допомагає розуміти структуру системи, ідентифікувати класи та їх зв'язки.

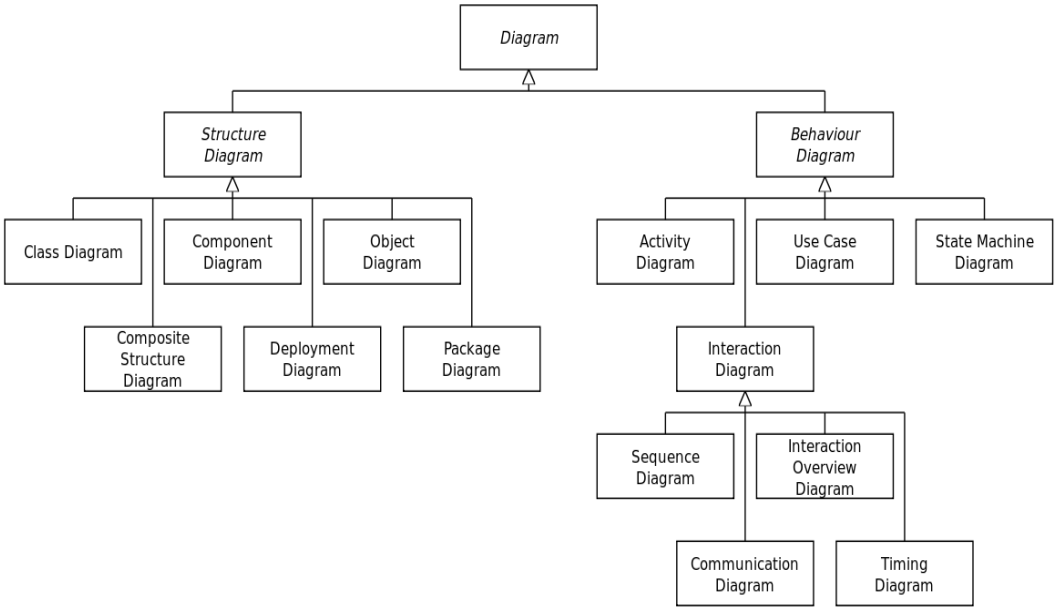

Рисунок 2.3 – «Діаграма класів»

`

3 ІМПЛЕМЕНТАЦІЯ ТА ТЕСТУВАННЯ

3.1 Написання програмного коду відповідно до проектування і вимог до системи

Процес написання програмного коду відповідно до проектування і вимог до системи включає кілька етапів. Основною метою цього процесу є розробка ефективного, читабельного і легко розширюваного програмного коду, який задовольняє вимоги до системи, описані у проектувальній документації.

Під час розробки нашої програми слід добре ознайомитись з вимогами до нашого програмного забезпечення, оскільки слід наперед враховувати багато моментів, щоб готовий продукт повністю відповідав зазначеним критеріям. Тільки найкращим розробникам вдається дотримуватися всіх вимог і при цьому підтримувати чистоту коду і безперебійний функціонал програми.

Для виконання курсової роботи було складено функціональні і нефункціональні вимоги, описані в першому розділі роботи. При розробці програми я старався максимально притримуватись всіх зазначених вимог.

Ось приклади розобки програми з дотримуванням критеріїв зв’язаних з спеціальностями:

- Як користувач, я хочу мати змогу видаляти спеціальності;
- Як користувач, я хочу мати змогу додавати спеціальності;
- Як користувач, я хочу мати змогу переглядати спеціальності;
- Як користувач, я хочу мати змогу оновлювати наявні;

Ось фрагмент коду контроллера на Java з використанням Spring Boot, який відповідає за цей функціонал.

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/speciality")
public class SpecialityController {
```

```
`
            private final SpecialityRepository specialityRepository;

            @GetMapping

            public List<Speciality> getAllSpecialities() {

                return specialityRepository.findAll();

            }

            @GetMapping("/{id}")

            public ResponseEntity<Speciality> getSpecialityById(@PathVariable Long id) {

                Optional<Speciality> speciality = specialityRepository.findById(id);

                return
speciality.map(ResponseEntity::ok).orElse(ResponseEntity.notFound().build());

            }

            @PostMapping

            public ResponseEntity<Speciality> createSpeciality(@Valid @RequestBody Speciality
speciality) {

                Speciality savedSpeciality = specialityRepository.save(speciality);

                URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")

                        .buildAndExpand(savedSpeciality.getId()).toUri();

                return ResponseEntity.created(location).body(savedSpeciality);

            }

            @PutMapping("/{id}")

            public ResponseEntity<Speciality> updateSpeciality(@PathVariable Long id,

                                                               @Valid @RequestBody Speciality
speciality) {

                Optional<Speciality> existingSpeciality = specialityRepository.findById(id);

                if (existingSpeciality.isPresent()) {

                    speciality.setId(id);

                    Speciality updatedSpeciality = specialityRepository.save(speciality);

                    return ResponseEntity.ok(updatedSpeciality);

