МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
СУМСКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

На тему:

«Інформаційна система-тренажер для виконання практичних робіт з дисципліни "Сучасні фреймоврки проектування. »

Завідувач випускаючої кафедри: Довбиш А.С.

Керівник роботи Берест О.Б.

Студент гр. Індн-71с Кругових М.Ю.

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Центр заочної, дистанційної і вечірньої форм навчання

Кафедра комп’ютерних наук

Затверджую___________
Зав. кафедрою Довбиш А.С.
“_____” ____________2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи . Індн-71с спеціальності “Комп’ютерних наук” дистанційної форми навчання Кругових Максима Юрійовича.

Тема: « Інформаційна система-тренажер для виконання практичних робіт з дисципліни “Сучасні фреймоврки проектування.»

Затверджена наказом по СумДУ
№________ від ___________2021 р.

Зміст пояснювальної записки: 1) аналітичний огляд методів розпізнавання образів; 2) постановка завдання й формування завдань дослідження; 3) опис основних положень, математичних моделей і критеріїв, що використовуються інформаційно-екстремальною інтелектуальною технологією; 5) розробка інформаційного й програмного забезпечення інтелектуальної системи; 6) аналіз результатів моделювання.

Дата видачі завдання “_____”____________2021 р.
Керівник випускної роботи ___________ Берест О.Б.
Завдання приняв до виконання _____________ Кругових М.Ю.

РЕФЕРАТ

Записка: 91 стор., 14рис., 1додаток, 11 джерел

Об’єкт дослідження - навчальні платформи та тренажери для Spring framework

Мета роботи — розробка інформаційної системи – тренажера для вивчення Spring framework, Використовуючи актуальні технології розробки.

Методи дослідження — метод розробки веб-додатку з використанням веб-фреймворку.

Результати — розроблено систему-тренажер для навчального предмету. Написаний сценарій для проходження цього тренажеру. Тренажер написаний на мові програмування Java з використанням Spring framework.

SQL, НАВЧАЛЬНА ПЛАТФОРМА, ВЕБ-РОЗРОБКА, ТРЕНАЖЕР, SPRING, JAVA, MARIADB, SWAGGER, POSTMAN, AOP, SECURITY

Зміст

Вступ 5
1 5
1.1 6
1.2 6
1.3 10
2 11
2.1 11
2.2 13
2.3 14
3 16
3.1 16
Завдання 0 16
Завдання 1 18
Завдання 2 20
Завдання 3 23
Завдання 4 25
Завдання 5 29
Завдання 6 33
Завдання 7 39
Завдання 8 42
3.2 49
3.3 50
3.4 50
3.6. Короткий опис програмної реалізації 52
3.7. Аналіз роботи програми 53
Висновки 55
Список літератури 56
Додаток А 57

Вступ

Останніми роками технологія стає все більш і більш невід’ємною частиною нашого повсякденного життя. Ми живемо у суспільстві, де ідеї є рушійною силою, а не основними товарами.

Існує дуже мало способів, якими технології не впливають на наше життя. Ми майже завжди використовуємо якусь технологію, і наша залежність від технології буде зростати лише з часом. За даними Міністерства праці України, зростання зайнятості протягом наступних кількох років буде зумовлений нашою зростаючою залежністю від технологій та постійною важливістю підтримки системної та мережевої безпеки. Через це роботодавці потребують добре освічених та висококваліфікованих людей, які розуміють найновіші технологічні розробки для заміщення високотехнологічних посад.

Виробництво технологій є важливою частиною світу, що розвивається. Це означає, що комп’ютерне програмування є надзвичайно важливим для нашого майбутнього як глобального суспільства. Випускники комп'ютерного програмування можуть допомогти створити це майбутнє шляхом автоматизації процесів, збору даних, аналізу інформації та обміну знаннями для постійних інновацій та вдосконалення існуючих процесів.

Це означає, що, хоча комп’ютерне програмування сьогодні надзвичайно важливо, воно може мати ще більший ефект у майбутньому. Поки комп’ютерні програмісти у всьому світі працюють над вивченням нових способів спілкування з машинами та комп’ютерами, галузь буде продовжувати рости. Зараз здобуття ступеня комп’ютерного програмування означає, що ви можете бути частиною цього дослідження та тестування для розробки функцій, які можуть допомогти суспільству.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Основна термінологія

Мови комп’ютерного програмування дозволяють давати вказівки комп’ютеру мовою, яку він розуміє. Подібно до того, як існує багато мов, заснованих на людині, існує безліч мов комп'ютерного програмування, які програмісти можуть використовувати для спілкування з комп'ютером. Частина мови, яку може зрозуміти комп’ютер, називається двійковою. Переклад мови програмування у бінарний файл відомий як компіляція. Кожна мова, від мови C до Python, має свої особливості, хоча є багато спільних рис між мовами програмування .

Ці мови дозволяють комп’ютерам швидко та ефективно обробляти великі та складні масиви інформації. Наприклад, якщо людині надається список рандомізованих чисел від однієї до десяти тисяч і його просять розмістити у порядку зростання, швидше за все, це займе значну кількість часу та включатиме деякі помилки.Комп’ютер в даному випадку зробить все швидко та правильно.

Система-тренажер це така система яка допомогає вивчити або закріпити навички набуті в процесі навчання.

1.2 Огляд відомих рішень

В сучасному світі можно знайти дуже багато систем-тренажерів, для вивчення необхідного навику.Тренажери для вивчення мов програмування не є винятком.

Давайте розглянемо деякі приклади таких тренажерів:

а) **Udemy**

Одним з найбільш популярних представників є Веб сайт https://www.udemy.com/ [1]

Цей веб ресурс надає можливість для самостійного навчання, має дуже багато курсів по різним напрямкам.

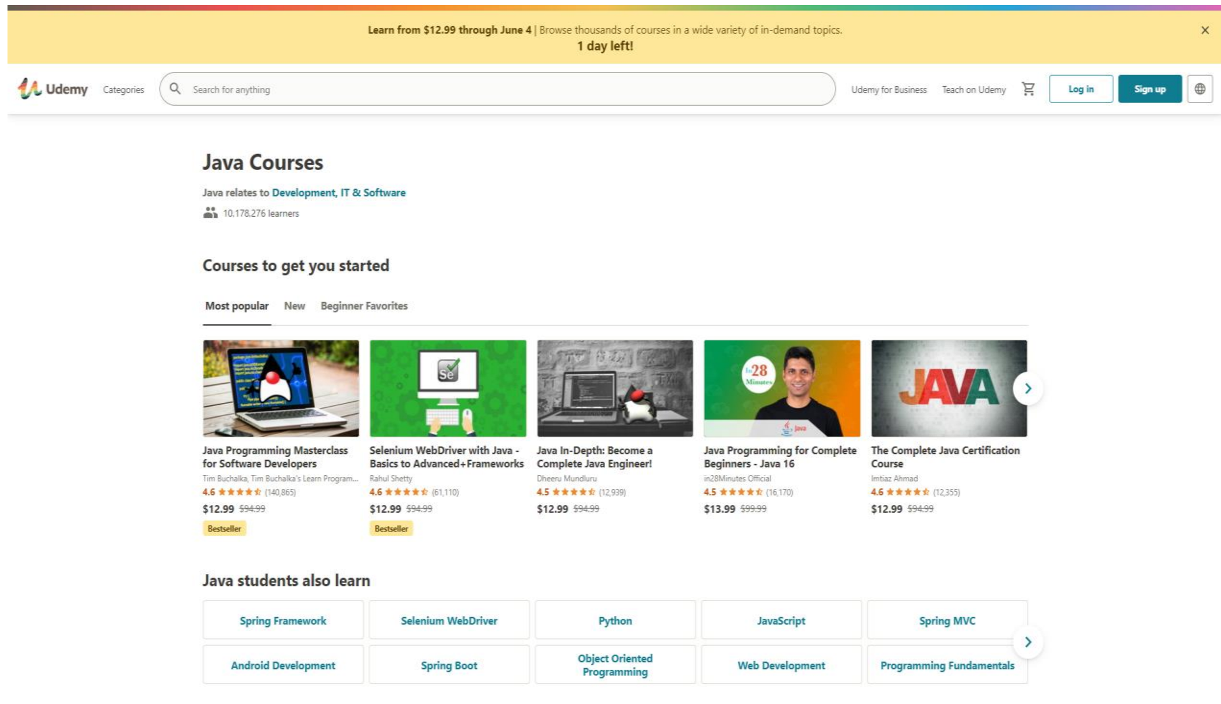

Рисунок 1.1 Скрін шот Udemy

До переваг данного ресурсу можно віднести:

- Дуже велиа курсотека
- Зручний UI
- Наявність сертефікату після закінчення курсу.

До недоліків можно віднести:

- Сервіс платний.
- Дуже багато неякісного контенту.
- Більша частина контенту англійською.

b) JavaRush

Наступний сервіс в нашому огляді буде https://javarush.ru/ [2]

Ця система-тренажер націлена на початківців які тільки почали свій шлях в вивченні мови програмування.

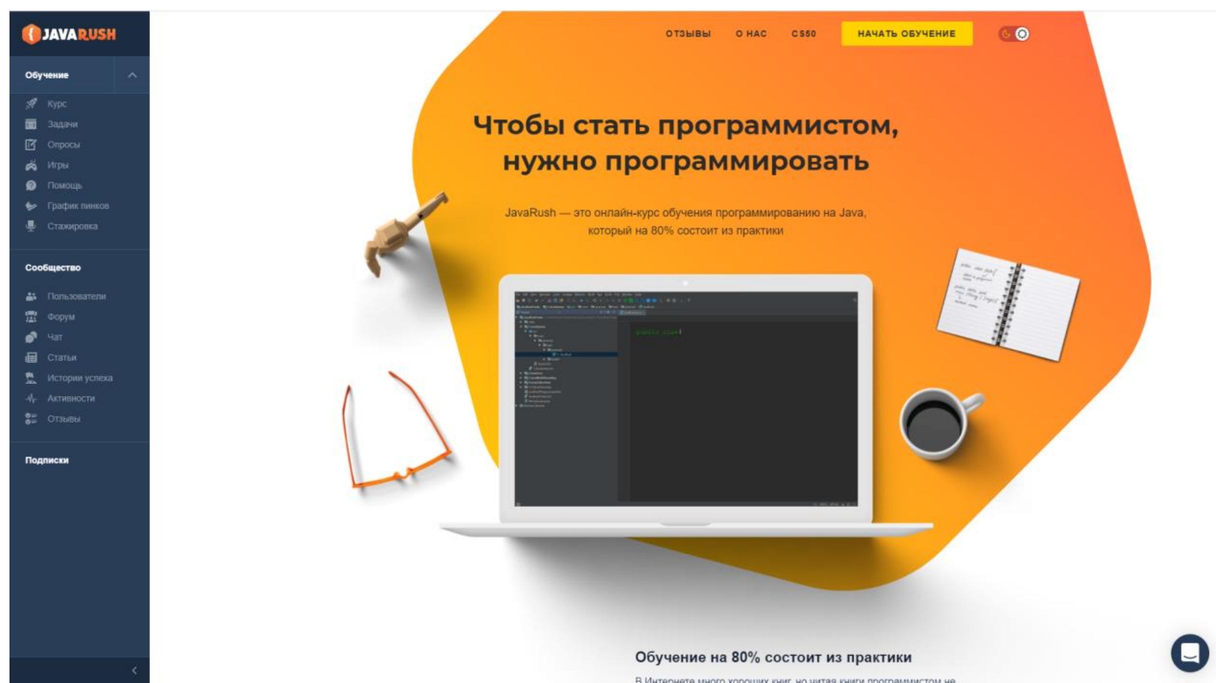

Рисунок 1.2 Скріншот JavaRush

До переваг данного ресурсу можно віднести:

- Наростаюча складність матеріалу
- Зручний UI
- Наявність великої кількості алгоритмічних задач.

