ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ

«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ

ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ ІМЕНІ ІВАНА ПУЛЮЯ»

ВІДДІЛЕННЯ ТЕЛЕКОМУНІКАЦІЙ ТА ЕЛЕКТРОННИХ СИСТЕМ

Циклова комісія комп’ютерних наук

**КУРСОВА РОБОТА**

з дисципліни:

**«Розробка клієнт-серверних застосувань»**

на тему: **«Розробка RESTful вебсервісу довіника нумізмата»**

Студента  4  курсу групи КН-421 спеціальності 122 «Комп’ютерні науки»

Вережак О.М.

(прізвище та ініціали)

Керівник: викладач   Слободян  Р.О.         
Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_\_ Руслан СЛОБОДЯН

(підпис)

\_\_\_\_\_\_\_ Роман ДІЛАЙ

(підпис)

м. Тернопіль – 2023

ЗМІСТ

[Вступ 6](#_Toc156565997)

[1 Технічне завдання 7](#_Toc156565998)

[1.1 Найменування та область застосування 7](#_Toc156565999)

[1.2 Підстави для розробки 7](#_Toc156566000)

[1.3 Призначення розробки 7](#_Toc156566001)

[1.4 Вимоги до програмного забезпечення 7](#_Toc156566002)

[1.5 Вимоги до програмної документації 9](#_Toc156566003)

[1.6 Техніко-економічні показники 10](#_Toc156566004)

[1.7 Стадії та етапи розробки 10](#_Toc156566005)

[1.8 Порядок контролю та прийому 11](#_Toc156566006)

[2 Розробка технічного та робочого проєкту 12](#_Toc156566007)

[2.1 Постановка задачі на розробку програмного забезпечення 12](#_Toc156566008)

[2.2 Проєктування і опис прикладного інтерфейсу сервера 14](#_Toc156566009)

[2.3 Розробка структури бази даних сервера 16](#_Toc156566010)

[2.4 Розробка системи класів 18](#_Toc156566011)

[2.5 Опис форматів запитів до сервера та відповідей сервера 20](#_Toc156566012)

[2.6 Опис файлової структури ПЗ 22](#_Toc156566013)

[3 Тестування програми і результати її виконання 24](#_Toc156566014)

[Висновки 29](#_Toc156566015)

[Перелік посилань 30](#_Toc156566016)

[Додаток А Лістинг файлу «pom.xml» 31](#_Toc156566017)

[Додаток Б Лістинг файлу «countrycontroller.java» 34](#_Toc156566018)

[Додаток В Лістинг файлу «countryresponse.java» 37](#_Toc156566019)

[Додаток Г Лістинг файлу «Countrydto.java» 40](#_Toc156566020)

[Додаток Ґ Лістинг файлу «COUNTRY.java» 41](#_Toc156566021)

[Додаток Д Лістинг файлу «countryservice.java» 43](#_Toc156566022)

[Додаток Е Лістинг файлу «countryserviceimpl.java» 44](#_Toc156566023)

[Додаток Є Лістинг файлу «countryrepository.java» 47](#_Toc156566024)

[Додаток Ж Лістинг файлу «HISTORY.java» 48](#_Toc156566025)

[Додаток З Лістинг файлу «Coin.java» 50](#_Toc156566026)

[Додаток И Лістинг файлу «collector.java» 52](#_Toc156566027)

[Додаток І Лістинг файлу «application.properties» 55](#_Toc156566028)

[Додаток Ї Лістинг файлу «applicationconstants.java» 56](#_Toc156566029)

[Додаток Д Лістинг файлу «employeeservice.java» 45](#_Toc156072672)

[Додаток Е Лістинг файлу «employeeserviceimpl.java» 46](#_Toc156072673)

[Додаток Є Лістинг файлу «employeerepository.java» 50](#_Toc156072674)

[Додаток Ж Лістинг файлу «room.java» 51](#_Toc156072675)

[Додаток З Лістинг файлу «guest.java» 54](#_Toc156072676)

[Додаток И Лістинг файлу «review.java» 57](#_Toc156072677)

[Додаток І Лістинг файлу «role.java» 60](#_Toc156072678)

[Додаток Ї Лістинг файлу «applicationconstants.java» 62](#_Toc156072679)

[Додаток Й Лістинг файлу «application.properties» 63](#_Toc156072680)

Вступ

Сучасна практика програмування вимагає постійного удосконалення методів розробки комп'ютерних програм з метою прискорення та полегшення процесу їх створення. Зокрема, акцент робиться на розробці клієнт-серверних застосувань, де програмні рішення спрямовані на ефективну взаємодію з користувачем. Цей підхід вимагає не лише врахування технологічних аспектів інтерфейсу, але й зосередження на психологічних та фізіологічних особливостях сприймання інформації користувачем.

Зростання приватного сектору економіки призвело до перегляду підходів до витрат на комп'ютерне обладнання та спеціалізоване програмне забезпечення. Ефективніше стало розробляти програмні продукти, спрямовані на вирішення конкретних, вузько спеціалізованих завдань. Таким чином, освітні заклади ставлять перед собою завдання підготовки фахівців, здатних створювати гнучкі програмні рішення, ефективно використовуючи ресурси і не витрачаючи надто багато фінансів та часу на їхню розробку та впровадження.

Для ілюстрації, у даному курсі демонструється розробка одного з таких спеціалізованих програмних рішень за допомогою мови програмування Java та з використанням принципів розробки клієнт-серверних застосувань. Мета цього проекту полягає в автоматизації та полегшенні роботи адміністратора готелю.

# Технічне завдання

## Найменування та область застосування

Найменування програми – Програма «CourseWork». Область застосування RESTful вебсервіс, який виступає як бекенд системи довідника нумізмата, може мати ряд областей застосування. Цей вебсервіс служить як інтерфейс для взаємодії між фронтендом (інтерфейсом користувача) та базою даних, обробляючи запити і надсилаючи відповіді.

## Підстави для розробки

Підставами для проведення розробки являється індивідуальне завдання на курсову роботу з дисципліни «Розробка клієнт-серверних застосувань».

Найменування теми курсової роботи – «Розробка RESTful вебсервісу довідника нумізмата».

Керівник курсової роботи – викладач Слободян Руслан Олесійович.

Виконавець – студент групи КН-421 Вережак Олександр Миколайович.

## Призначення розробки

Автоматизація цілей включає спрощення, полегшення та підвищення ефективності дослідження довідника нумізмата, а також спрощення процесу поповнення інформації щодо монет та колекціонерів в довіднику нумізмата.

## Вимоги до програмного забезпечення

### Вимоги до функціональних характеристик

RESTful вебсервісу довідника нумізмата повинна забезпечити спрощення виконання таких функцій:

- Облік монет – програма повинна включає зручний механізм обліку існуючих монет. Таблиця містить інформацію про країну випуску та історію виникнення монети. Ця таблиця складається з трьох стовпців, що відповідають кількості монет у довіднику.

- Облік колекціонерів – програма повинна дозволяти додавати, видаляти та редагувати інформацію про колекціонерів, а також зберігати необхідні дані для подальшої роботи програми. Таблиця колекціонерів складається з чотирьох стовпців і рядків, що відповідають кількості існуючих колекціонерів.

- Облік країн – програма повинна включати можливість додавання, видалення та редагування інформації про країни походження монет, а також збереження їх даних. Таблиця країн походження складається з двох стовпців і рядків, відповідних кількості країн.

- Облік історій монет – програма повинна додавати, видаляти, редагувати та зберігати історії монет. Таблиця історій монет складається з двох і рядків, що відповідають кількості історій.

### Вимоги до часових характеристик

Вимоги до часових характеристик веб-сервісу наступні:

* Час відповіді (Response Time) – максимальний час, який сервіс може витратити на обробку запиту та повернення відповіді, має становити менше 500 мс для більшості запитів до веб-сервісу.
* Час виконання операцій бази даних – максимальний час, який веб-сервіс може витрачати на виконання операцій бази даних, які можуть бути необхідні для виконання запитів - менше 200 мс для виконання основних операцій бази даних.