                }

                return ResponseEntity.notFound().build();
```

```
`
            }

            @DeleteMapping("/{id}")

            public ResponseEntity<Void> deleteSubject(@PathVariable Long id) {

                Optional<Speciality> existingSubject = specialityRepository.findById(id);

                if (existingSubject.isPresent()) {

                    specialityRepository.deleteById(id);

                    return ResponseEntity.noContent().build();

                }

                return ResponseEntity.notFound().build();

            }

        }
```

Наприклад, метод, що знаходиться під *@GetMapping* відповідає за виконання вимоги перегляду спеціальностей, наступний метод під анатацією *@PostMapping* - відповідає за створення, *@PutMapping* – за оновлення, *@DeleteMapping* – за видалення. Це далеко не весь код для функціонування спеціальностей, крім цього потрібно створити репозиторій та сервіс див. Додаток Б.

3.2 Реалізація класів, об'єктів та їх методів на Java

У архітектурі програми для менеджменту кафедр ВНЗ, ChairApplication.java є серцем продукту і відповідає за об’єднання всіх процесів у Java Spring Boot.

У цьому класі використовується анотація @SpringBootApplication, яка вказує на те, що цей клас є основним класом додатку та містить в ньому точку входу.

Основним методом у класі ChairApplication є метод main, який є точкою входу для виконання додатку. В цьому методі використовується статичний метод run з класу SpringApplication для запуску додатку. Метод run отримує два

`

аргументи: клас ChairApplication.class, який вказує на основний клас додатку, і args, який представляє масив аргументів командного рядка.

Після запуску додатку, Spring Boot виконує ряд процесів. Основний процес, що відбувається в ChairApplication, включає такі кроки:

- *Створення та налаштування контейнера інверсії керування (IoC).* Spring Boot використовує IoC контейнер для керування створенням та управлінням об'єктами в додатку.
- *Сканування класів та компонентів додатку*. Spring Boot автоматично сканує пакети, вказані в основному класі ChairApplication, для пошуку класів-компонентів, таких як контролери, сервіси та репозиторії.
- *Конфігурування та ініціалізація бінів*. Spring Boot на основі сканування класів автоматично конфігурує біни (об'єкти) та встановлює їх залежності.
- *Завантаження налаштувань додатку*. Spring Boot намагається завантажити конфігураційні файли та налаштування для додатку. Це можуть бути файли типу application.properties або application.yml, або інші варіанти налаштувань.
- *Запуск веб-сервера*. При наявності залежностей, що підключають вбудований веб-сервер, Spring Boot запускає його, дозволяючи додатку обробляти HTTP-запити.

Після успішного запуску додатку ви зможете звертатися до його ресурсів, використовувати веб-сервіси, взаємодіяти з базою даних та іншими компонентами, що були налаштовані та створені під час процесу запуску.

Ось код самого класу:

```
@SpringBootApplication
public class ChairApplication {
	public static void main(String[] args) {
		SpringApplication.run(ChairApplication.class, args);
	}
}
```

`

3.3 Результати функціонування програми

Результати функціонування програми є важливими з точки зору оцінки її продуктивності, якості та ефективності. Ось кілька причин, для чого потрібні результати функціонування програми:

- *Виявлення помилок і проблем*: Результати функціонування програми допомагають виявляти помилки, баги та неполадки, які можуть виникати під час виконання програми. Це дає змогу розробникам виправити проблеми та забезпечити належне функціонування програми.
- *Оцінка продуктивності*: Результати функціонування програми дозволяють оцінити продуктивність програми, включаючи швидкість виконання, використання ресурсів (таких як процесор, пам'ять, мережа) і ефективність роботи програми в цілому. Це важливо для забезпечення оптимальної продуктивності та покращення її якості.
- *Верифікація та тестування*: Результати функціонування програми використовуються для верифікації її правильності та тестування. Розробники можуть порівняти очікувані результати зі здобутими для переконанняся, що програма виконує заплановані функції та поводиться належним чином у різних сценаріях.

У проєкті менеджмент кафедри ВНЗ, тестуватись програма буде у середовищі PostMan, який дозволяє робити HTTP запити до наших ендпоїнтів і відображати результати виконання поставлених задач. Для того, щоб вдало виконувати запити потібно налаштувати PostMan, а саме вказати URL-адресу нашого серверу у даному випадку це locahost з портом 8080, наступним кроком буде вказування типу даних, які відправляються на сервер – потрібно поставити JSON.

Для початку перевіримо чи працює створення у кожної кафедри ВНЗ та відобразимо результати у вигляді скрішотів:

- Створення кафедри (див. Рисунок 3.1)

`

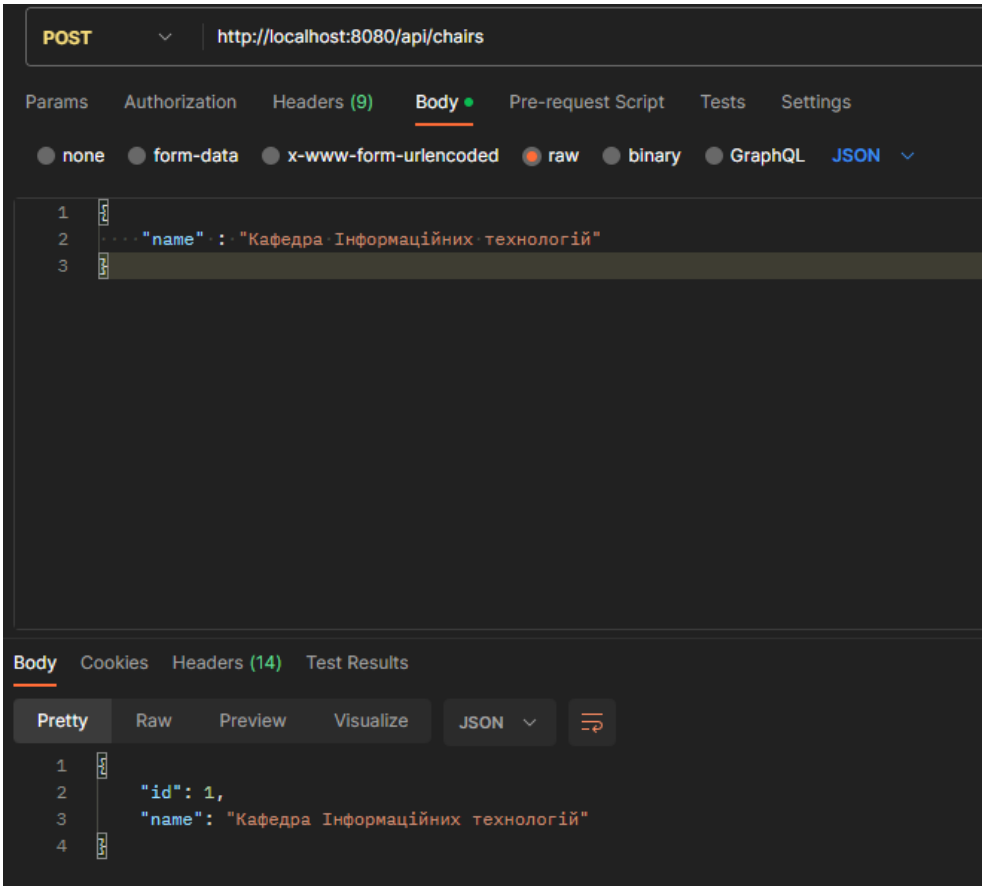

Рисунок 3.1 – «Результат виконання запиту створення кафедри»

- Створення спеціальності (див. Рисунок Б.1)
- Створення предмету (див. Рисунок Б.2)
- Створення викладача (див. Рисунок Б.3)

У вдалому POST запиті у відповідь ми отримуємо об’єкт, який ми відправляли і код 201.

Настпним кроком є отримання створених даних:

- Отримання існуючих кафедр (див. Рисунок 3.2):

`

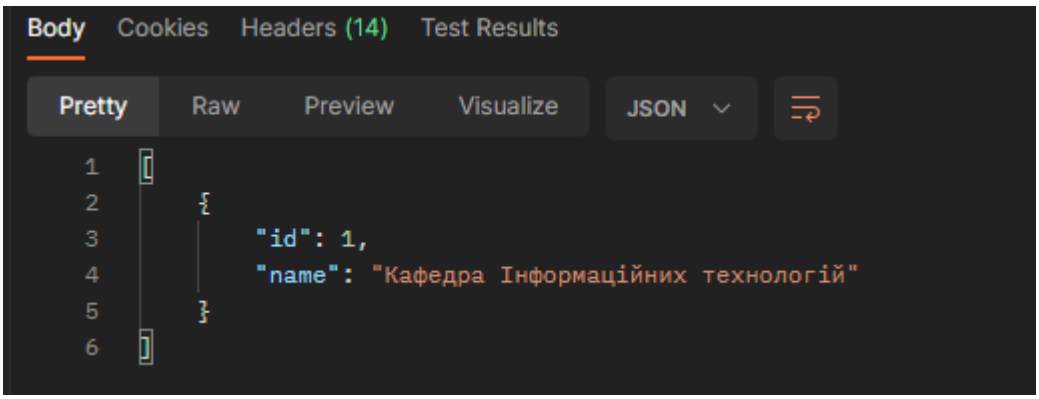

Рисунок 3.2 – «Результат виконання запиту отримання кафедр»

- Отримання існуючих спеціальностей (див. Рисунок Б.4);
- Отримання існуючих предметів (див. Рисунок Б.5);