До недоліків можно віднести:

- Сервіс платний.
- Немає необхідної частини про Web development.
- Немає матеріалу про Spring framework.

c) Coursera

Останній на огляді сервіс https://ru.coursera.org/ [3]

Цей ресурс надає можливість для самостійного навчання за безліччю спеціальностями.

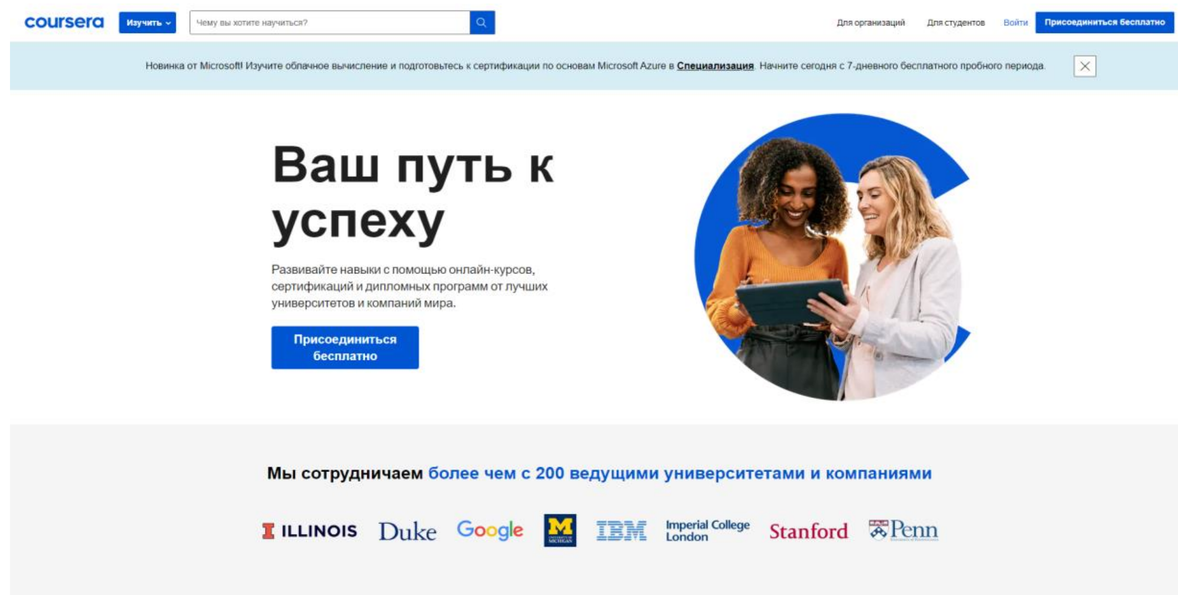

Рисунок 1.3 Скріншот Cousera

До переваг данного ресурсу можно віднести:

- Дуже велиа курсотека
- Зручний UI
- Матеріал безкоштовний.
- Якісний контент

До недоліків можно віднести:

- Сервіс платний (якщо плануєте отримати сертифікат).
- Більша частина контенту англійською.
- Необхідні курси розраховані на слухачів з певним рівнем знання в сфері.
- Не детальний розбір особливостей роботи Фреймворків.

1.3 Постановка задачі

Зробивши аналіз , стало зрозуміло що на ринку немає необхідного інструменту для самонавчання для студентів. Тому за прохання викладача Берест Олега Борисовича, була почата робота над створення власного тренажера який би відповідав всім необхідним крітеріям.

У данній роботі потрібно створити інформаційну систему-тренажер для виконання практичних робіт з дисципліни "Сучасні фреймоврки проектування. Spring"

Вимоги до продутку:

- Актуальність технологій
- Обширна кількість тем для вивчення фреймворку.
- Можливість кастомізації
- Приклади коду
- Відповідність code style guide
- Мова програмування Java

Метою данної роботи є саме розробленя серверної части а не дизайну.Продукт повинен відповідати всім вимогам сучасного IT-ринку, що після проходження курсу студент мав можилвість пройти співбесіду за данним направленням.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Вибір апаратно-програмного інструментарію

Як мову для розробки серверної частини додатку було обрано Java. Причиною слуговало те що, цей додаток розробляется для дисциліни по вивчення фреймворку мови програмування Java.

Java – відносна проста там популярно мультипрлатформна мова програмування. Ця мова дуже добре підходить для серверів які маю дуже велике щоденне навантаження за її швидкодію та відказостійкість. Також викроситовування Java доцільно для security проктів. Вона позволяє розробити захищене, легко маштабуємий серверну прикладну програму

Згідно технічном завданню було вибрано фреймворк Spring. Spring Framework забезпечує комплексну модель програмування та конфігурації для сучасних корпоративних програм на базі Java - на будь-якій платформі розгортання.

Ключовим елементом Spring є інфраструктурна підтримка на рівні додатків: Spring фокусується на «трубопроводі» корпоративних додатків, щоб команди могли зосередитися на бізнес-логіці на рівні додатків, без зайвих прив’язок до конкретних середовищ розгортання.

Переваги Spring framework:

− Структура Spring стосується більшості функціональних можливостей інфраструктури додатків Enterprise. Нижче наведено кілька основних переваг Spring Framework.

− Spring дозволяє розробникам розробляти корпоративні додатки, використовуючи POJO. Перевага розробки програм за допомогою POJO полягає в тому, що нам не потрібно мати корпоративний контейнер, такий як сервер

додатків, але ми маємо можливість використовувати надійний контейнер сервлетів.

− Spring забезпечує рівень абстракції для існуючих технологій, таких як сервлети, jsps, jdbc, jndi, rmi, jms та пошта Java тощо, щоб спростити процес розробки.

− Spring постачається з деякими існуючими технологіями, такими як ORM framework, logging framework, J2EE та JDK Timers тощо, отже, нам не потрібно явно інтегрувати ці технології.

− Spring може виключити створення синглтон-класу та заводських класів.

− Spring забезпечує послідовний інтерфейс управління транзакціями, який може масштабуватися до локальної транзакції та масштабуватися до глобальних транзакцій (за допомогою JTA).

− Spring framework включає підтримку управління бізнес-об’єктами та надання їх послуг компонентам рівня презентації, щоб веб-програми та настільні програми мали доступ до тих самих об’єктів.

− Спрінг-фреймворк використав найкращу практику, яка була доведена протягом багатьох років у кількох додатках та оформлена як шаблон дизайну.

− Spring додаток може бути використаний для розробки різних типів додатків, таких як автономні програми, автономні програми з графічним інтерфейсом, веб-програми та аплети.

− Spring підтримує як конфігурації xml, так і анотації.

− Spring Framework дозволяє розробити автономну, настільну, 2-шинову архітектуру та розподілені програми.

− Spring надає вбудовані послуги проміжного програмного забезпечення, такі як об'єднання з'єднань, управління транзакціями тощо

− Spring забезпечує легкий контейнер, який можна активувати без використання веб-сервера або сервера додатків.

В якості бази данних використаємо MariaDB. MariaDB - це гілка оригінального програмного забезпечення MySQL.

Співвідношення між цими двома платформами також означає, що:

- Клієнтські протоколи, API та структури однакові.
- Файли визначення таблиць та даних повністю сумісні.

Переваги використання MariaDB над MySQL:

- MariaDB пропонує жорсткіші заходи безпеки
- Показники швидші та ефективніші
- Ви отримаєте доступ до кращої підтримки користувачів

2.2 Вибір середовища розробки

Для розробки додатку найбільш вдалим рішенням буде використати інтегроване середовище програмування (Integrated Development Environment, далі IDE).

IDE, або інтегроване середовище розробки, дозволяє програмістам консолідувати різні аспекти написання комп'ютерної програми. IDE збільшують продуктивність програміста, поєднуючи загальні дії з написання програмного забезпечення в єдину програму: редагування вихідного коду, створення виконуваних файлів та налагодження.

Найпопулярніші IDE для розробки на Java:

- Eclipse
- IntelliJ Idea
- NetBeans

Для розробки було обрано IntelliJ Idea через те що в цій IDE найкращще доповнення коду та аналізатор коду, що сприяє комфорту та шквидкості розробки.

2.3 Вибір архетектури проекту

Архітектурою проекту було виброно Rest. REST API (також відомий як RESTful API) - це інтерфейс програмування програм (API або веб-API), який відповідає обмеженням архітектурного стилю REST і дозволяє взаємодіяти з веб-службами RESTful. REST позначає репрезентативну передачу стану і був створений інформатиком Роєм Філдінгом.

API - це набір визначень та протоколів для побудови та інтеграції прикладного програмного забезпечення. Іноді це називають контрактом між постачальником інформації та користувачем інформації - встановлення контенту, що вимагається від споживача (дзвінок), і контенту, який вимагає виробник (відповідь). Наприклад, конструкція API для метеорологічної служби може визначати, що користувач надає поштовий індекс, а виробник відповідає двоскладовою відповіддю, перша - висока температура, а друга - низька.

Іншими словами, якщо ви хочете взаємодіяти з комп’ютером або системою для отримання інформації або виконання функції, API допомагає вам повідомляти, що ви хочете, цій системі, щоб вона могла зрозуміти та виконати запит.Ви можете уявити API як посередника між користувачами або клієнтами та ресурсами або веб-службами, які вони хочуть отримати. Це також спосіб для організації ділитися ресурсами та інформацією, зберігаючи при цьому безпеку, контроль та автентифікацію - визначаючи, хто до чого отримує доступ.

Ще однією перевагою API є те, що вам не потрібно знати особливості кешування - як отримується ваш ресурс або звідки він береться.