### Вимоги до надійності

Надійність функціонування вебсервісу має бути забезпечено інтуїтивно зрозумілим програмним інтерфейсом та допоміжними повідомленнями.

### Умови до експлуатації

Розроблений програмний продукт не буде потребувати яких-небудь видів особливої експлуатації, для користування достатньо базових навичок користування ПК з під керівництвом дистрибутиву операційної системи Linux.

### Вимоги до складу і параметрів технічних засобів

В склад технічних засобів повинні входити: монітор клавіатура, мишка, та ПК з обсягом дискової пам’яті – 100 ГБ.

### Вимоги до інформаційної і програмної сумісності

Вихідні коди програми повинні бути реалізовані мовою Java. В якості середовища розробки програми повинне використовуватись середовище IntelliJ IDEA версії не нижче 2023 [3].

Системні програми та утиліти, що використовує програма повинні запускатися любою версією Linux з підтримкою PostgreSQL 15 та JRE не нижче 17 версії.

## Вимоги до програмної документації

По закінченню розробки ПЗ потрібно приготувати таку документацію:

* інструкція щодо інсталяції програми;
* загальні відомості про можливості програми.

## Техніко-економічні показники

Розрахунок економічної ефективності і вартості розробки програмного продукту не проводиться. Приблизне число використань розробленої програми в рік – більше 300 раз.

## Стадії та етапи розробки

Розробка програми для довідника нумізмата включає кілька ключових етапів:

 Аналіз**.**

Головною метою етапу аналізу є створення максимально повного опису поставленої задачі. Під час цього етапу активно вивчається предметна область задачі, проводиться об'єктна декомпозиція програми для висвітлення взаємодії між об'єктами, а також визначаються унікальні особливості поведінки цих об'єктів. По завершенню аналізу формується структурна схема програми, яка включає детальний опис об'єктів та подій, що впливають на їхню функціональність.

 Проектування.

Проектування поділяється на дві основні складові: логічну та фізичну. У логічному проектуванні відбувається розробка структури класів, з визначенням полів для збереження компонентів об'єктів та алгоритмів методів, що детально описують їхню поведінку. Фізичне проектування, з свого боку, включає в себе об'єднання опису класів у модулі, вибір схем підключення класів (статична чи динамічна), визначення взаємодії з обладнанням, операційною системою та іншим програмним забезпеченням, таким як бази даних та мережеві середовища. На цьому етапі використовуються прийоми розробки класів, такі як спадкування, композиція та наповнення. Результатом цього етапу є створення ієрархії класів та детальний опис їх функціональності.

 Еволюція.

Етап еволюції системи передбачає поетапну реалізацію та інтеграцію класів у проекті. Процес розпочинається зі створення основного інтерфейсу проекту та бібліотеки класів. На даному етапі розробляється опис полів класів, визначаються алгоритми методів та уточнюються вимоги до програми, вносяться необхідні зміни. Отриманий прототип продукту піддається тестуванню та налагодженню для забезпечення його ефективності та відповідності вимогам.

## Порядок контролю та прийому

Прийом розробленого програмного забезпечення повинен відбуватися на об'єкті Замовника в зазначені в індивідуальному завданні терміни.

Для завершення прийому роботи Виконавець повинен надати на розгляд:

* функціонуючу програму, що повністю відповідає вимогам технічного завдання;
* вихідний програмний код, який записаний разом із програмою на носій інформації. [1]

Прийом програмного забезпечення буде проводитися перед комісією, складеною з двох осіб. Цей процес включатиме наступні етапи:

* доповідь від Виконавця про виконану роботу;
* демонстрація роботи програми Виконавцем;
* проведення контрольних випробувань щодо роботи програмного продукту;
* відповіді на запитання та зауваження комісії. [1]

# Розробка технічного та робочого проєкту

## Постановка задачі на розробку програмного забезпечення

В даному підрозділі пояснювальної записки описується такий етап розробки програми, як аналіз. Метою аналізу є максимально повний опис поставленої задачі, відповідно до технічного завдання.

Потрібно створити REST API вебсервіс довідника нумізмата.

Під час проходження етапу аналізу спочатку було визначено такі питання:

 Хто буде діючими суб'єктами?

 Хто вводитиме інформацію?

 Чи буде хто-небудь ще взаємодіяти з програмою?

 Чи буде сама програма взаємодіяти з іншими програмами?

З програмою буде взаємодіяти один суб’єкт. Він має змогу додавати монети, їх країну походження та історію виникнення а також колекціонерів до бази даних.

Наступне, що потрібно було з'ясувати, це групу дій, які ініціюватиме діючий суб'єкт. Діючий суб'єкт буде виконувати такі дії:

 почати роботу з програмою;

 ввести дані про країну походження монети;

 ввести дані про історію виникнення монети;

 ввести дані про монету;

 ввести дані колекціонерів;

 вивести таблицю країн походження;

 вивести таблицю історій виникнення;

 вивести таблицю монет;

 вивести таблицю колекціонерів;

Наступний крок етапу аналізу це опис варіантів використання, де необхідно детально описати всі варіанти використання:

* Запуск програми. При запуску програми консоль відобразить усі компоненти, які активуються одночасно з програмою. Якщо запуск програми відбувається правильно, у консолі з'явиться повідомлення про успішний запуск CourseWorkApplication за певний період часу.
* Додавання країни у базу даних. У базу даних можна додати країну за допомогою POST-запиту.
* Видалення країни з бази даних. З бази даних можна видалити країну за допомогою DELETE-запиту.
* Редагування інформації країни у базі даних. У базі даних можна редагувати інформацію про країну за допомогою PUT-запиту.
* Виведення інформації про країни у базі даних. За допомогою GET-запиту можна вивести інформацію про усіх наявні країни у базі даних.
* Додавання історії виникнення у базу даних. Додавання інформації про історію виникнення у базу даних може бути здійснене через POST-запит.
* Видалення історії виникнення з бази даних. Видалення інформації про історію виникнення з бази даних можливе за допомогою DELETE-запиту.
* Редагування інформації про історію виникнення у базі даних. Редагування інформації про історію виникнення у базі даних здійснюється за допомогою PUT-запиту.
* Виведення інформації про історію виникнення у базі даних. Виведення інформації про наявні історії виникнення монет у базі даних можливе завдяки GET-запиту.
* Додавання монети у базу даних. Додавання монети у базу даних може бути здійснене через POST-запит.
* Видалення монети з бази даних. Видалення монети з бази даних можливе за допомогою DELETE-запиту.
* Редагування інформації про монету у базі даних. Редагування інформації про монету у базі даних здійснюється за допомогою PUT-запиту.
* Виведення інформації про наявні монети у базі даних. Виведення інформації про наявні монети у базі даних можливе завдяки GET-запиту.
* Додавання інформації про колекціонера у базу даних. Додавання інформації про колекціонера у базу даних може бути здійснене через POST-запит.
* Видалення інформації про колекціонера з бази даних. Видалення інформації про колекціонера з бази даних можливе за допомогою DELETE-запиту.
* Редагування інформації про колекціонера у базі даних. Редагування інформації про колекціонера у базі даних здійснюється за допомогою PUT-запиту.
* Виведення інформації про наявних колекціонерів у базі даних. Виведення інформації про наявних колекціонерів у базі даних можливе завдяки GET-запиту.

Під час розробки REST API для автоматизації роботи з довідником нумізмата, буде розроблено ефективну структуру програмного продукту. Застосуються відповідні патерни та принципи об'єктно-орієнтованого програмування для створення чіткої архітектури.