- Отримання існуючих викладачів (див. Рисунок Б.6):

У вдалому GET запиті у відповідь ми отримуємо дані і код 200.

Настпним кроком є оновлення даних:

- Оновлення існуючої кафедри (див. Рисунок 3.3):

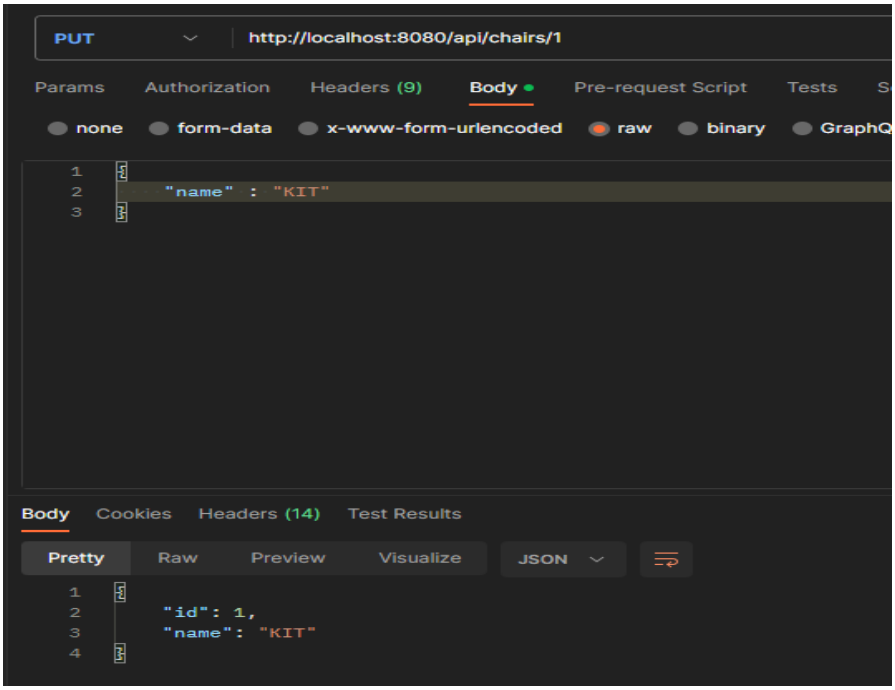

Рисунок 3.3 – «Результат виконання запиту оновлення кафедри»

- Оновлення існуючої спеціальності (див. Рисунок Б.7)

`

- Оновлення існуючого предмету (див. Рисунок Б.8)
- Оновлення існуючого викладача (див. Рисунок Б.9)

У вдалому PUT запиті у відповідь ми отримуємо оновлені дані і код 201.

Настпним кроком є отримання видалення даних:

- Видалення кафедри (див. Рисунок 3.4)

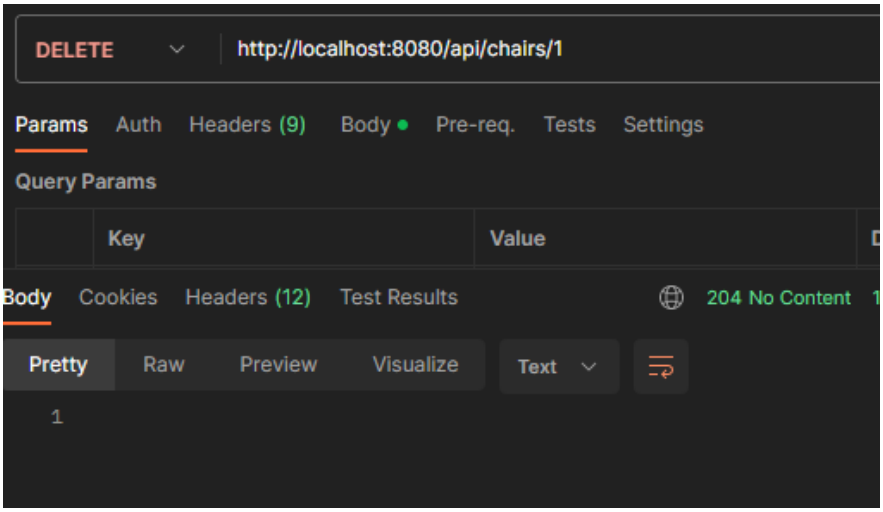

Рисунок 3.4 – «Результат виконання запиту отримання кафедр»

- Видалення спеціальності (див. Рисунок Б.10)
- Видалення предмету (див. Рисунок Б.11)
- Видалення викладача (див. Рисунок Б.12)

У вдалому DELETE запиті у відповідь ми отримуємо код 204.

`

ВИСНОВКИ

Під час процесу розроблення даного проєкту було створену програму для менеджменту кафедр ВНЗ. У даній роботі як основу було використано ООП у реалізації на Java Spring Boot також використовувалася MySql для побудови ER діаграми. За допомогою цього середовища та фреймворку було створено ряд ендпоїнтів, які дозволяють виконувати мінімальні потреби кафедри згідно до вимог ПЗ. Розроблений продукт не може претендувати на повне охоплення всієї області роботи кафедр, але основна мета – отримання доступу до менджементу таких одиниць як викладачі, спеціальності, предмети та кафедри була досягнена.

`

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Spring Framework

   URL: https://uk.wikipedia.org/wiki/Spring_Framework (дата звернення: 24.05.2023)

2. Об’єктно орієнтоване програмування

   URL:https://uk.wikipedia.org/wiki/Об'єктно_орієнтоване_програмування (дата звернення: 25.05.2023)

3. Java

   URL: https://uk.wikipedia.org/wiki/Java (дата звернення: 26.05.2023)

4. SQL

   URL:https://uk.wikipedia.org/wiki/SQL (дата звернення: 26.05.2023)

5. OOD

   URL: https://en.wikipedia.org/wiki/Object-oriented_design (дата звернення: 27.05.2023)

6. Workbench

   URL:https://uk.wikipedia.org/wiki/MySQL_Workbench (дата звернення: 27.05.2023)

7. OpenAI

   URL: https://chat.openai.com/ (дата звернення: 27.05.2023)

8. ER-diagram

   URL:https://www.lucidchart.com/pages/er-diagrams (дата звернення: 28.05.2023)

ДОДАТОК А

Посилання на репозиторій Git-Hub: https://github.com/mesnuk/kursova_sovtus

ДОДАТОК Б

Репозиторій програми (файл SpecialityRepository.java)