REST - це набір архітектурних обмежень, а не протокол або стандарт. Розробники API можуть реалізувати REST різними способами.Коли запит клієнта робиться через RESTful API, він передає подання стану ресурсу запитувачу або кінцевій точці. Ця інформація або подання подається в одному з декількох форматів через HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP або звичайний текст. JSON - найпопулярніша мова програмування, оскільки, незважаючи на свою назву, вона є мовно-агностичною, а також доступною для читання як людьми, так і машинами.

Ще щось, про що слід пам’ятати: заголовки та параметри також важливі для методів HTTP запиту HTTP-запиту RESTful API, оскільки вони містять важливу інформацію про ідентифікатори щодо метаданих запиту, авторизації, єдиного ідентифікатора ресурсу (URI), кешування, файлів cookie та більше. Існують заголовки запитів та заголовки відповідей, кожен із яких має власну інформацію про з'єднання HTTP та коди стану.

Данні в системі представлятимуть модель MVC. Model-View-Controller (MVC) - це архітектурний шаблон, який розділяє додаток на три основні логічні компоненти: модель, вигляд та контролер. Кожен із цих компонентів створений для обробки конкретних аспектів розробки програми. MVC - одна з найбільш часто використовуваних галузевих стандартів веб-розробки для створення масштабованих та розширюваних проектів.

3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Розроблення сценарію для проекту.

Завдання 0

Для того щоб почати вивчення фреймворку Spring потрібно підготувати середовище для цього.

Потрібно встановити:

- JDK
- Intelij IDEA
- GIT
- MySQL (чи іншу базу даних)
- Maven
- Postman

Як встановити JDK дивіться - https://www.youtube.com/watch?v=IJ-PJbvJBGs&ab_channel=ProgrammingKnowledge.

Для встановлення Intelij Idea скористайтесь посиланням https://www.jetbrains.com/ru-ru/idea/download та виберіть версію (Ultimate якщо маєте ліцензію та Community якщо не маєте)

Як встановити Git дивіться - https://www.youtube.com/watch?v=2j7fD92g-gE&ab_channel=Simplilearn

Як встановити MySQL дивіться - https://www.youtube.com/watch?v=GIRcpjg-3Eg&ab_channel=edureka%21

Як встановити Maven дивіться https://www.youtube.com/watch?v=Tu2L1pg4jns&ab_channel=JavaGuides

Як встановити Postman https://www.toolsqa.com/postman/download-and-install-postman/

Після встановлення необхідних інструментів, вам потрібно створити Spring boot проект, використовуючи spring initializr https://spring.io/quickstart, чи за допомогою **Intelij IDEA Ultimate** , якщо маєте ліцензію.

Для початку буде достатньо цих залежностей: *Spring Boot DevTools , Spring Web.*Далі в процесі навчанні вы будете збільшувати їх кількість.

Додаткові посилланя для ознайомлення (обовязкове для опрацювання) : https://proselyte.net/tutorials/maven/build-life-cycle/

https://scand.com/company/blog/pros-and-cons-of-using-spring-boot/#:~:text=Spring%20Boot%20helps%20developers%20to,variety%20of%20tools%20for%20development.

https://www.baeldung.com/spring-vs-spring-boot

Завдання 1

Мета завдання:

- Спроектувати базу данних для вашого проекту. Створити ER-діаграмму.
- Продумати звязки між сутностями.При проектировці бази данних не забувати про нормальні форми.
- Для створенної структури , створити прошарок repository. Для цього використовуйте Spring Data.

Для того щоб додати Spring Data до проекту потрібно в Maven Repository (https://mvnrepository.com/) за допомогою пошуку знайти Spring Data Core , вибрати потрібну версію

Скопіювати залежність та вставити в проект в файл pom.xml в блок <dependencies> </dependencies>

Таким же чином шукаємо та додаємо **spring-boot-starter-data-jpa**, та **MySQL Connector/J** для вашої бази данних. Та виконати команду **mvn dependency:resolve** це підтяне залежності якщо вони не підтянулись автоматично.

Далі в проекті потрібно створити package з назвою *repository*, в середині якого потрібно створити package entity.В цьому package створити всі класси в залежності від вашої структури бази данних, та за допомогою Анотацій в Spring data створити залежності між ними.Також добавити необхідні коструктори та геттери з сеттерами. *Можно використовувати Lombok.*В Spring data ми маємо @OneToMany @ManyToOne @ManyToMany .Як створити залежності читайте за посиланням :

- https://www.baeldung.com/spring-data-rest-relationships
- https://www.baeldung.com/hibernate-one-to-many
- https://www.baeldung.com/jpa-one-to-one

Для того щоб Spring Data «побачила» ваші класи та згенерувала базу данних на їх основі, потрібно помітити їх анотаціями *@Entity(name = "name_entity") @Table.*Далі вам потрібно налаштувати конфігурацію вашої бази данних в application.properties що знаходится src/main/resources/application.properties.

Вам потрібно вказати :

- spring.datasource.url – url до вашої бази данних
    - ***Базу данних потрібно завчасно створити за допомогую SQL.***
- spring.datasource.username
- spring.datasource.password
- spring.datasource.driver-class-name – *імя драйверу*
- spring.jpa.generate-ddl - *чи буде спріг генерувати базу даних на основі ентіті*
- spring.jpa.hibernate.ddl-auto – *що спрінг будет робити по замовчуванням з вашою базою данних (create, create-drop, update ...)*

Далі вам потрібно перевірити чи створились таблиці в базі данних. Для цього потрібно запустити проект командою ***mvnw spring-boot:run***.На останок потрібно створити CRUD для кожної сутності.В Spring Data це робиться в 2 кроки:

1.Створити інтерфейс з назвою ***nameEntity***Repository.

2.Помітити його анотацією *@Repository*

3. Наслідуватися від *JpaRepository* та параметризувати її <entityname,typeOfId>

Наприклад:

@Repository

*public interface UserRepository extends JpaRepository<User, Integer> {}*Деталі читайте тут : [*https://habr.com/ru/post/435114/*](https://habr.com/ru/post/435114/)

Завдання 2

Мета завдання:

- Створити базовий прошарок Service для вашого CRUD.
- Створити data transfer objects (https://stackabuse.com/data-transfer-object-pattern-in-java-implementation-and-mapping/#:~:text=A%20Data%20Transfer%20Object%20is,or%20multiple%20sources%20as%20well.)
- Створити мапер Entity ↔ Dto.
- Покрити код Unit тестами (code coverage > 70%).

Для початку створіть ïїрархію packages.

```
-Src
    --main
        --java
            --com.domain.app
                    --dto
                    --repository
                    --service
                        --mapper
```

dto – для data transfer object

service – для сервісів та мапстракту

Для dto обєктів продумайте які данні ви будете відправляти/отримувати від клієнту.

Та створіть відповідні классы. *(не забувайте про getters,setters, equals and hashcode)(В більшості випадків dto та энтіті буде мати ті самі поля , але іноді dto має більше чи менше в залежності від логіки вашого applciation.)*

Використовуйте **validation-api** (попередньо додавши до pom.xml) для валідації полів дто

Приклад:

```
@Pattern(regexp = "[A-Za-zA-Яа-яёЁ]{2,200}", message = "First name inCorrect")
   private String firstName;
```

Виберіть маппер та додайте залежності до pom.xml . Реалізуйте логіку маперів в package *mapper*.

Створіть всі необхідні сервіси для ваших данних. Помічаючи їх анотацією @Service (https://www.javacodegeeks.com/2019/05/component-repository-service-spring.html).

Наприклад:

```
@Service
public class UserService {}
```

Для роботи с одним типом Энтіті , створити власний Service. (*User -> UserService, Address -> Address – service …*)

Приклад UserService:

```
@RequiredArgsConstructor
@Service
public class UserService {
   private final UserRepository userRepository;
   private final UserMapper userMapper;

   public UserDto findByUserId(Integer id) {
      return userMapper.userToUserDto(userRepository.findById(id)
```

```
            .orElseThrow(() -> new EntityNotFoundException("User id = [" + id +
                    "] not found")));
    }
        // add another methods.For example: save, update ..
}
```

Зверніть увагу що Crud Repository повертає Optional (https://www.baeldung.com/java-optional)

Також потрібно дадати логування , використовуючи логер який вам довподоби. *(Slf4g, logback….).*

Завдання 3

Мета завдання:

- Додати liquibase. https://www.liquibase.org/
- Створити change sets для створення та заповення бази данних данними.

Для того щоб додати liquibase до проекту потрібно додати залежність до pom.xml.

Потім в package – resources -> db створити файл changelog-master.yaml

Та pacage changelog , де будуть зберігатись ваші ченжсети.

Приклад changelog-master.yaml:

```yaml
databaseChangeLog:
  - includeAll:
      path: db/changelog/logs
```

Приклад change set для наповнення данними таблиці user:

```yaml
databaseChangeLog:
  - changeSet:
      id: User data
      author: Maksym Kruhovykh
      changes:
        - insert:
            tableName: user
            columns:
```

```yaml
      - column:
          name: email
          value: "maksym@gmail.com"
      - column:
          name: first_name
          value: "Maksym"
      - column:
          name: last_name
          value: "Kruhovykh"
      - column:
          name: password
          value: "qweqweqwe"
      - column:
          name: password
          value: "qweqweqwe"
  - insert:
      tableName: user
      columns:
      - column:
          name: email
```

value: "Test1@gmail.com"

- column:

name: first_name

value: "Andey"

- column:

name: last_name

value: "Shpak"

- column:

name: password

value: "qwerty123"

Тут ви бачите два INSERT для таблиці User , з колонками.

- Додайте *spring.liquibase.enabled=true до application.properties*
- Створіть власні change sets для наповнення бази данних.Використовуючи поради по написанню https://habr.com/ru/post/179425/

Перезапустіть проект, та переконайтесь в тому що чежсети примінились для вашої бази данних.В логах проекту ви побачете SQL запити заповнення вашої бази данних.

Детальніше про Liquibase https://habr.com/ru/post/339084/

https://qastack.ru/programming/29760629/why-and-when-liquibase

https://devmark.ru/article/liquibase-spring-boot

Завдання 4

Мета завдання:

- Створити controllers
- Перевірити їх роботу за допомогую Postman

Для проекту було вибрано Архітектуру REST https://www.codecademy.com/articles/what-is-rest.

Тому потрібно створити RestControlles.

Для того щоб створити controllers потрібно:

1. Створити package -> controllers
2. Створтити класс *ControllerNameController*
3. Помітити цей клас анотаціями
   - i. *@RestController*
   - ii. *@RequestMapping*

4.Реалізувати методи доступа до сервісів через контроллер.