Основні етапи проектування включають:

* розробка схеми бази даних для зберігання інформації про монети, їх країни походження та історії виникнення;
* визначення зв'язків між таблицями та їх атрибутами;
* проєктування прикладного програмного інтерфейсу (API) шляхом визначення необхідних ендпоінтів.

## Проєктування і опис прикладного інтерфейсу сервера

Проаналізувавши поставлені вимоги до REST API вебсервісу автоматизованого робочого місця адміністратора готелю, було визначено необхідність у наявності наступних ендпоінтів:

1. Ендпоінти класу History (історії виникнення):

* Метод: POST. Шлях: /histories. Призначення: Додавання інформації про історію виникнення монети.
* Метод: DELETE. Шлях: /histories/{id}. Призначення: Видалення інформації про історію виникнення монети за її унікальним ідентифікатором.
* Метод: PUT. Шлях: /histories/{id}. Призначення: Редагування даних про історію виникнення монети на основі її унікального ідентифікатора.
* Метод: GET. Шлях: /histories. Призначення: Отримання списку всіх історій виникнення та їхньої інформації.

1. Ендпоінти класу Country (країна походження):

* Метод: POST. Шлях: /countries. Призначення: Додавання країни походження монети.
* Метод: DELETE. Шлях: /countries /{id}. Призначення: Видалення інформації про країну походження за її унікальним ідентифікатором.
* Метод: PUT. Шлях: /countries /{id}. Призначення: Редагування даних про країну походження на основі її унікального ідентифікатора.
* Метод: GET. Шлях: /countries. Призначення: Отримання списку всіх країн походження та їхньої інформації.

1. Ендпоінти класу Coin (монета)

* Метод: POST. Шлях: /coins. Призначення: Додавання нової монети.
* Метод: DELETE. Шлях: /coins/{id}. Призначення: Видалення інформації про монету за її унікальним ідентифікатором.
* Метод: PUT. Шлях: /coins/{id}. Призначення: Редагування даних про монету на основі її унікального ідентифікатора.
* Метод: GET. Шлях: /coins. Призначення: Отримання списку всіх монет та їхньої інформації.

1. Ендпоінти класу Collector (колекціонерів):

* Метод: POST. Шлях: /collectors. Призначення: Додавання нового колекціонера.
* Метод: DELETE. Шлях: /collectors/{id}. Призначення: Видалення інформації про колекціонера за його унікальним ідентифікатором.
* Метод: PUT. Шлях: /collectors/{id}. Призначення: Редагування даних про колекціонера на основі його унікального ідентифікатора.
* Метод: GET. Шлях: /collectors. Призначення: Отримання списку всіх колекціонерів та їх інформації.

## Розробка структури бази даних сервера

База даних вебсервісу автоматизованого робочого місця адміністратора готелю складається з п’яти таблиць. Для полегшення створення таблиць було використано ORM фреймворк Hibernate [4]. База даних створена з використанням СКБД PostgreSQL [5].

Перелік таблиць бази даних вебсервісу наступний:

* countries – містить інформацію про країни походження;
* histories – містить інформацію про історії виникнення;
* coins – містить інформацію про монети;
* collectors – містить інформацію про колекціонерів.

У таблиці countries є стовбці з назвами:

* “id” - порядковий номер країни у таблиці бази даних;
* “name” – назва країни.

У таблиці histories є стовбці з назвами:

* “id” – порядковий номер історії у базі даних;
* “history\_body” – тіло історії виникнення монети.

У таблиці coins є стовбці з назвами:

* “id” – порядковий номер монети у базі даних;
* “country\_id” – порядковий номер країни походження монети в базі даних;
* “history\_id” – порядковий номер історії виникнення монети в базі даних.

У таблиці collectors є стовбці з назвами:

* “id” – порядковий номер колекціонера у базі даних;
* “name” – ім’я колекціонера;
* “surname” – прізвище колекціонера;
* “number\_of\_coins” – кількість монет у колекіцонера;
* “number\_of\_rare\_coins” – кількість рідкісних монет у колекціонера.

ER-діаграма бази даних вебсервісу довідника нумізмата представлено на рисунку 2.2.

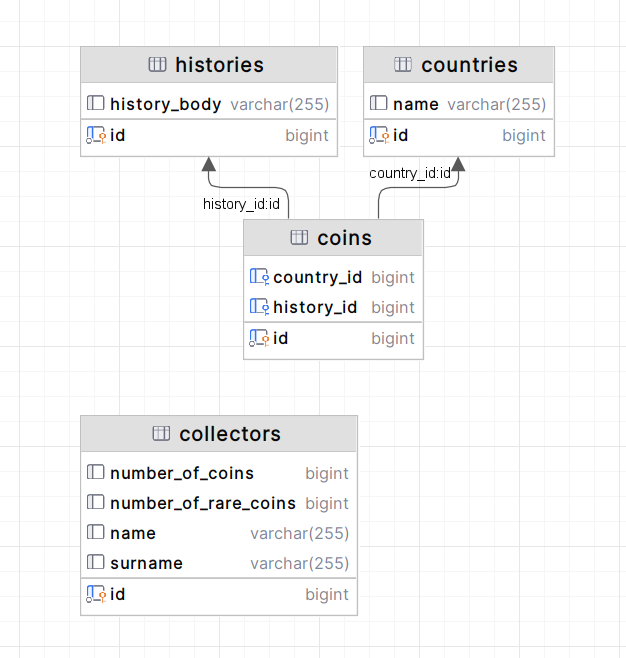


Рисунок 2.2 – ER-діаграма бази даних

## Розробка системи класів

### Доменна модель

Цей рівень містить в собі класи сутностей (Entity), об’єкти яких зберігаються у базі даних. У програмі реалізовані такі класи сутностей:

* Country – лістинг коду наведено у додатку Ґ;
* History – лістинг коду наведено у додатку Ж;
* Coin – лістинг коду наведено у додатку З;
* Collector – лістинг коду наведено у додатку И;

### Рівень доступу до даних

Рівень доступу до даних у програмі називається ключовим словом Repository. Цей рівень взаємодіє з базою даних, та її сутностями (Entity). У програмі класи рівня доступу до даних називаються так:

* CountryRepository – інтерфейс, на основі якого Spring Data [6] генерує клас для роботою з базою даних для збереження, отримання, оновлення та видалення інформації про країни походження. Лістинг даного інтерфейсу наведено у додатку Є;
* HistoryRepository – інтерфейс, на основі якого Spring Data генерує клас для роботою з базою даних для збереження, отримання, оновлення та видалення інформації про історії виникнення.
* CoinRepository – інтерфейс, на основі якого Spring Data генерує клас для роботою з базою даних для збереження, отримання, оновлення та видалення інформації про монети.
* CollectorRepository – інтерфейс, на основі якого Spring Data генерує клас для роботою з базою даних для збереження, отримання, оновлення та видалення інформації про колекціонерів.

### Сервісний рівень

Сервісний рівень використовує класи рівня доступу до даних (Repository) для своєї роботи, у своїй назві містить ключове слово Service. У програмі реалізовані такі інтерфейси сервісного рівня:

* CountryService – сервіс, який містить бізнес-логіку для операцій, пов'язаних із країнами походження. Використовує CountryRepository для доступу до даних. Лістинг даного інтерфейсу наведено у додатку Д, а лістинг класу реалізації даного інтерфейсу у додатку Е;
* HistoryService – сервіс, який надає бізнес-логіку для операцій, пов'язаних із історіями виникнення. Використовує HistoryRepository для доступу до даних.
* CoinService – сервіс, який містить бізнес-логіку для операцій, пов'язаних із монетами. Використовує CoinRepository для доступу до даних.
* CollectorService – сервіс, який містить бізнес-логіку для операцій, пов'язаних із колекціонерами. Використовує CollectorRepository для доступу до даних.