```
@Repository
public interface SpecialityRepository extends JpaRepository<Speciality, Long> {

}
```

Сервіс програми (файл Speciality.java)

```
public class Speciality {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name", nullable = false)
    private String name;
}
```

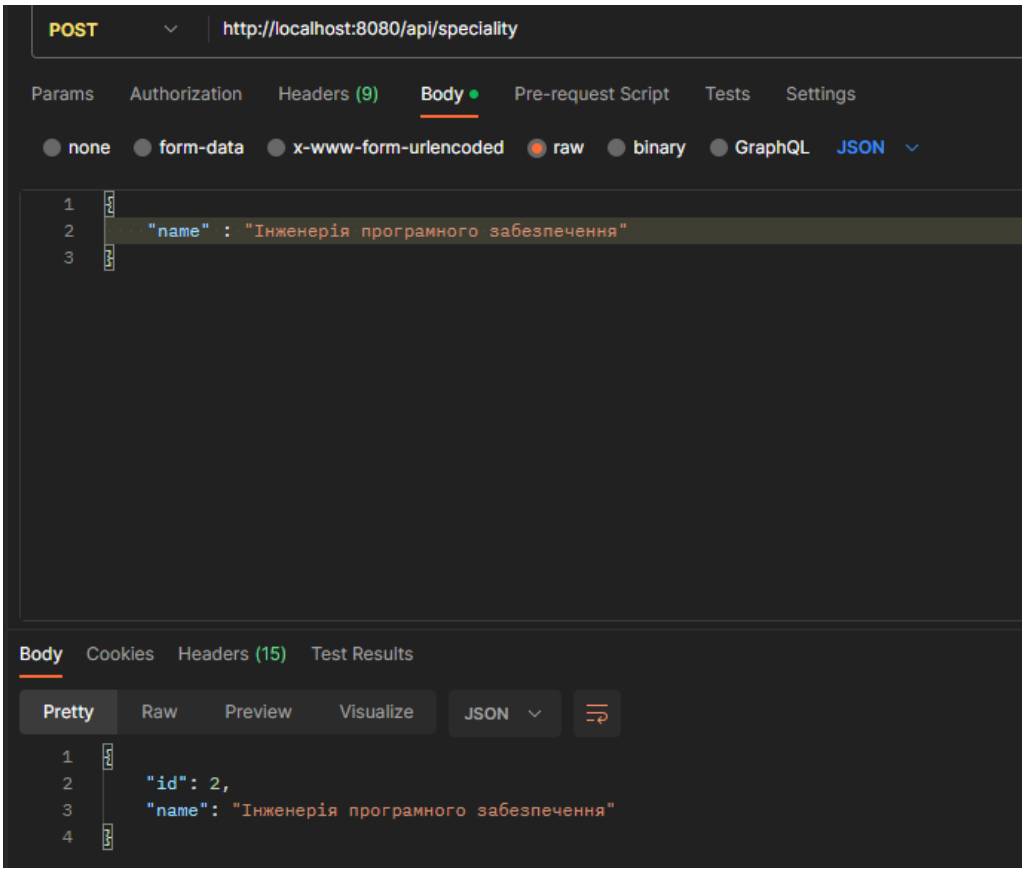

Рисунок Б.1 – «Результат виконання запиту створення спеціальності»