Для цього потрібно додати анотації , обробників запитів

@ PostMapping   (для post запитів)

@GetMapping   (для get запитів)

@PutMapping   (для put запитів)

@DeleteMapping   (для delete запитів)

Детальніше:

Request Mapping: https://www.baeldung.com/spring-requestmapping

Controller vs RestController: https://itsobes.ru/JavaSobes/kakaia-raznitsa-mezhdu-controller-i-restcontroller/

Типи HTTP запитів: https://javarush.ru/quests/lectures/questcollections.level10.lecture02

https://habr.com/ru/post/50147/

Приклад ClientController:

```
@AllArgsConstructor
@RestController
@RequestMapping("/api/client")
public class ClientController {
    private static final int DEFAULT_SIZE_PAGE = 10;
    private final ClientService clientService;

    @GetMapping("/{id}")
    @ResponseStatus(HttpStatus.OK)
    public ClientDto findById(@PathVariable Integer id) {
        return clientService.findById(id);  }

    @GetMapping
    @ResponseStatus(HttpStatus.OK)
```

```
    public Page<ClientDto> findAll(@PageableDefault(size = DEFAULT_SIZE_PAGE) Pageable pageable) {
        return clientService.findAll(pageable);    }


    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public ClientDto createClient(ClientDto clientDto) {
        return clientService.create(clientDto); }
    @PutMapping
    @ResponseStatus(HttpStatus.OK)
    public ClientDto updateDriver(ClientDto clientDto) {
        return clientService.update(clientDto);}
    @DeleteMapping
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteClient(ClientDto clientDto) {
        clientService.delete(clientDto);}}
```

В Rest взаємодія між клієнтом та сервером відбувається за допомогою JSON.

Кожен метод повертає обьект в формі JSON.

Розберемося на прикладі методу

```
@GetMapping("/{id}")
```

```
@ResponseStatus(HttpStatus.OK)
public ClientDto findById(@PathVariable Integer id) {
    return clientService.findById(id);  }
```

В цьому методі відбувається пошут клієнта по ID що прийшов зі сторони клієнта.

Сервер шукає цього юзера в базі данних і якщо находить то повертає *ClientDto* та перед відправкою переобразовує його в JSON. Та додає респонс статус Ок.

Після цього вам потрібно запустити проект та за допомогою Postman протестувати що ваші ендпоінти працюють.По замовчуванням Spring Boot проект запускається локально на порті 8080.

В Постмані потрібно Створити нову колекцію для вашего проекту . New -> Collection , додати назву та опис вашої коллекціїДалі натиснути ПКМ на вашій коллекції та натиснути add request, Вказати назву та опис (за бажанням) Далі Вибрати створений Реквест , в ньйому вибрати тип запиту (get, post …) там вказати URL

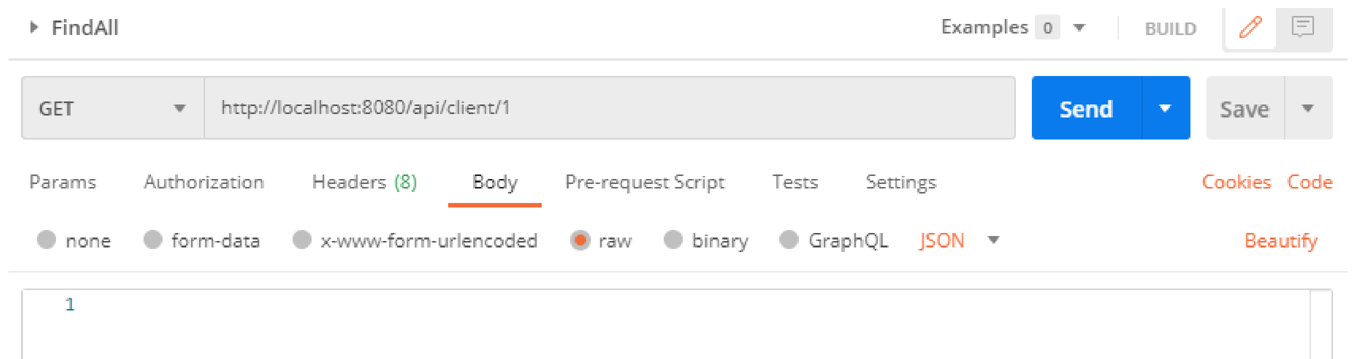

Рисунок 3.1 Postman

Додати body (якщо потрібно) та натиснути Send.

Детальніше як використовувати Postman дивіться тут https://www.youtube.com/watch?v=juldrxDrSH0&list=PLhW3qG5bs-L-oT0GenwPLcJAPD_SiFK3C&ab_channel=AutomationStepbyStep-RaghavPal

Завдання 5

Мета завдання:

- Створити власні ексепшени
- Стоврити Ексепшен Хендлер
- Додати *Swagger*

Створіть package Exception.Для того щоб створити Власні Ексепшени потрібно:

– Стоврити класс, та наслідуватись від *RuntimeException* (для *unchecked*) Та *Exception* (для *checked*)

Який тип використовувати дивіться по ситуації.

Після цього потрібно переопределити необхідні для вас методи .

Приклад:

```
public class UniqueFieldConstraintException extends RuntimeException {

   public UniqueFieldConstraintException(String message) {

      super(message);

   }

}
```

Далі потрібно створити *EceptionHandler*.Для цього створіть класс *EceptionHandler* додайте анотацію @ControllerAdvice Та створіть методи обробники ексепшенів.

Приклад :

```
@org.springframework.web.bind.annotation.ExceptionHandler(value = AccessDeniedException.class)
public ResponseEntity<ErrorResponse> accessDeniedException(AccessDeniedException exception) {
    exception.printStackTrace();
    return new ResponseEntity<>(
            createErrorResponse(exception.getMessage()),
            HttpStatus.FORBIDDEN);
}
private ErrorResponse createErrorResponse(String errorMessage) {
    return ErrorResponse.builder()
            .message(errorMessage)
            .build();
}
```

Детальніше -> https://www.baeldung.com/exception-handling-for-rest-with-spring

Створіть обробник для кожно ексепшену що використується в проекті!

Далі додайте Swagger для вашего проекту.

Swagger - то мова опису інтерфейсів для опису RESTful API, виражених за допомогою JSON. Детальніше https://swagger.io/

Для підключення його до проекту потрібно знайти потрібну залежність та додати до проекту.

– Створіть SwaggerConfig

Приклад:

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {

   @Bean
   public Docket api() {
      return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.basePackage("Ваш пакет"))
            .paths(PathSelectors.ant("/api/**"))
            .build()
            .apiInfo(ApiInfo.DEFAULT);
   }
}
```

Далі за посиланням http://localhost:8080/swagger-ui.html ви зможете побачити інформацію по ендпоінтам вашого проекту.

Наприклад:

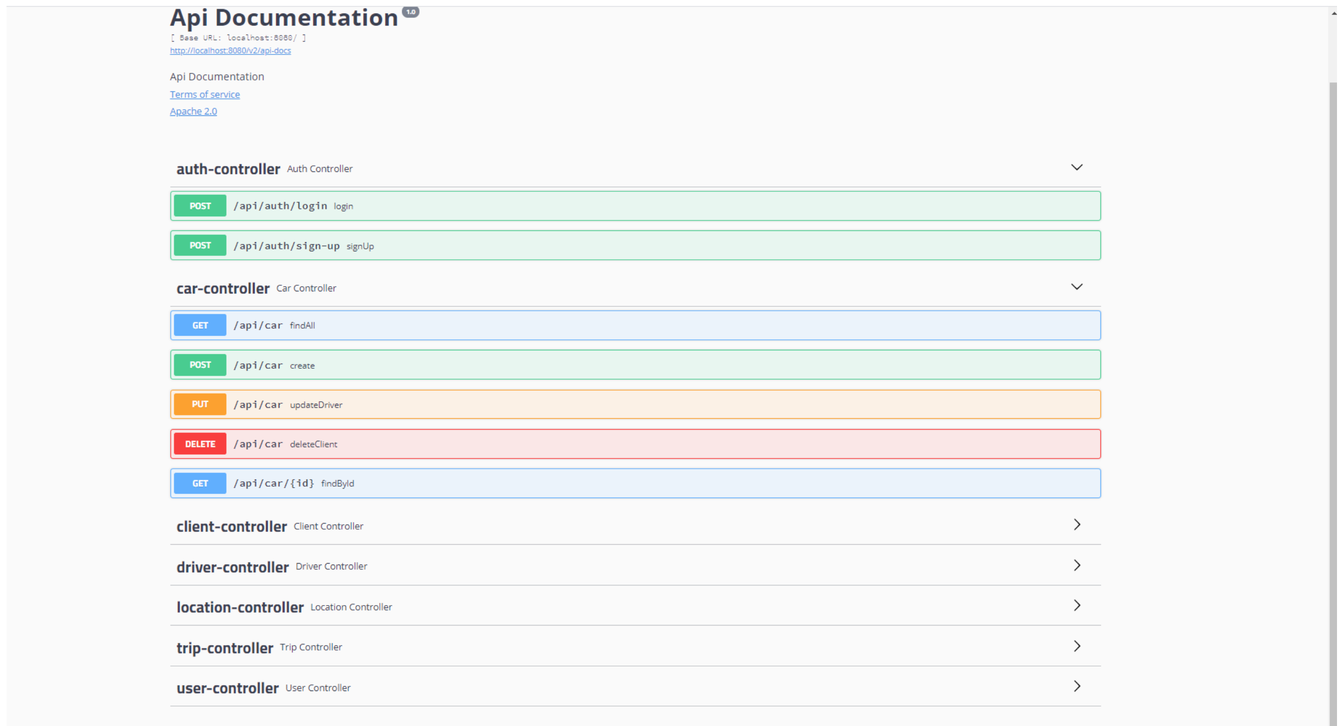

Рисунок 3.2 Postman

Завдання 6

Мета завдання:

- Додати Spring Security
  - Створити JwtFilter
  - Створити JwtProvider
  - Створити SecurityConfiguration
  - Створити UserPrincipalDetailsService

Для початку роботи з Spring Security потрібно додати залежність

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

```
    <version>2.5.0</version>
  </dependency>
```

До pom.xml

Далі створити package security та в ньому створити клас JwtProvider.Це клас який будет генерувати , валідувати оброблювати та надавати информацію по нашому JWT Токену.

Приклад:

```
@Component
public class JwtProvider {

    private static final String CLAIMS_ROLES_KEY = "auth";
    public static final String TOKEN_PREFIX = "Bearer ";
    public static final String TOKEN_HEADER = "Authorization";
    private static final String MESSAGE_WRONG_ROLE = "User must have role!";
    private String encodedSecretKey;

    @Value("${jwt.secret}")
    private String secretKey;

    @Value("${jwt.expiration.time}")
    private String expirationTimeHours;

    @PostConstruct
    private void init() {
        encodedSecretKey = Base64.getEncoder().encodeToString(secretKey.getBytes());
```