### Рівень контролерів

Рівень контролерів використовує класи сервісного рівня (Service) для виконання HTTP-запитів. Класи цього рівня у своїй назві містять ключове слово Controller. У програмі реалізовані такі класи рівня контролерів:

* CountryController – контролер, який обробляє HTTP-запити, пов'язані з країнами походження. Викликає методи CountryService для виконання запитів. Лістинг даного інтерфейсу наведено у додатку Б;
* HistoryController – контролер, який обробляє HTTP-запити, пов'язані з історіями виникнення. Викликає методи HistoryService для виконання запитів.
* CoinController – контролер, який обробляє HTTP-запити, пов'язані з монетами. Викликає методи CoinService для виконання запитів.
* CollectorController – контролер, який обробляє HTTP-запити, пов'язані з колекціонерами. Викликає методи CollectorService для виконання запитів.

Нижче представлена UML-діаграма класів (див. рис. 2.3). Клас ServiceImpl це імплемент інтерфейсу Service, ці два класи знаходяться на сервісному рівні. Класи рівню доступу до даних (Repository) теж є інтерфейсами.

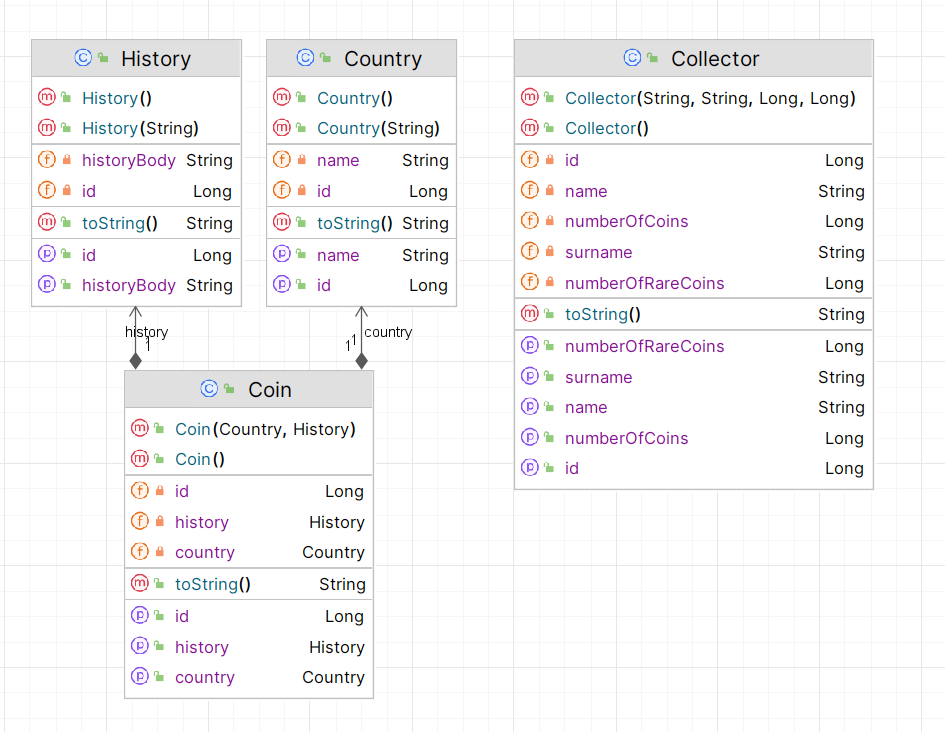


Рисунок 2.3 – UML-діаграма класів сутностей вебсервісу

## Опис форматів запитів до сервера та відповідей сервера

Формати запитів до сервера та відповідей сервера можуть варіюватися в залежності від використовуваного протоколу та стеку технологій. Однак для багатьох вебзастосунків використовуються стандартні протоколи, такі як HTTP, які визначають спосіб взаємодії між клієнтом та сервером.

Формати запитів HTTP:

* GET – клієнт відправляє запит на отримання певного ресурсу.

Приклад: GET /api/users

* POST – клієнт відправляє дані для створення нового ресурсу.

Приклад*:* POST /api/users з тілом запиту, що містить дані нового користувача.

* PUT *–* клієнт відправляє дані для оновлення існуючого ресурсу.

Приклад: PUT /api/users/123 з тілом запиту, що містить нові дані користувача з ідентифікатором 123.

* DELETE –клієнт відправляє запит на видалення ресурсу.

Приклад: DELETE /api/users/123.

Також на кожен запит сервер надає відповідь, що записується кодом з 3 цифр[7]. Формати відповідей HTTP використовуються наступні:

* Успішна відповідь (200 OK).

Приклад: 200 OK з тілом відповіді, яке містить дані запиту.

* Створено (201 Created).

Приклад: 201 Created з тілом відповіді, яке містить дані новоствореного ресурсу.

* Помилка в запиті (400 Bad Request).

Приклад: 400 Bad Request з тілом відповіді, яке містить інформацію про помилку в запиті.

* Не знайдено (404 Not Found).

Приклад: 404 Not Found з тілом відповіді, яке вказує на те, що ресурс не знайдено.

* Помилка сервера (500 Internal Server Error).

Приклад: 500 Internal Server Error з тілом відповіді, яке містить інформацію про внутрішню помилку сервера.

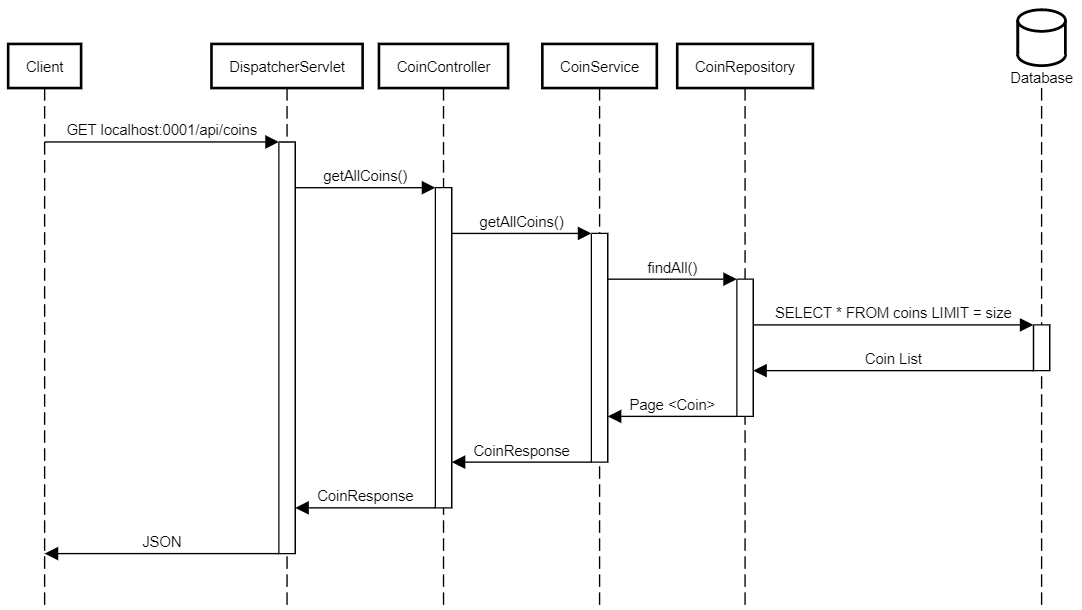


Рисунок 2.4 – Діаграма послідовності взаємодії із вебсервісом

## Опис файлової структури ПЗ

Файли проєкту вебсервісу розділені на пакети згідно принципів розробки тришарової архітектури REST вебсервісу та стандартної структури Maven-проекту:

* controller – містить файли класів рівня контролерів, для прикладу у лістингу Б наведено лістинг файлу класу CountryController;
* dto – містить класи, якими оперують класи рівня сервісів. Файл CountryDTO наведений у додатку Г;
* model – містить класи рівня сутностей, файл Country наведений у додатку Ґ, файл History – у додатку Ж, файл Coin представлений у додатку З та файл Collector знаходиться у додатку И;
* repository – містить класи рівня доступу до даних, файл CountryRepository наведений у додатку Є;
* service – містить класи сервісного рівня, файл інтерфейсу CountryService представлений у додатку Д та його імплементація у додатку Е;
* utils – містить константи що використовує ПЗ, такий файл наведений у додатку Ї.