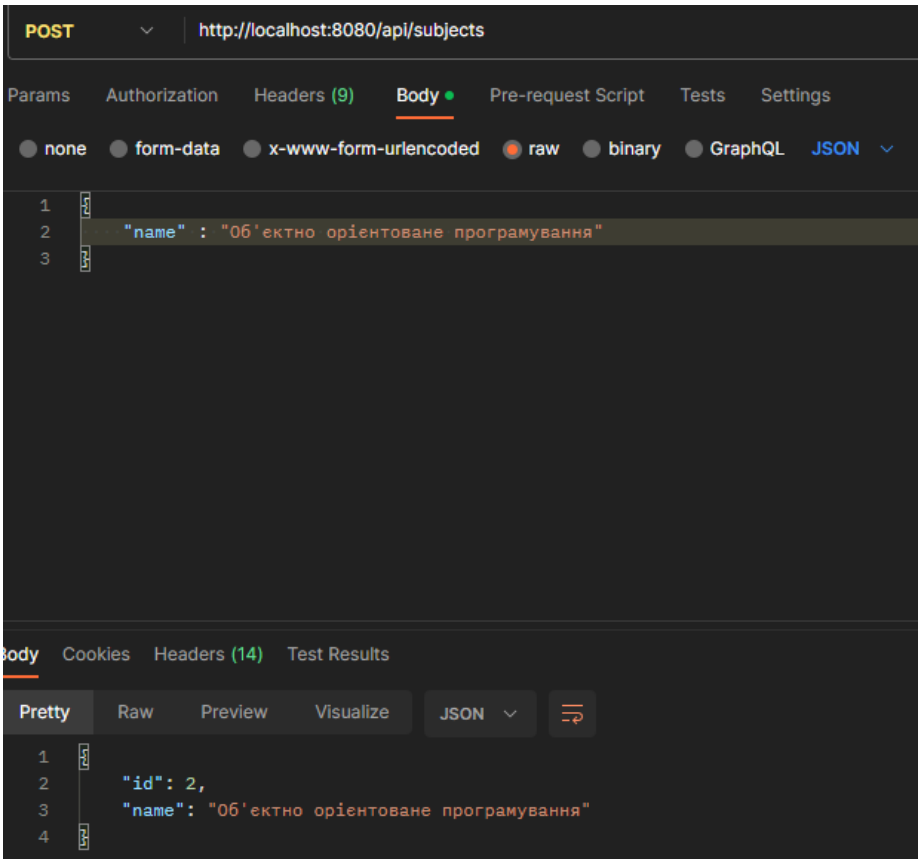

Рисунок Б.2 – «Результат виконання запиту створення предмету»

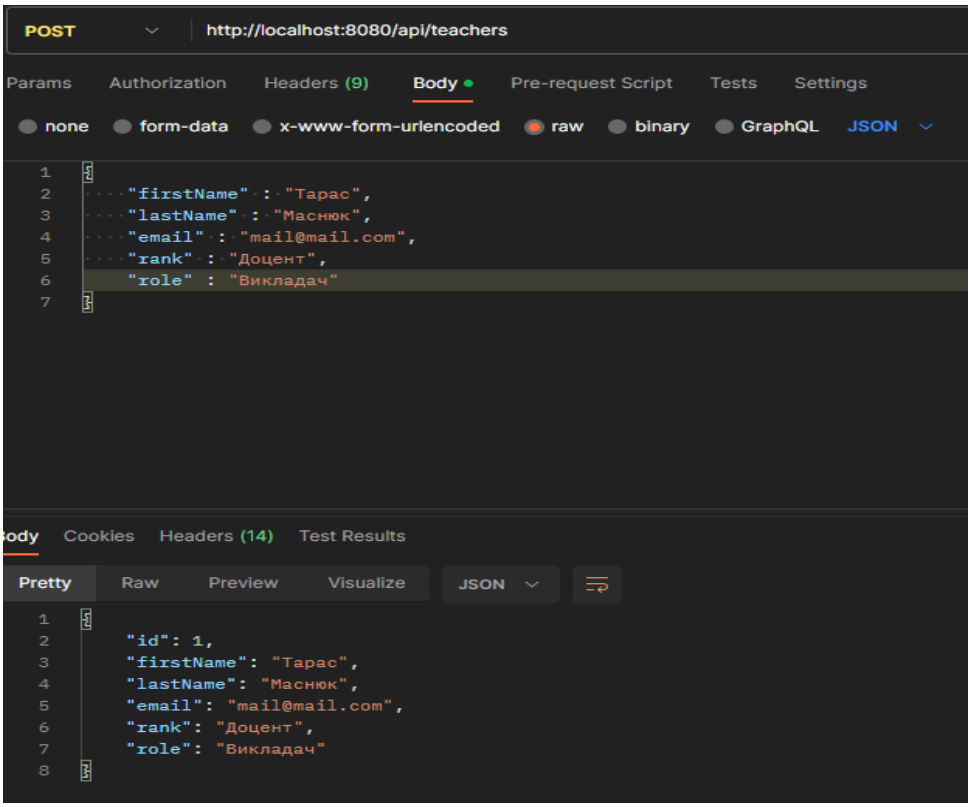

Рисунок Б.3 – «Результат виконання запиту створення викладача»

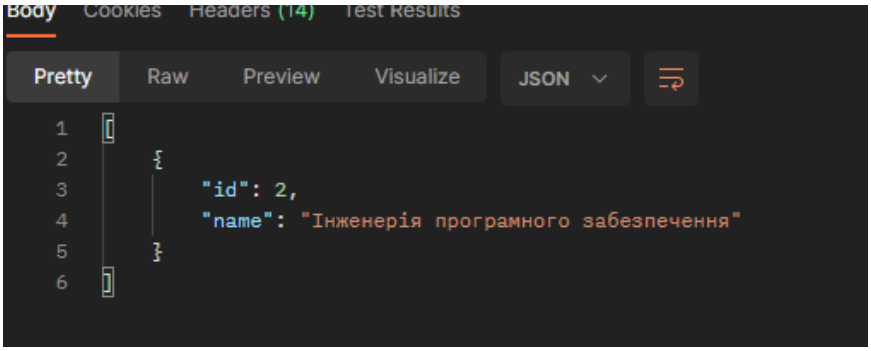

Рисунок Б.4 – «Результат виконання запиту отримання спеціальностей»

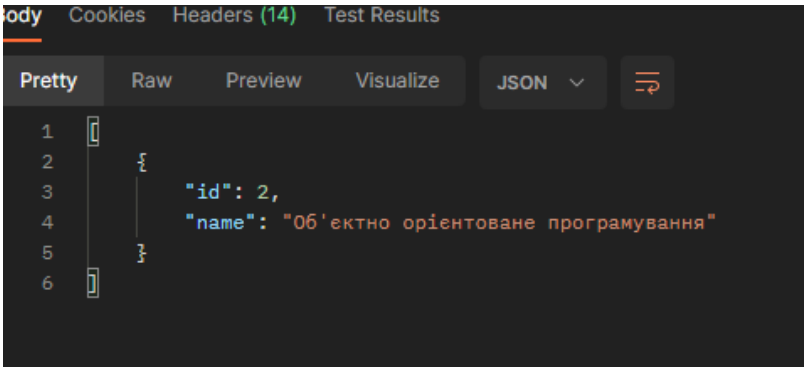