```
    }

    public String createToken(String email, Collection<String> roles) {
        Claims claims = Jwts.claims().setSubject(email);
        claims.put(CLAIMS_ROLES_KEY, roles.stream().reduce((role, prev) -> prev + ","
+ role).orElseThrow(() ->
                new IllegalArgumentException(MESSAGE_WRONG_ROLE)));

        return Jwts.builder()
                .setClaims(claims)
                .setIssuedAt(Timestamp.valueOf(LocalDateTime.now()))

.setExpiration(Timestamp.valueOf(LocalDateTime.now().plusHours(Long.parseLong(ex
pirationTimeHours))))
                .signWith(SignatureAlgorithm.HS512, encodedSecretKey)
                .compact();
    }

    public String resolveToken(HttpServletRequest req) {
        String bearerToken = req.getHeader(TOKEN_HEADER);
        if (bearerToken != null && bearerToken.startsWith(TOKEN_PREFIX)) {
            return bearerToken.substring(TOKEN_PREFIX.length());
        }
        return null;
    }

    public boolean validateToken(String token) {
```

```
        try {
            Jwts.parser().setSigningKey(encodedSecretKey).parseClaimsJws(token);
            return true;
        } catch (JwtException e) {
            throw new JwtAuthorizationException(e);
        }
    }

    public Collection<GrantedAuthority> getAuthorities(String token) {
        String auth = getClaimsFromToken(token).get(CLAIMS_ROLES_KEY).toString();
        return Stream.of(auth.split(","))
                .map(SimpleGrantedAuthority::new)
                .collect(Collectors.toList());
    }

    public String getUsername(String token) {
        return getClaimsFromToken(token).getSubject();
    }

    private Claims getClaimsFromToken(String token) {
        return
Jwts.parser().setSigningKey(encodedSecretKey).parseClaimsJws(token).getBody();
    }
```

ЗВЕРНІТЬ УВАГУ ЩО ВИКОРИСТОВУЮТЬСЯ ВЛАСНІ ЕКСЕПШЕНИ.

Далі Додати JwtFilter

Зробити наслідування від классу OncePerRequestFilter https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/filter/OncePerRequestFilter.html

Та переопреділити метод protected void doFilterInternal

Приклад:

```
@AllArgsConstructor
public class JwtFilter extends OncePerRequestFilter {
    private final JwtProvider provider;

    @Override
    protected void doFilterInternal(
            HttpServletRequest request, HttpServletResponse response, FilterChain chain)
            throws ServletException, IOException {
        String token = provider.resolveToken(request);
        try {
            if (token != null && provider.validateToken(token)) {
                Authentication authentication
                        = new UsernamePasswordAuthenticationToken(
                        provider.getUsername(token), null, provider.getAuthorities(token)
                );
                SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        } catch (JwtAuthorizationException ex) {
            SecurityContextHolder.clearContext();
            response.sendError(ex.getHttpCode(), ex.getMessage());
```

```
            return;
        }
        chain.doFilter(request, response);
    }
}
```

Далі потрібно створити UserPrincipalDetailsService який реалізує інтерфейс UserDetailsService, та переопреділити метод loadUserByUsername , який буде шукати юзера по Email в нашій базі данних.

Далі потрібно створити SecurityConfiguration позначити обовязковими анотаціями

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)

@Configuration

Наслідувати класс WebSecurityConfigurerAdapter, та переопреділити два методи protected void configure(AuthenticationManagerBuilder auth) в якому в auth потрібно засетапити значення реалізації UserDetailsService, та public void configure(HttpSecurity http) де буде конфигурація доступу до наших ендпоінтів та добавлені фільтри (JWTfilter and ect) https://spring.io/guides/gs/securing-web/

Приклад:

```
 @Override
   public void configure(HttpSecurity http) throws Exception {
       http
               .csrf().disable()
         .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
```

```
            .and()
            .authorizeRequests()
            .antMatchers("/api/auth").permitAll()
            .antMatchers("/api/client/**", "/api/driver/**", "/api/user/**", "/api/car/**").hasAuthority("ROLE_ADMIN").anyRequest().authenticated()
            .and()
            .addFilterBefore(new JwtFilter(provider), UsernamePasswordAuthenticationFilter.class)
            .cors();
   }
```

Також в цьому классі потрібно створити біни нашего password encoder та authenticationManager

```
  @Bean
   public PasswordEncoder getPasswordEncoder() {
      return new BCryptPasswordEncoder();
   }
   @Bean
   AuthenticationManager getAuthenticationManager() throws Exception {
      return authenticationManager();
   }
```

Та додати енкодінг паролю при створенні юзера

Не забудьте додати файли Swagger до конфігурації Spring Security.

Детальніше про Spring Security та JWT token:

https://www.toptal.com/spring/spring-security-tutorial

https://habr.com/ru/post/340146/

https://www.baeldung.com/spring-security-oauth-jwt

https://habr.com/ru/post/278411/

https://www.youtube.com/watch?v=X80nJ5T7YpE

https://spring.io/projects/spring-security

Завдання 7

Мета завдання:

- За допомогою АОР створити функціонал для треку часу виконная публічних методів.
- Створити кастомну анотацію яка з параметрів Security дістає дані про юзера.

Для створення функціоналу через АОР спочатку додайте Залежність

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

Далі створіть свій класс наприклад LogAspect

Помітте його анотаціями

@Aspect

@Component

Та реалізуйте два методи.

Один з Анотацією @Pointcut

```
@Pointcut(value = "execution(* maksym.kruhovykh.app.controller..*(..))" +
        " || execution(* maksym.kruhovykh.app.service..*(..))")
public void logMethodPointcut() {
}
```

Де в value передаються пакети для яких буде примінений наш функціонал.Інший метод помітте @Around

@Around("logMethodPointcut()")

public Object logMethod(ProceedingJoinPoint pjp) throws Throwable {}

Всередені цього метода реалізуйте функціонал треку часу виконнаня методу.Детальніше: https://docs.spring.io/spring-framework/docs/2.5.x/reference/aop.html.Для того щоб створити анотацію потрібно створити Інтерфейс , та перед словом interface поставити @ , наприклад

public @interface CurrentUser {}

Також потрібно додати анотації

@Target(ElementType.PARAMETER)

@Retention(RetentionPolicy.RUNTIME)

Детальніше:

https://docs.oracle.com/javase/7/docs/api/java/lang/annotation/RetentionPolicy.html

https://www.baeldung.com/java-custom-annotation

Далі потрібно написати Хендлер для цієї анотації.Так як наша анотація будет на рівні параметру та буде доставати з Security юзера створіть класс CurrentUserHandlerMethodArgumentResolver реалізуючи інтерфейс HandlerMethodArgumentResolver та переопреділіть та реалізуйте два методи

@Override

public boolean supportsParameter(MethodParameter methodParameter) { } **та**

@Override

public Object resolveArgument(MethodParameter methodParameter,

ModelAndViewContainer mavContainer, NativeWebRequest webRequest,

WebDataBinderFactory binderFactory){}

Данні про юзера знаходятся в webRequest (повертає Principal) .Вам потрібно дістати цю інформацію зробити перевірки на коректність данних , та повернути їх через return.

Детальніше про ці методи читайте в документації до інтерфейсу *HandlerMethodArgumentResolver*

Завдання 8

Мета завдання:

- Створити UI
- Створити декілька контроллерів та звязати це з UI

Для створення UI будемо використовувати **thymeleaf** додайте до залежностейОзнайомтесь з цією технологію перед написання Контроллерів.

Перш за все вам потрібно створити packages в package “resources “

- static (для статичних файлів js, css)
- тtemplates (для html)

Далі зверстайте дизайн вашого аплікейшен та додайте файли до проекту.Далі потрібно створити *WebMvcConfig* для того щоб Security допустило до файлів ресурсів.