Весь вихідний код розташований на веб-сервісі Github в репозиторію [8]

# Тестування програми і результати її виконання

Для тестування було використано програму Postman – клієнт для автоматизації тестування API, який починав свій шлях як плагін для браузера Chrome, але пізніше отримав і десктопні версії для Windows, Linux і Mac. Але автоматизація – не основний напрямок для використання цього додатка. Він підійде і для проведення дослідницького (exploratory) тестування. Також Postman – ефективний інструмент для розробників з метою створення та перевірки API. [9]

Спочатку було протестовано коректну роботу ендпоїнтів програми для класу History. Першим ендпоінтом було протестовано методом POST (див. рис. 3.1), котрий працює коректно. Далі було протестовано ендпоінт методом GET (див. рис. 3.2) і потім PUT (див. рис. 3.3) та методом DELETE (див. рис. 3.4).

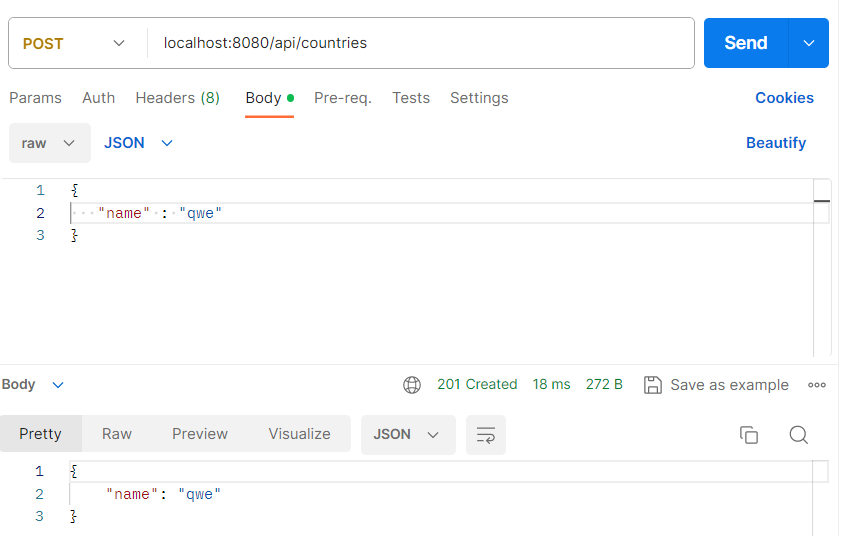


Рисунок 3.1 – Результат виконання запиту для створення нової країни походження

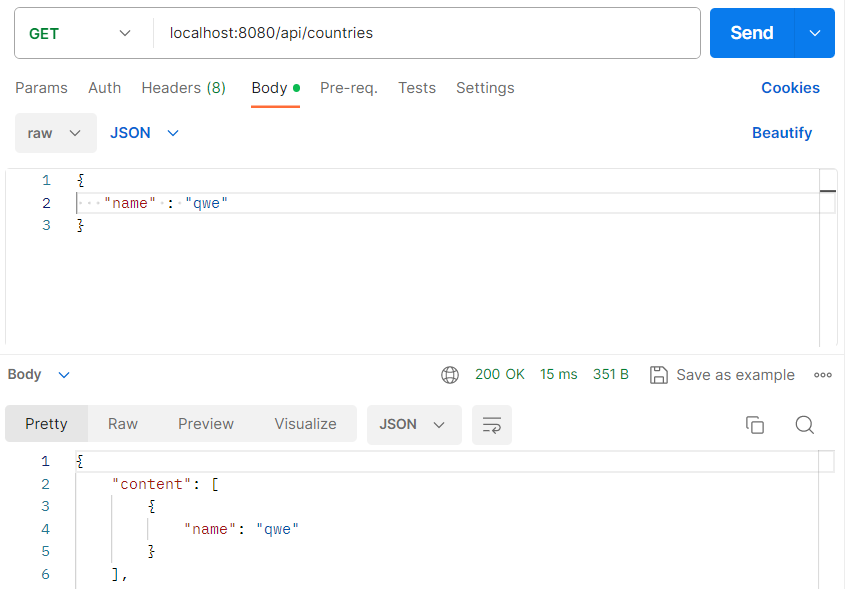


Рисунок 3.2 – Результат виконання запиту для виведення усіх країн походження

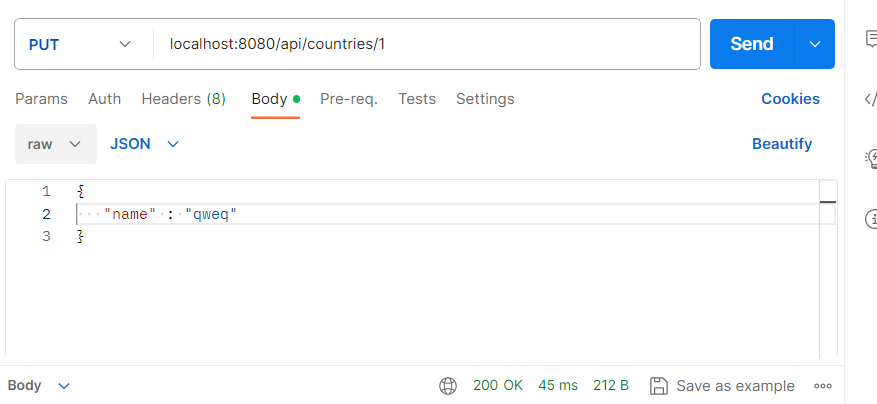


Рисунок 3.3 – Результат виконання запиту для оновлення інформації про країни

Також було протестовано ендпоінти класу Coin, так як у нього є пов’язане поле з класом Country. Ендпоінт POST працює коректно (див. рис. 3.5), запит GET теж (див. рис. 3.6), методи PUT (див. рис. 3.7) та DELETE (див. рис. 3.8) теж не видали помилок.