Рисунок Б.5 – «Результат виконання запиту отримання предметів»

Рисунок Б.6 – «Результат виконання запиту отримання викладачів»

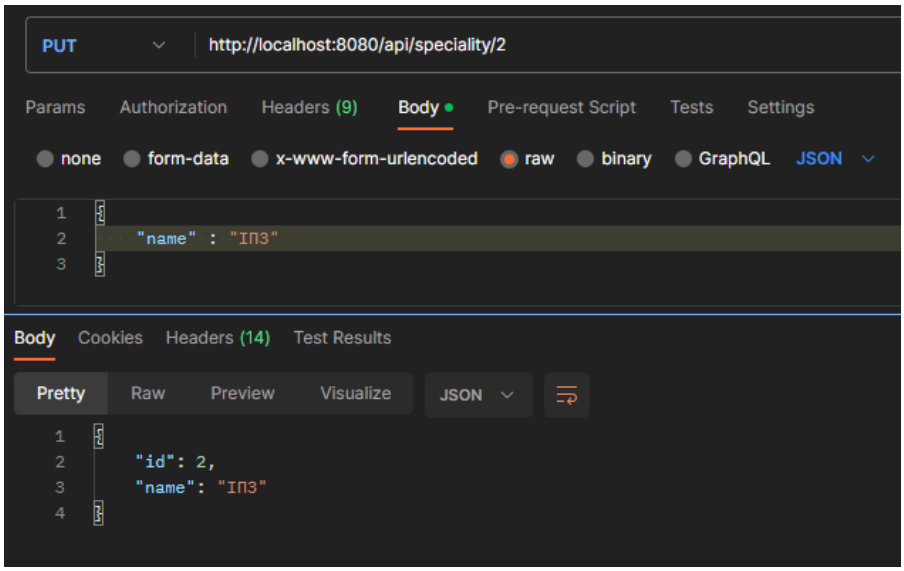

Рисунок Б.7 – «Результат оновлення існуючої спеціальності»

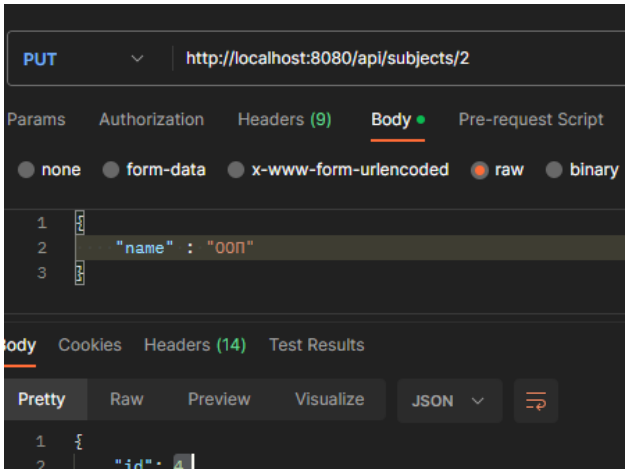

Рисунок Б.8 – «Результат оновлення існуючої предмету»

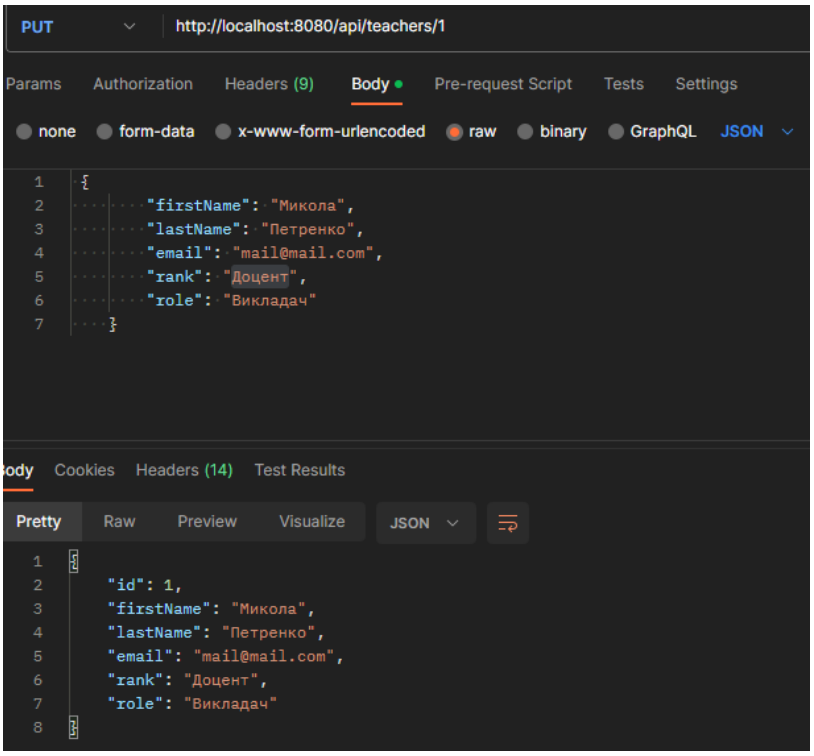