```
@Configuration
public class WebMvcConfig implements WebMvcConfigurer {
   @Override
   public void addResourceHandlers(ResourceHandlerRegistry registry) {
      registry
            .addResourceHandler("/resources/**")
            .addResourceLocations("/resources/"); }}
```

Далі створіть звичайні (з анотацією @Controller) контроллерів та додайте thymeleaf до HTML

Детальніше дивіться https://www.baeldung.com/thymeleaf-in-spring-mvc

3.2. Реалізація діаграми класів

Діаграми класів є одним з найкорисніших типів діаграм в UML, оскільки вони чітко відображають структуру певної системи шляхом моделювання її класів, атрибутів, операцій та взаємозв’язків між об’єктами.

Уніфікована мова моделювання (UML) може допомогти вам моделювати системи різними способами. Одним з найбільш популярних типів в UML є діаграма класів. Популярні серед інженерів програмного забезпечення для документування архітектури програмного забезпечення, діаграми класів є типом структурної діаграми, оскільки вони описують те, що має бути присутнім у системі, що моделюється. Незалежно від рівня обізнаності з UML або діаграмами класів, наше програмне забезпечення UML розроблено таким, щоб бути простим і простим у використанні.

UML був створений як стандартизована модель для опису об'єктно-орієнтованого підходу до програмування. Оскільки класи є будівельним блоком об'єктів, діаграми класів є будівельними блоками UML. Різні компоненти на діаграмі класів можуть представляти класи, які насправді будуть запрограмовані, основні об’єкти або взаємодії між класами та об’єктами.

Сама форма класу складається з прямокутника з трьома рядками. Верхній рядок містить ім'я класу, середній рядок містить атрибути класу, а нижній розділ відображає методи або операції, які клас може використовувати. Класи та підкласи згруповані разом, щоб показати статичний зв'язок між кожним об'єктом.

Бібліотека форм UML у може допомогти вам створити майже будь-яку власну діаграму класу за допомогою нашого інструменту діаграм UML.

Переваги діаграм класів:

- Ілюструють моделі даних для інформаційних систем, якими б простими та складними вони не були.
- Допомогають краще зрозуміти загальний огляд схем програми.
- Допомогають візуально виразити будь-які конкретні потреби системи та поширювати цю інформацію протягом усього бізнесу.
- Допомогають створіти докладні діаграми, які висвітлюють будь-який конкретний код, який потрібно запрограмувати та реалізувати в описаній структурі.
- Надають незалежний від реалізації опис типів, що використовуються в системі, які пізніше передаються між її компонентами.

Стандартна діаграма класів складається з трьох розділів:

- Верхній розділ: Містить назву класу. Цей розділ завжди необхідний, незалежно від того, йдеться про класифікатор чи об’єкт.
- Середній розділ: Містить атрибути класу. Використовуйте цей розділ, щоб описати якості класу. Це потрібно лише при описі конкретного екземпляра класу.
- Нижній розділ: Включає операції класу (методи). Відображається у форматі списку, кожна операція займає свій рядок. Операції описують взаємодію класу з даними.