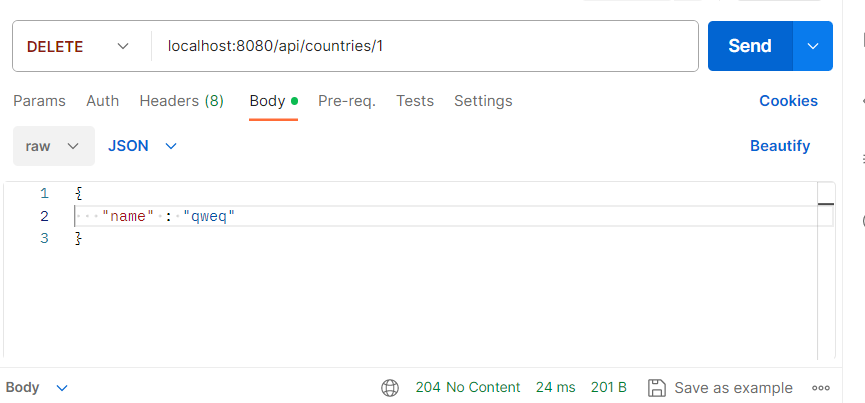


Рисунок 3.4 – Результат виконання запиту для видалення країни

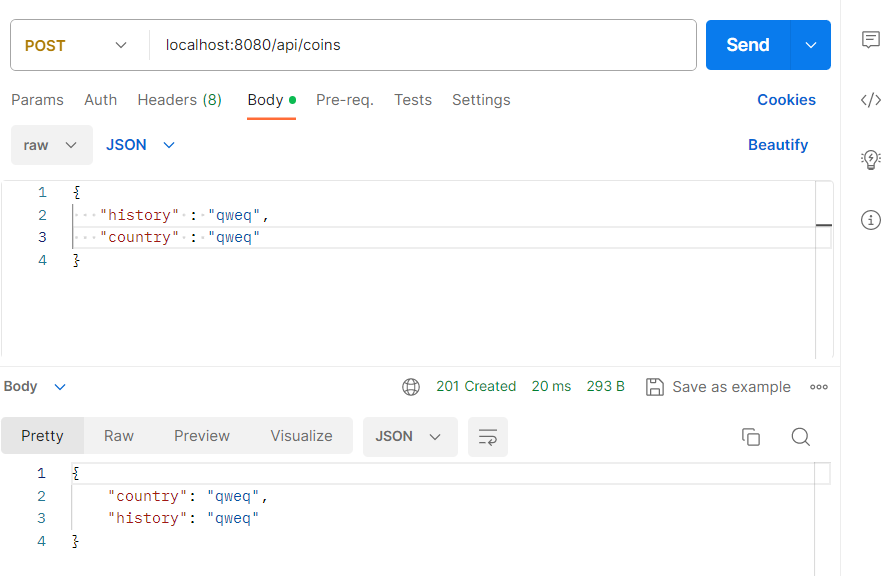


Рисунок 3.5 – Результат виконання запиту для створення нової монети

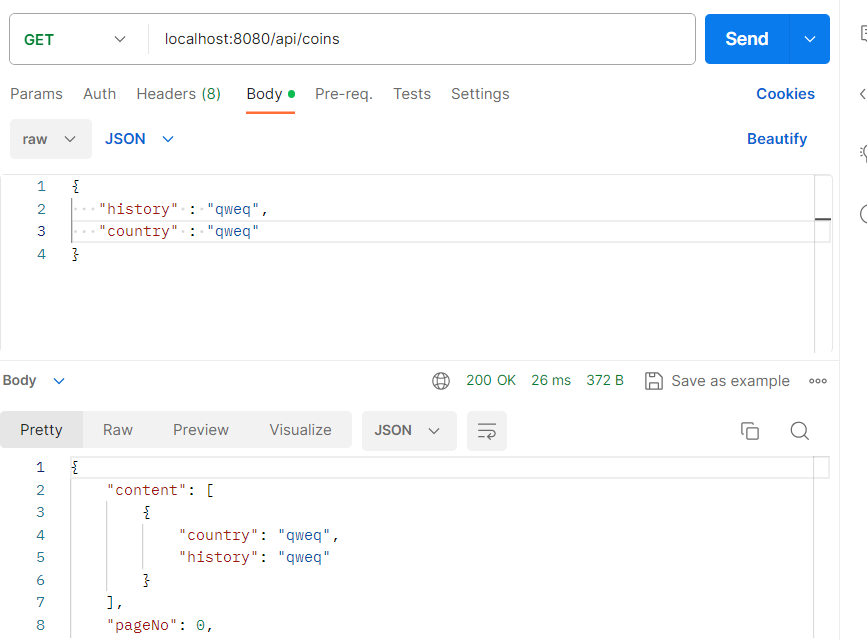
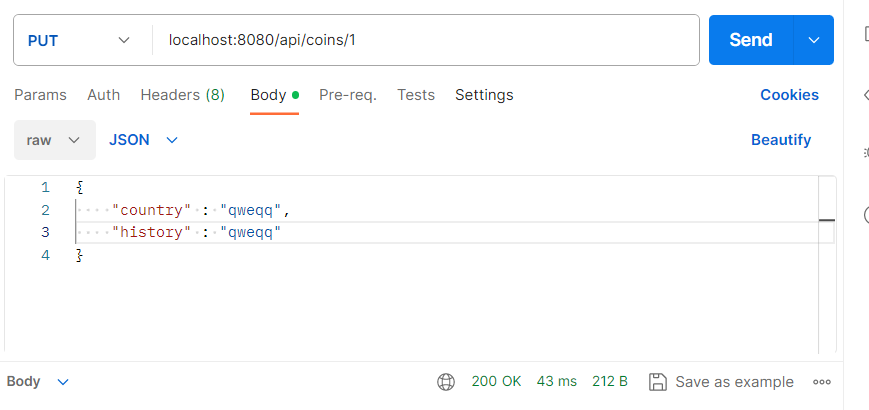


Рисунок 3.6 – Результат виконання запиту для виведення усіх монет

Рисунок 3.7 – Результат виконання запиту для оновлення інформації монети

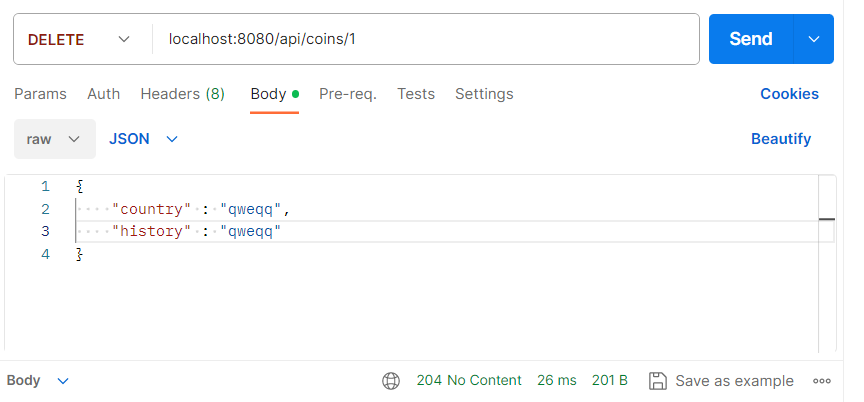


Рисунок 3.8 – Результат виконання запиту для видалення монети

Також було проведено юніт-тестування для класу сутностей Coin, для перевірки роботи методів даного класу (див. рис. 3.9).



Рисунок 3.9 – Результат тестування методів класу Coin

Висновки

У ході курсової роботи була розроблена система, яка забезпечує автоматизацію довідника нумізмата. Система базується на технологіях Hibernate для роботи з базою даних, фреймворк Spring Boot для реалізації REST API та бізнес-логіки та Maven для підключення залежностей програми. [10]

Система дозволяє додавати, видаляти, редагувати та виводити інформацію про монети, їх країни походження та історії виникнення, про колекціонерів. Кожен з цих елементів має свої відповідні ендпоінти, що дозволяє ефективно керувати базою даних.

В проекті були вдало використані технології Hibernate для роботи з об'єктно-реляційною базою даних, Spring Boot для реалізації логіки та створення REST API та кешування.

Процес розробки включав в себе етапи аналізу вимог, проектування системи, реалізацію та тестування. Зустрічалися виклики, пов'язані з вибором оптимальних архітектурних рішень та взаємодією між компонентами.

Розроблена система дозволяє покращити ефективність роботи довідника нумізмата, спрощуючи процеси ведення обліку, реєстрації колекціонерів та монет. Перспективою розвитку може бути додавання нового функціоналу, оптимізація продуктивності та розширення можливостей взаємодії із зовнішніми системами.