Рисунок Б.9 – «Результат оновлення існуючого викладача»

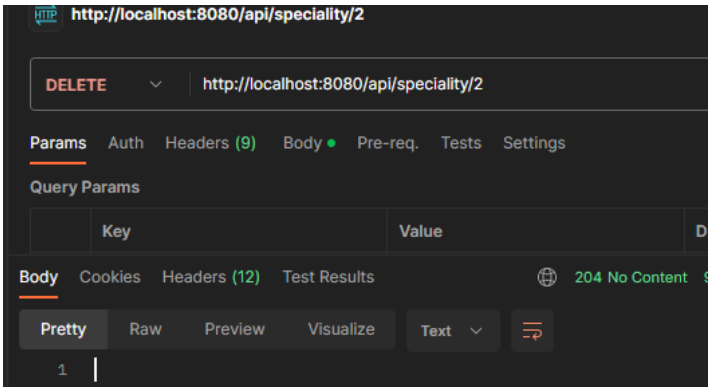

Рисунок Б.10 – «Результат виконання запиту видалення спеціальності»

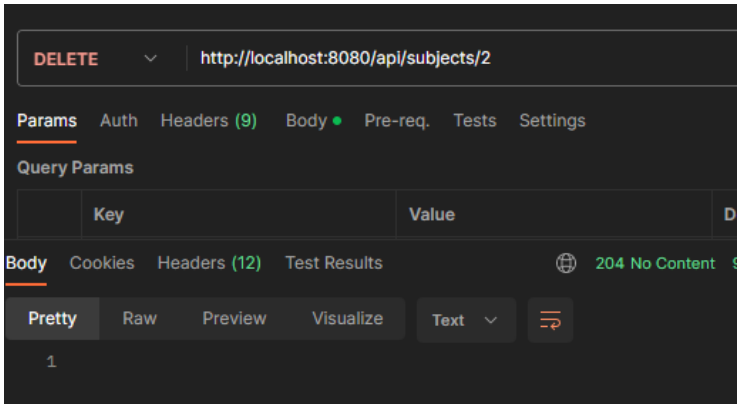

Рисунок Б.11 – «Результат виконання запиту видалення предмету»

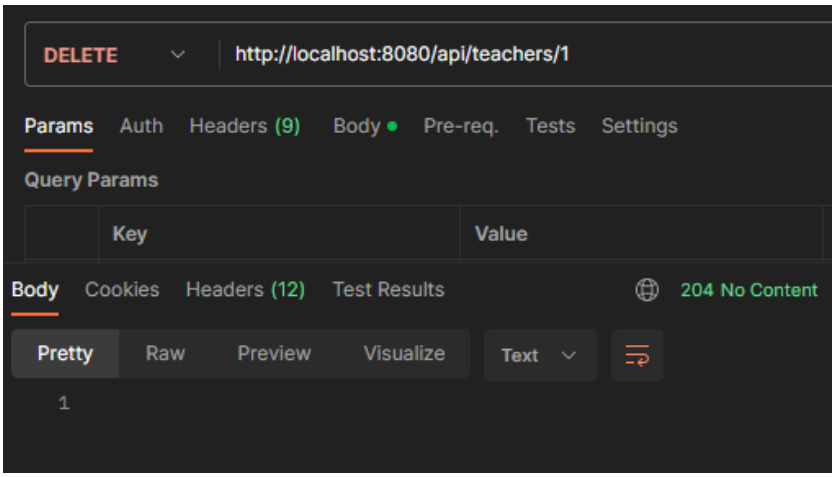

Рисунок Б.12 – «Результат виконання запиту видалення викладача»