Перелік посилань

1. Слободян Р.О. Методичні вказівки до виконання курсової роботи з дисципліни «Розробка клієнт-серверних застосувань» для студентів спеціальності 122 «Комп’ютерні науки». Тернопіль : ВСП «ТФК ТНТУ», 2021. 45с.
2. What is a REST API\_ \_ IBM : вебсайт. URL: https://www.ibm.com/topics/rest-apis (дата звернення 13.12.2023)
3. Getting started \_ IntelliJ IDEA Documentation : вебсайт, URL: https://www.jetbrains.com/help/idea/ (дата звернення 14.12.2023)
4. Documentation - 6.4 - Hibernate ORM : вебсайт. URL: https://hibernate.org/orm/documentation/6.4/ (дата звернення 15.12.2023)
5. PostgreSQL\_ Documentation : вебсайт. URL: https://www.postgresql.org/docs/ (дата звернення 15.12.2023)
6. Spring Framework Documentation \_\_ Spring Framework : вебсайт. URL: https://docs.spring.io/spring-framework/reference (дата звернення: 15.12.2023).
7. HTTP response status codes - HTTP \_ MDN : вебсайт. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status (дата звернення 16.12.2023)
8. GitHub - numismatbackend : вебсайт. URL: https://github.com/verezhakolexandr/numismatbackend
9. Огляд популярних інструментів для тестування API \_ Онлайн-курси від компанії QATestLab : вебсайт. URL: https://training.qatestlab.com/blog/technical-articles/api-testing-tools
10. Maven – Maven Documentation : вебсайт. URL: https://maven.apache.org/guides/index.html (дата звернення 16.12.2023)

Додаток А  
Лістинг файлу «pom.xml»

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.2.0</version>  
 <relativePath/> *<!-- lookup parent from repository -->* </parent>  
 <groupId>com.example</groupId>  
 <artifactId>CourseWork</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>CourseWork</name>  
 <description>Demo project for Spring Boot</description>  
 <properties>  
 <java.version>21</java.version>  
 <org.mapstruct.version>1.5.5.Final</org.mapstruct.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.postgresql</groupId>  
 <artifactId>postgresql</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.mapstruct</groupId>  
 <artifactId>mapstruct</artifactId>  
 <version>1.6.0.Beta1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.modelmapper</groupId>  
 <artifactId>modelmapper</artifactId>  
 <version>2.3.2</version>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 </dependency>  
  
 </dependencies>  
  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.11.0</version>  
 <configuration>  
 <annotationProcessorPaths>  
 <path>  
 <groupId>org.mapstruct</groupId>  
 <artifactId>mapstruct-processor</artifactId>  
 <version>1.6.0.Beta1</version>  
 </path>  
 </annotationProcessorPaths>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

Додаток Б  
Лістинг файлу «countrycontroller.java»

package com.example.CourseWork.controller;  
  
import com.example.CourseWork.dto.CountryDTO;  
import com.example.CourseWork.dto.responses.CountryResponse;  
import com.example.CourseWork.service.CountryService;  
import com.example.CourseWork.utils.ApplicationConstants;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.Optional;  
  
@CrossOrigin(origins = "http://localhost:0001")  
@RestController  
@RequestMapping("/api/countries")  
public class CountryController {  
 private CountryService countryService;  
   
 public CountryController(CountryService countryService) {  
 this.countryService = countryService;  
 }  
 @GetMapping("")  
 public CountryResponse getAllCountries(  
 @RequestParam(value = "pageNo", defaultValue = ApplicationConstants.*DEFAULT\_PAGE\_NUMBER*, required = false) int pageNo,  
 @RequestParam(value = "pageSize", defaultValue = ApplicationConstants.*DEFAULT\_PAGE\_SIZE*, required = false) int pageSize,  
 @RequestParam(value = "sortBy", defaultValue = ApplicationConstants.*DEFAULT\_SORT\_BY*, required = false) String sortBy,  
 @RequestParam(value = "sortDir", defaultValue = ApplicationConstants.*DEFAULT\_SORT\_DIRECTION*, required = false) String sortDir)  
 {  
 return countryService.getAllCountries(pageNo, pageSize, sortBy, sortDir);  
 }  
  
 @GetMapping("/{id}")  
 public ResponseEntity<CountryDTO> getCountryById(@PathVariable("id") Long id) {  
 Optional<CountryDTO> countryDTO = Optional.*ofNullable*(countryService.findCountryById(id));  
 if (countryDTO.isPresent()) {  
 return new ResponseEntity<>(countryDTO.get(), HttpStatus.*OK*);  
 } else {  
 return new ResponseEntity<>(HttpStatus.*NOT\_FOUND*);  
 }  
 }  
  
 @PostMapping("")  
 public ResponseEntity<CountryDTO> createCountry(@RequestBody CountryDTO countryDTO) {  
 return new ResponseEntity<>(countryService.createCountry(countryDTO), HttpStatus.*CREATED*);  
 }  
  
  
 @PutMapping("/{id}")  
 public ResponseEntity<CountryDTO> updateCountry(@PathVariable("id") Long id, @RequestBody CountryDTO countryDTO) {  
 Optional<CountryDTO> countryData = Optional.*ofNullable*(countryService.findCountryById(id));  
 if (countryData.isPresent()) {  
 countryService.updateCountryInfo(countryDTO, id);  
 return new ResponseEntity<>(HttpStatus.*OK*);  
 } else {  
 return new ResponseEntity<>(HttpStatus.*NOT\_FOUND*);  
 }  
 }  
  
  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity<HttpStatus> deleteCountry(@PathVariable("id") Long id) {  
 try {  
 countryService.deleteCountry(id);  
 return new ResponseEntity<>(HttpStatus.*NO\_CONTENT*);  
 } catch (Exception e) {  
 return new ResponseEntity<>(HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
}

Додаток В  
Лістинг файлу «countryresponse.java»

package com.example.CourseWork.dto.responses;  
  
import com.example.CourseWork.dto.CoinDTO;  
import com.example.CourseWork.dto.CountryDTO;  
  
import java.util.List;  
  
public class CountryResponse {  
 private List<CountryDTO> content;  
 private int pageNo;  
 private int pageSize;  
 private long totalElements;  
 private int totalPages;  
 private boolean last;  
  
 public CountryResponse() {  
 }  
  
 public CountryResponse(List<CountryDTO> content, int pageNo, int pageSize, long totalElements, int totalPages, boolean last) {  
 this.content = content;  
 this.pageNo = pageNo;  
 this.pageSize = pageSize;  
 this.totalElements = totalElements;  
 this.totalPages = totalPages;  
 this.last = last;  
 }  
  
 public List<CountryDTO> getContent() {  
 return content;  
 }  
  
 public void setContent(List<CountryDTO> content) {  
 this.content = content;  
 }  
  
 public int getPageNo() {  
 return pageNo;  
 }  
  
 public void setPageNo(int pageNo) {  
 this.pageNo = pageNo;  
 }  
  
 public int getPageSize() {  
 return pageSize;  
 }  
  
 public void setPageSize(int pageSize) {  
 this.pageSize = pageSize;  
 }  
  
 public long getTotalElements() {  
 return totalElements;  
 }  
  
 public void setTotalElements(long totalElements) {  
 this.totalElements = totalElements;  
 }  
  
 public int getTotalPages() {  
 return totalPages;  
 }  
  
 public void setTotalPages(int totalPages) {  
 this.totalPages = totalPages;  
 }  
  
 public boolean isLast() {  
 return last;  
 }  
  
 public void setLast(boolean last) {  
 this.last = last;  
 }  
  
 @Override  
 public String toString() {  
 return "CountryResponse{" +  
 "content=" + content +  
 ", pageNo=" + pageNo +  
 ", pageSize=" + pageSize +  
 ", totalElements=" + totalElements +  
 ", totalPages=" + totalPages +  
 ", last=" + last +  
 '}';  
 }  
}

Додаток Г  
Лістинг файлу «Countrydto.java»

package com.example.CourseWork.dto;  
  
public record CountryDTO(String name) {  
 public CountryDTO {  
 }  
}

Додаток Ґ  
Лістинг файлу «COUNTRY.java»

package com.example.CourseWork.model;  
  
import jakarta.persistence.\*;  
  
  
@Entity  
@Table(name = "countries")  
public class Country {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "id")  
 private Long id;  
  
 @Column(name = "name")  
 private String name;  
  
  
 public Country() {  
 }  
  
 public Country(String name) {  
 this.name = name;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String toString() {  
 return "Country{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 '}';  
 }  
}

Додаток Д  
Лістинг файлу «countryservice.java»

package com.example.CourseWork.service;  
  
import com.example.CourseWork.dto.CountryDTO;  
import com.example.CourseWork.dto.HistoryDTO;  
import com.example.CourseWork.dto.responses.CountryResponse;  
import com.example.CourseWork.dto.responses.HistoryResponse;  
import org.springframework.stereotype.Service;  
  
@Service  
public interface CountryService {  
 CountryDTO createCountry(CountryDTO countryDTO);  
  
 CountryDTO findCountryById(Long id);  
  
 CountryDTO updateCountryInfo(CountryDTO countryDTO, Long id);  
  
 CountryDTO deleteCountry(Long id);  
  
 CountryResponse getAllCountries(int pageNo, int pageSize, String sortBy, String sortDir);  
}

Додаток Е  
Лістинг файлу «countryserviceimpl.java»

package com.example.CourseWork.service.impl;  
  
import com.example.CourseWork.dto.CountryDTO;  
import com.example.CourseWork.dto.responses.CountryResponse;  
import com.example.CourseWork.model.Country;  
import com.example.CourseWork.repository.CountryRepository;  
import com.example.CourseWork.service.CountryService;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.PageRequest;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
import java.util.stream.Collectors;  
  
@Service  
public class CountryServiceImpl implements CountryService {  
 private CountryRepository countryRepository;  
  
 private CountryDTO toCountryDTO(Country country){  
 CountryDTO countryDTO;  
 countryDTO = new CountryDTO(country.getName());  
 return countryDTO;  
 }  
  
 public CountryServiceImpl(CountryRepository countryRepository) {  
 this.countryRepository = countryRepository;  
 }  
 @Override  
 public CountryDTO createCountry(CountryDTO countryDTO) {  
 Country country = new Country();  
 country.setName(countryDTO.name());  
 return toCountryDTO(countryRepository.save(country));  
 }  
  
 @Override  
 public CountryDTO findCountryById(Long id) {  
 Optional<Country> country = countryRepository.findById(id);  
 if(country.isPresent()) {  
 return toCountryDTO(country.get());  
 }  
 return null;  
 }  
  
 @Override  
 public CountryDTO updateCountryInfo(CountryDTO countryDTO, Long id) {  
 Optional<Country> country = countryRepository.findById(id);  
 if (country.isPresent()) {  
 Country countryModel = country.get();  
 countryModel.setName(countryDTO.name());  
 return toCountryDTO(countryRepository.save(countryModel));  
 }  
 return null;  
 }  
  
 @Override  
 public CountryDTO deleteCountry(Long id) {  
 Optional<Country> countryData = countryRepository.findById(id);  
 if (countryData.isPresent()) {  
 countryRepository.delete(countryData.get());  
 }  
 return null;  
 }  
  
 @Override  
 public CountryResponse getAllCountries(int pageNo, int pageSize, String sortBy, String sortDir) {  
 Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortBy).ascending()  
 : Sort.by(sortBy).descending();  
  
 Pageable pageable = PageRequest.of(pageNo, pageSize, sort);  
  
 Page<Country> countries = countryRepository.findAll(pageable);  
 List<Country> listOfCountries = countries.getContent();  
  
 List<CountryDTO> content = listOfCountries.stream().map(country -> toCountryDTO(country)).collect(Collectors.toList());  
 CountryResponse countryResponse = new CountryResponse();  
 countryResponse.setContent(content);  
 countryResponse.setPageNo(countries.getNumber());  
 countryResponse.setPageSize(countries.getSize());  
 countryResponse.setTotalElements(countries.getTotalElements());  
 countryResponse.setTotalPages(countries.getTotalPages());  
 countryResponse.setLast(countries.isLast());  
  
 return countryResponse;  
 }  
}

Додаток Є  
Лістинг файлу «countryrepository.java»

package com.example.CourseWork.repository;  
  
import com.example.CourseWork.model.Country;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface CountryRepository extends JpaRepository<Country, Long> {  
 Country findByName(String name);  
}

Додаток Ж  
Лістинг файлу «HISTORY.java»

package com.example.CourseWork.model;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "histories")  
public class History {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "id")  
 private Long id;  
 @Column(name = "history\_body")  
 private String historyBody;  
  
 public History() {  
 }  
  
 public History(String historyBody) {  
 this.historyBody = historyBody;  
 }  
  
 public void setHistoryBody(String historyBody) {  
 this.historyBody = historyBody;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public String getHistoryBody() {  
 return historyBody;  
 }  
  
 @Override  
 public String toString() {  
 return "History{" +  
 "id=" + id +  
 ", historyBody='" + historyBody + '\'' +  
 '}';  
 }  
}

Додаток З  
Лістинг файлу «Coin.java»

package com.example.CourseWork.model;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "coins")  
public class Coin {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "id")  
 private Long id;  
 @OneToOne(cascade = CascadeType.*ALL*)  
 @JoinColumn(name = "country\_id", referencedColumnName = "id")  
 private Country country;  
 @OneToOne(cascade = CascadeType.*ALL*)  
 @JoinColumn(name = "history\_id", referencedColumnName = "id")  
 private History history;  
  
 public Coin() {  
 }  
  
 public Coin(Country country, History history) {  
 this.country = country;  
 this.history = history;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public Country getCountry() {  
 return country;  
 }  
  
 public History getHistory() {  
 return history;  
 }  
  
 public void setCountry(Country country) {  
 this.country = country;  
 }  
  
 public void setHistory(History history) {  
 this.history = history;  
 }  
  
 @Override  
 public String toString() {  
 return "Coin{" +  
 "id=" + id +  
 ", country=" + country +  
 ", history=" + history +  
 '}';  
 }  
}

Додаток И  
Лістинг файлу «collector.java»

package com.example.CourseWork.model;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "collectors")  
public class Collector {  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name = "id")  
 private Long id;  
 @Column(name = "name")  
 private String name;  
 @Column(name = "surname")  
 private String surname;  
 @Column(name = "number\_of\_coins")  
 private Long numberOfCoins;  
 @Column(name = "number\_of\_rare\_coins")  
 private Long numberOfRareCoins;  
  
 public Collector() {  
 }  
  
 public Collector(String name, String surname, Long numberOfCoins, Long numberOfRareCoins) {  
 this.name = name;  
 this.surname = surname;  
 this.numberOfCoins = numberOfCoins;  
 this.numberOfRareCoins = numberOfRareCoins;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public void setNumberOfCoins(Long numberOfCoins) {  
 this.numberOfCoins = numberOfCoins;  
 }  
  
 public void setNumberOfRareCoins(Long numberOfRareCoins) {  
 this.numberOfRareCoins = numberOfRareCoins;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public Long getNumberOfCoins() {  
 return numberOfCoins;  
 }  
  
 public Long getNumberOfRareCoins() {  
 return numberOfRareCoins;  
 }  
  
 @Override  
 public String toString() {  
 return "Collector{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", surname='" + surname + '\'' +  
 ", numberOfCoins=" + numberOfCoins +  
 ", numberOfRareCoins=" + numberOfRareCoins +  
 '}';  
 }  
}

Додаток І  
Лістинг файлу «application.properties»

spring.datasource.url= jdbc:postgresql://localhost:5432/courseworkdb  
spring.datasource.username= postgres  
spring.datasource.password= qwe  
  
spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation= true  
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.PostgreSQLDialect  
  
*# Hibernate ddl auto (create, create-drop, validate, update)*spring.jpa.hibernate.ddl-auto= create

Додаток Ї  
Лістинг файлу «applicationconstants.java»

package com.example.CourseWork.utils;  
  
public class ApplicationConstants {  
 public static final String *DEFAULT\_PAGE\_NUMBER* = "0";  
 public static final String *DEFAULT\_PAGE\_SIZE* = "10";  
 public static final String *DEFAULT\_SORT\_BY* = "id";  
 public static final String *DEFAULT\_SORT\_DIRECTION* = "asc";  
  
}