Діаграмму классів додатку та залежностей представлено нижче.

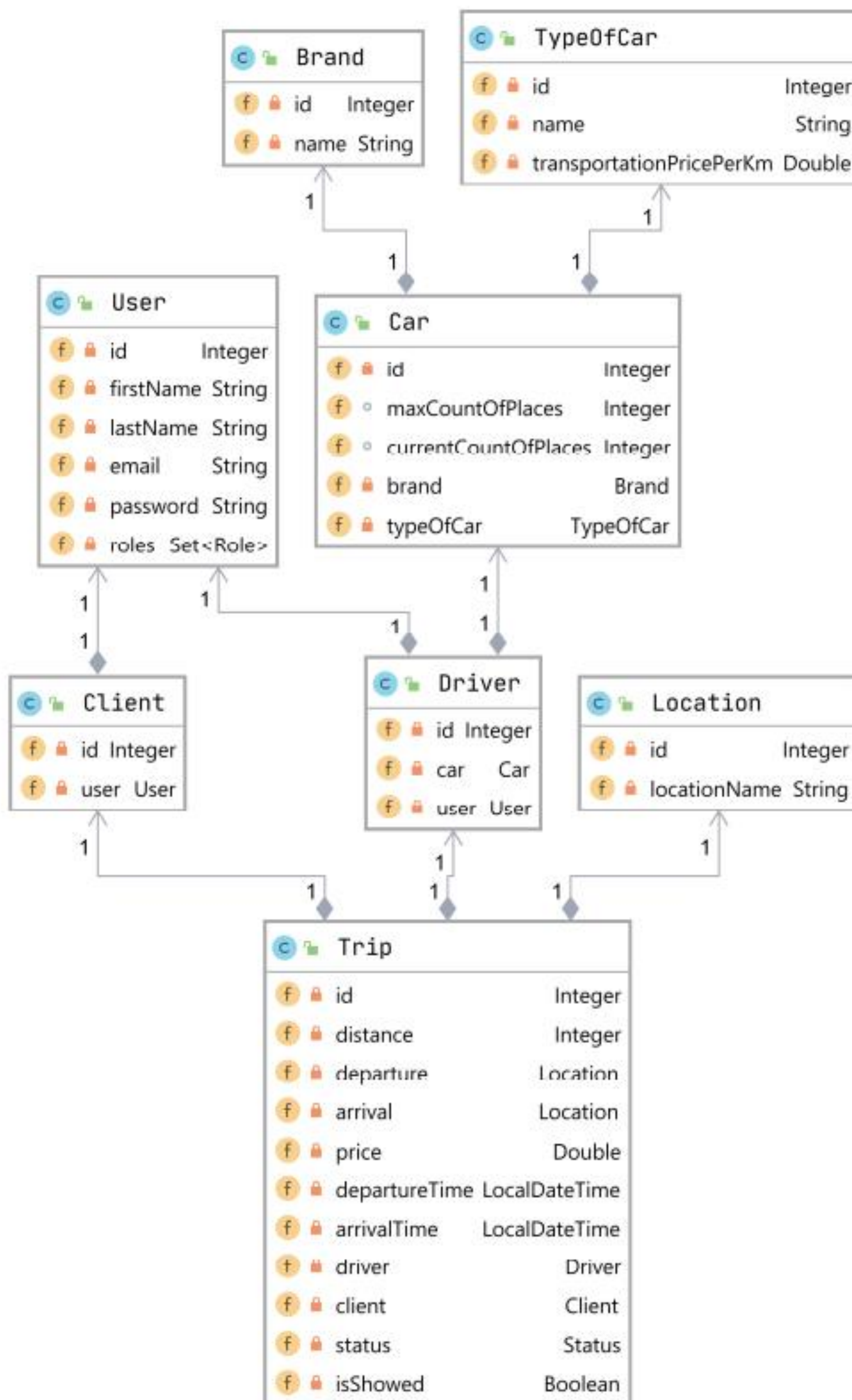

Рисунок 3.4 Entity UML

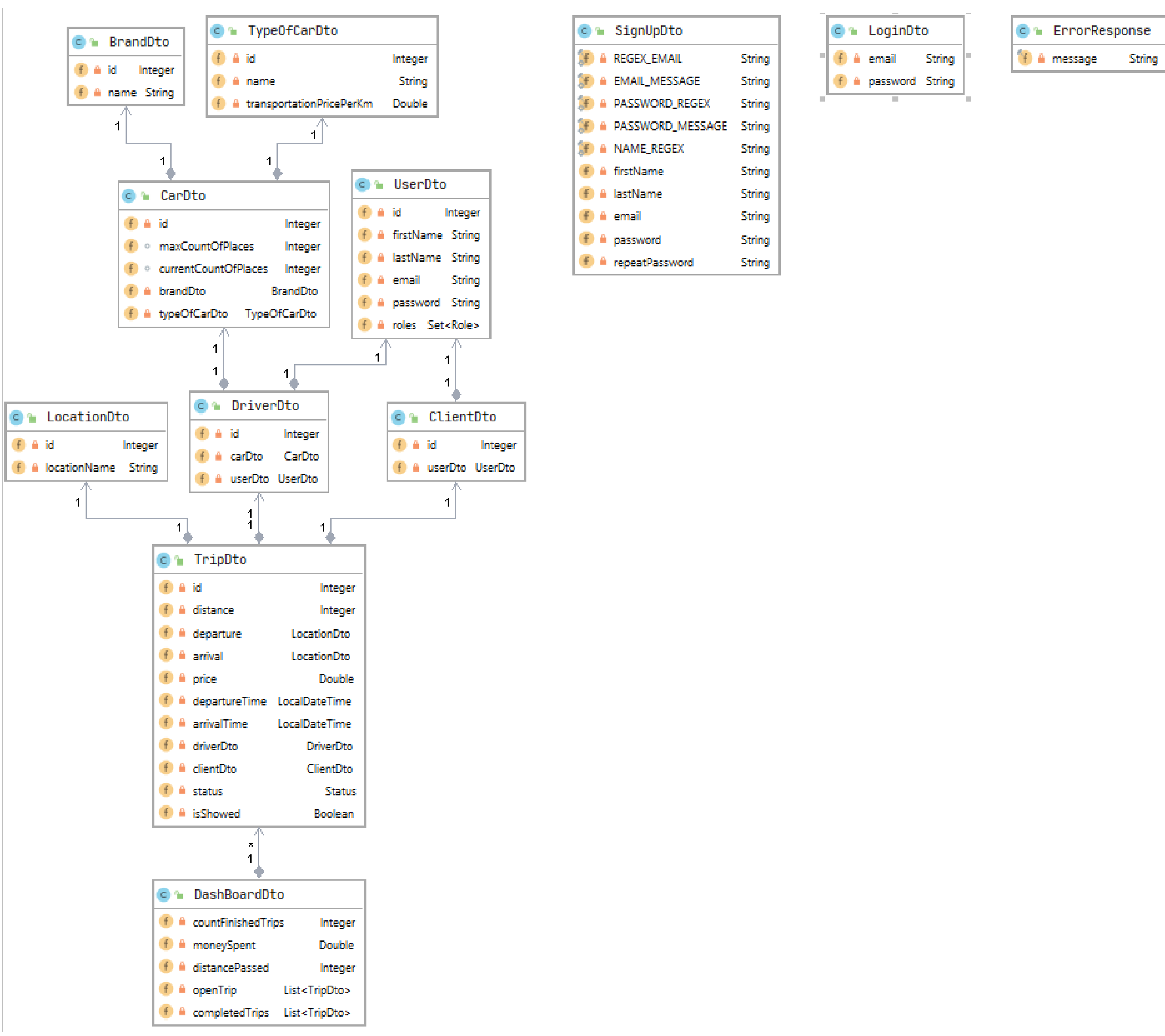

Рисунок 3.5 Dto UML

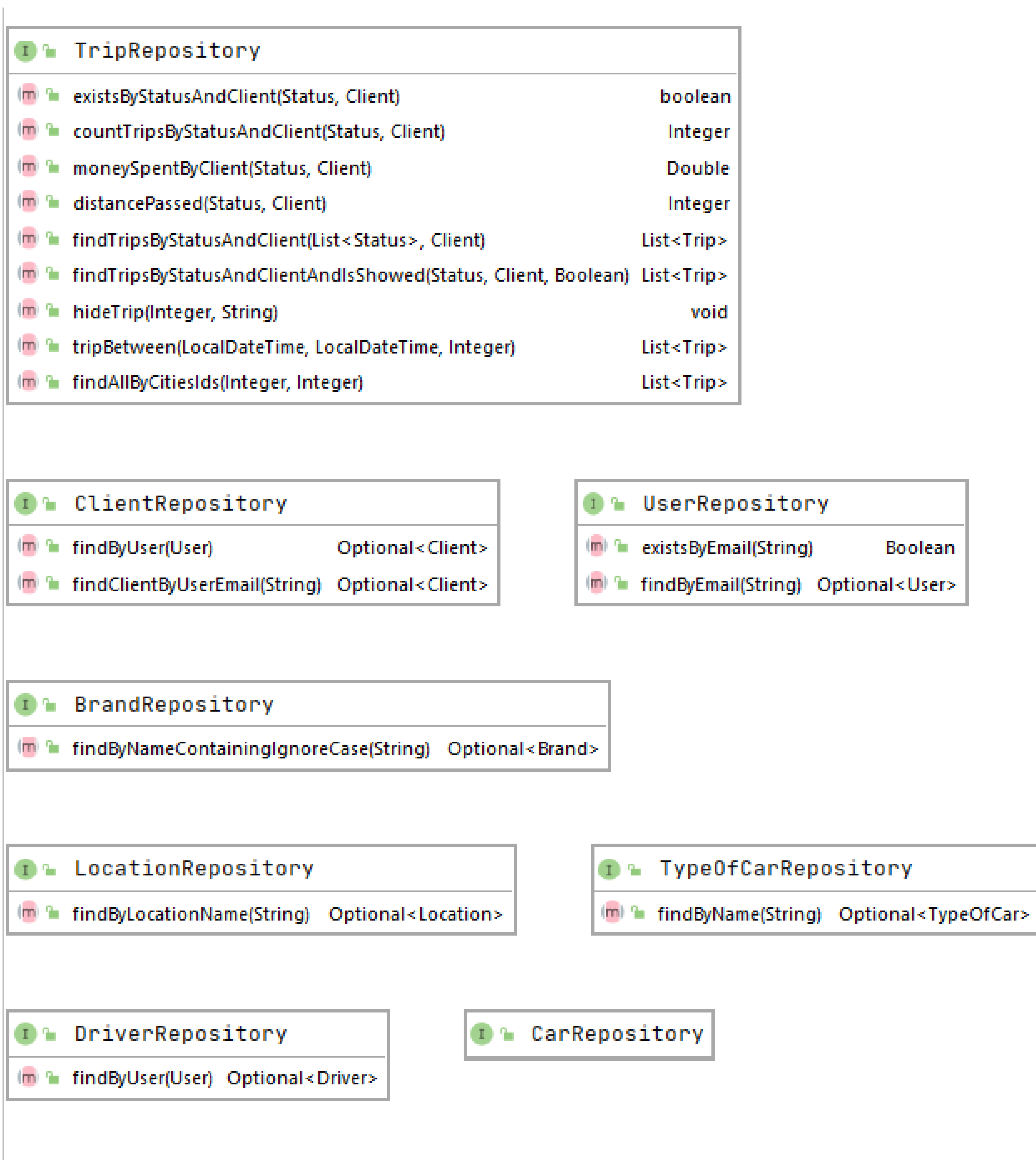

Рисунок 3.6 UML Доступ к Базі данних, Spring JPA

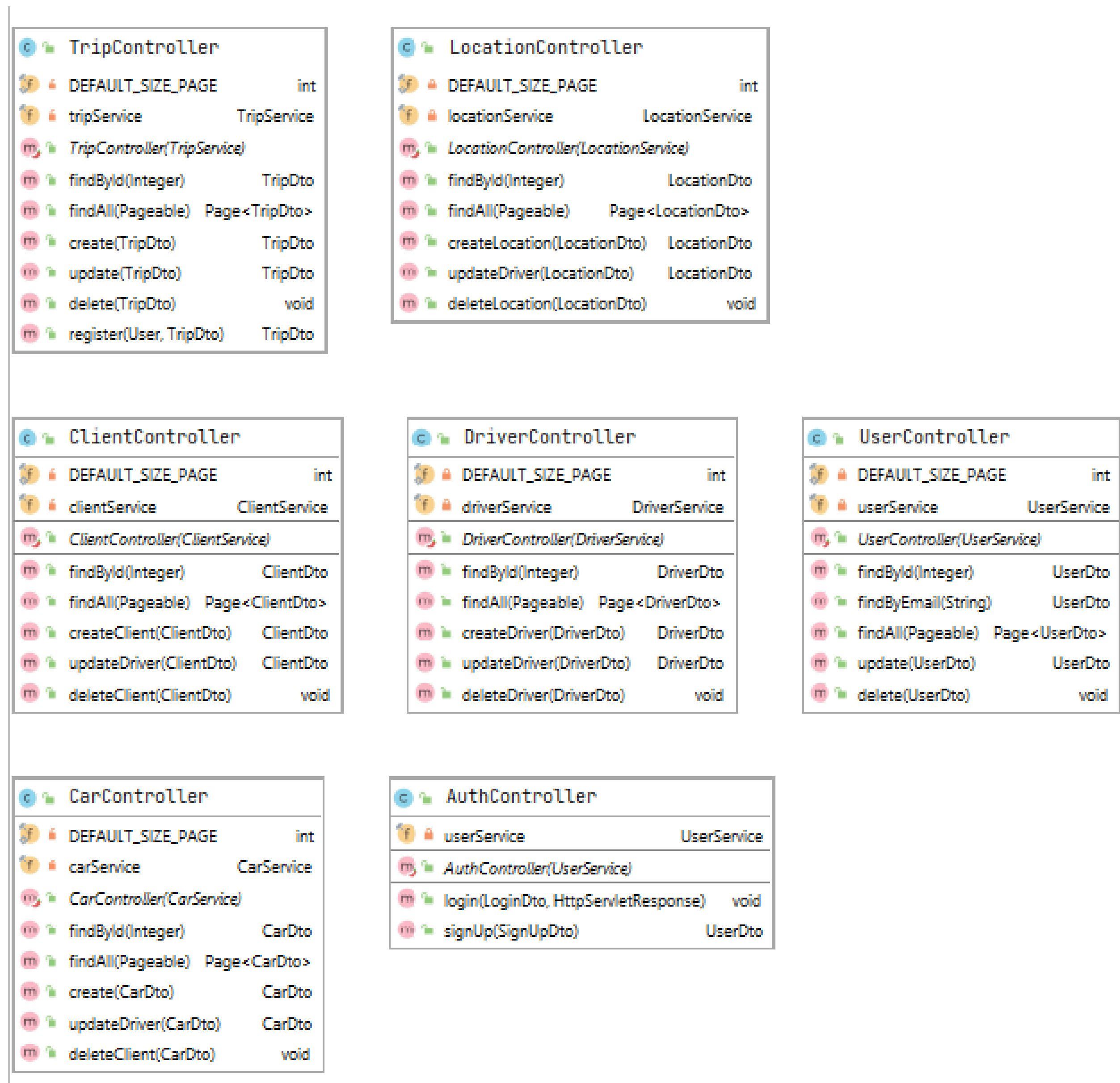

Рисунок 3.7 Controllers UML

3.2 Опис даних тренажеру

В системі для корректної роботи вже є наповення базовими данними.

Це данні про користувачів, подорожі, список міст, список брендів автомобілей, список типів автомобілів.

3.3 Опис данних користувача

Система має можливість зчитати данні від користувача. Це логін, пароль, пітвердження паролю.

Для адміністраторів додатково : бренд автомобіля , тип автомобіля , місто прибуття, місто відправки, ціна за кілометр.

Для кожного з полів введені обмеження.

- логін – може складатися із цифр та символів, має бути унікальним;
- пароль – може складатися із цифр та символів. Довжина –не менше 8 символів;
- бренд – може складатися тільки з символів.
- Місто – може складатися тільки з символів.
- Ціна за кілометр – може складаись тільки з числа.
- Тип автомобіля може складатись тільки з символів

3.4 Архітектура додатку та організація коду

Для ефективної організації коду ми використаємо патерн MVC.

Даний шаблон передбачаєрозділення класів та ресурсів додатку на наступні типи:

- Controller – класи, що приймають запити від клієнта та передають для обробки в сервіси;
- Model – класси які представляють інформацію про сутності моделі – їх поля.

View – шаблони (зазвичай HTML) що відображають графічний вигляд нашого додатку.

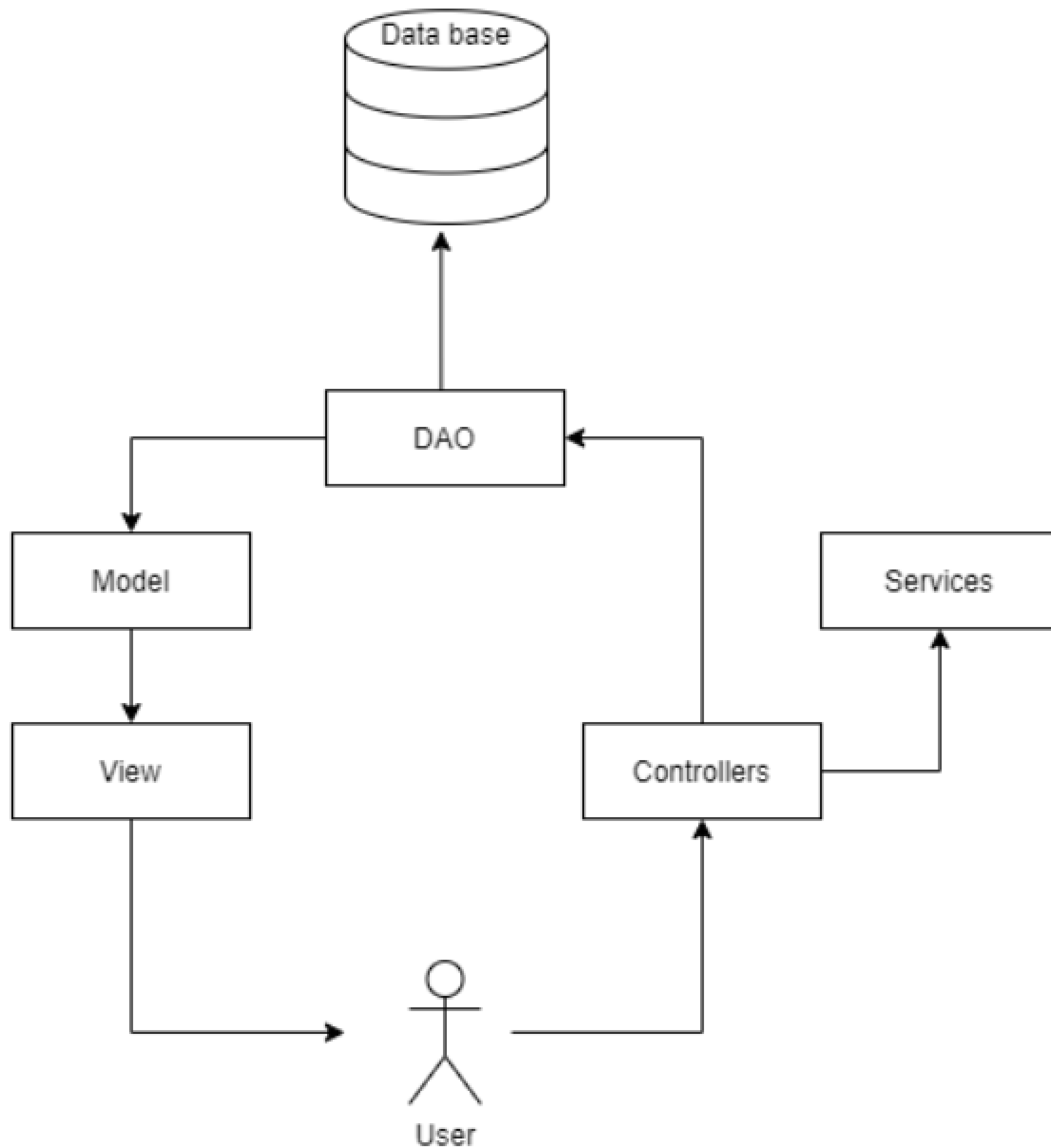

Рисунок 3.8 Схема Архітектури додатку

На рисунку 11 представлена структура проекту.

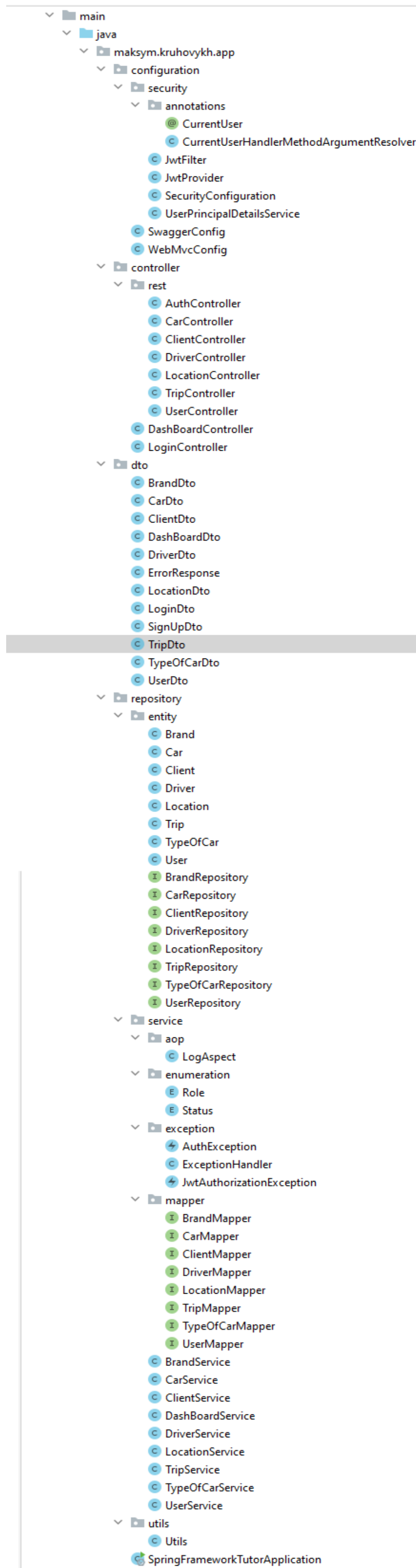
main
java
maksym.kruhovykh.app
configuration
security
annotations
CurrentUser
CurrentUserHandlerMethodArgumentResolver
JwtFilter
JwtProvider
SecurityConfiguration
UserPrincipalDetailsService
SwaggerConfig
WebMvcConfig
controller
rest
AuthController
CarController
ClientController
DriverController
LocationController
TripController
UserController
DashBoardController
LoginController
dto
BrandDto
CarDto
ClientDto
DashBoardDto
DriverDto
ErrorResponse
LocationDto
LoginDto
SignUpDto
TripDto
TypeOfCarDto
UserDto
repository
entity
Brand
Car
Client
Driver
Location
Trip
TypeOfCar
User
BrandRepository
CarRepository
ClientRepository
DriverRepository
LocationRepository
TripRepository
TypeOfCarRepository
UserRepository
service
aop
LogAspect
enumeration
Role
Status
exception
AuthException
ExceptionHandler
JwtAuthorizationException
mapper
BrandMapper
CarMapper
ClientMapper
DriverMapper
LocationMapper
TripMapper
TypeOfCarMapper
UserMapper
BrandService
CarService
ClientService
DashBoardService
DriverService
LocationService
TripService
TypeOfCarService
UserService
utils
Utils
SpringFrameworkTutorApplication

Рисунок 3.94 Структура проекту

3.5 Короткий опис программної реалізації

Розглянемо приклад реалізаціїї класу UserDto:

```
@Data
@Builder
@RequiredArgsConstructor
@AllArgsConstructor
public class UserDto {

    private Integer id;
    @Pattern(regexp = "[A-Za-zA-Яа-яёЁ]{2,200}", message = "First name inCorrect")
    private String firstName;

    @Pattern(regexp = "[A-Za-zA-Яа-яёЁ]{2,200}", message = "Last name inCorrect")
    private String lastName;

    @Pattern(regexp = "[a-zA-Z0-9]{1,}[@]{1}[a-z]{3,}[.]{1}+[a-z]{2,}", message =
"enter the email in the specified format : name@domain.com")
    private String email;

    @Pattern(regexp = "[A-Za-zA-Яа-яёЁ!_#$%^&*()-=+-]{2,32}", message = "Password
inCorrect")
    private String password;

    private Set<Role> roles;

}
```

Данний клас має анотації Lombok що дозволяє підшвидшити розробку, та покращити читабільність коду. @Data генерує геттери, сеттери, иквалс та хешкод. @Builder генерую реалізацію патерну билдер. @RequiredArgsConstructor генерує необхідні конструктори. Для доступу до бази данних використовується Spring JPA. Базові CRUD операції автоматично будуть доступні з JpaRepository.

```
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {

    Boolean existsByEmail(String email);

    Optional<User> findByEmail(String email);
}
```

Для реалізації прикладу UI ми будемо використовувати двигун Thymeleaf та Material UI + Bootstrap.

```
<div class="card card-stats">
    <div class="card-header card-header-warning card-header-icon">
        <div class="card-icon">
            <i class="material-icons">drive_eta</i>
        </div>
        <p class="card-category">Trips done</p>
        <h3 class="card-title" th:text="${dashBoard.countFinishedTrips}">
            <small>Trips</small>
        </h3>
    </div>
    <div class="card-footer">
        <div class="stats">
            <i class="material-icons">favorite</i>Travel is the only thing you buy,
that makes
            you richer.
        </div>
    </div>
</div>
```

3.6 Аналіз роботи програми

Перегляд сторінки DashBoard. На цій сторінці користувач має можливість переглянути свої подорожі та вибрати нову подорjж.

Рисунок 3.105 Dashboard

Перегляд сторінки реєстрації та входу.

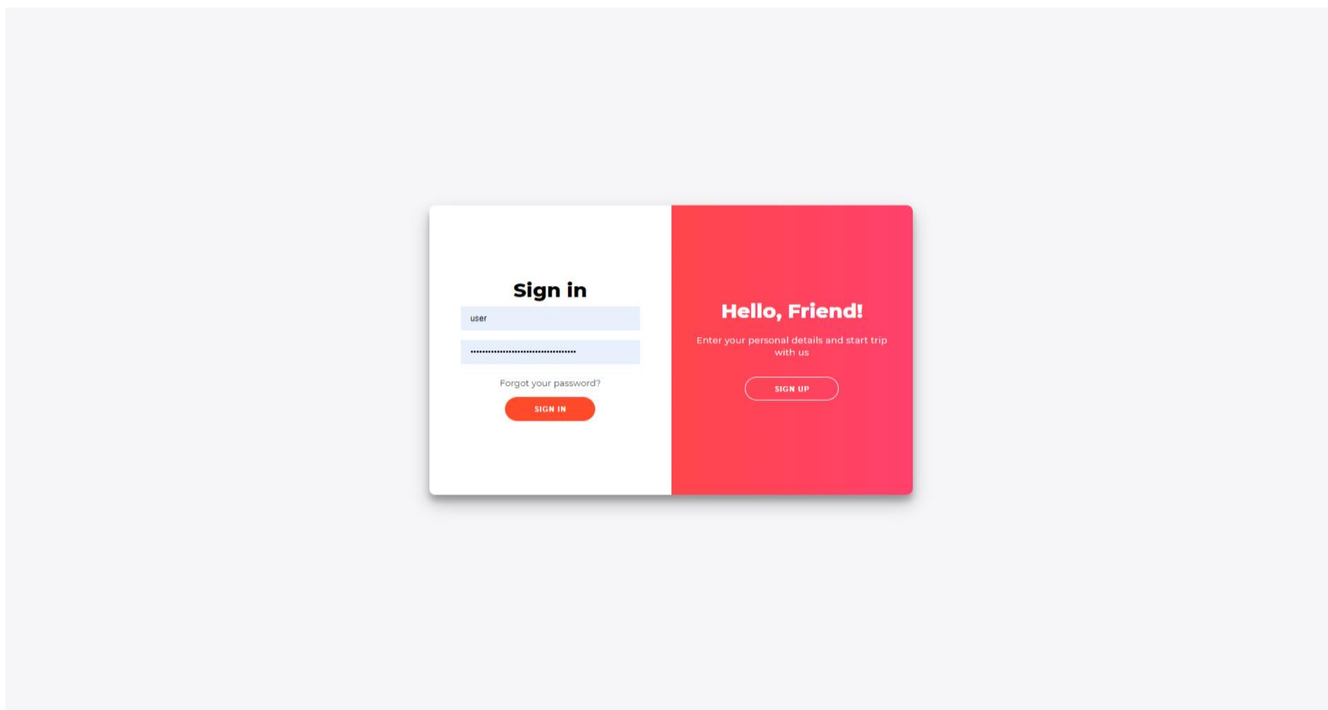

Рисунок 3.11 Sign In

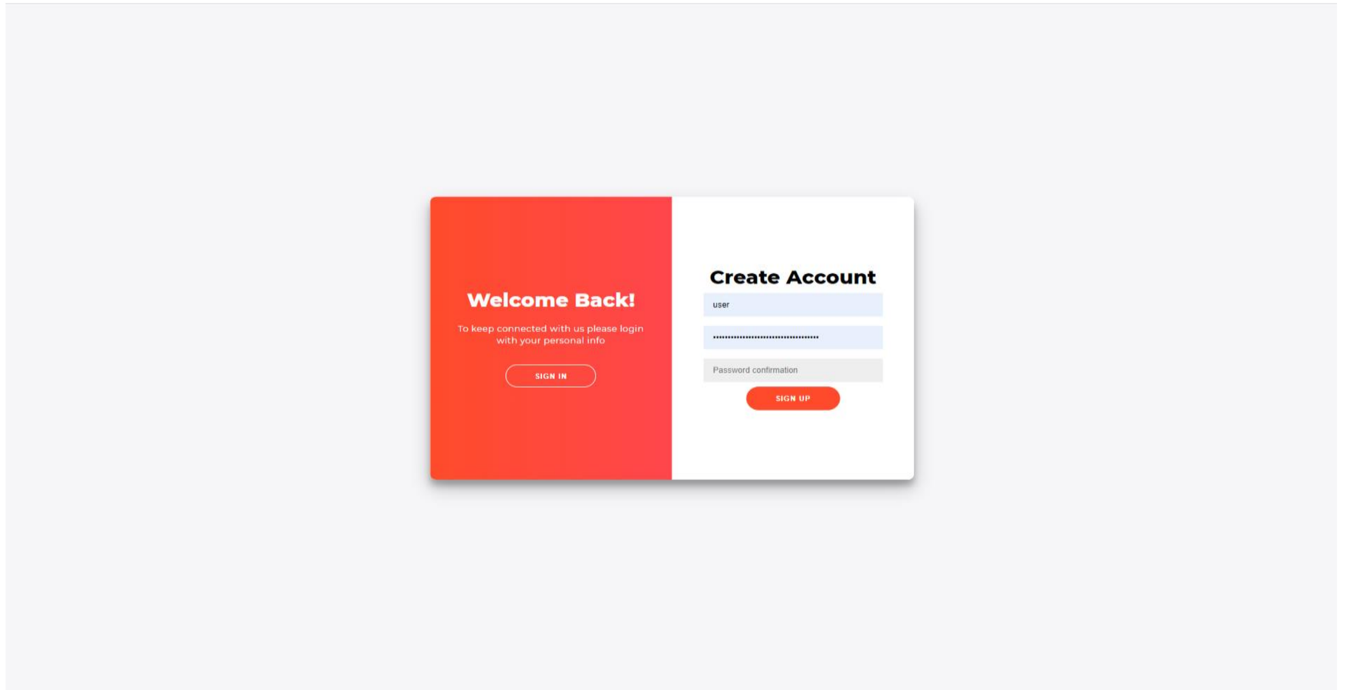

Рисунок 6 Sing up

Висновки

Робота присвячена проблемі відсутності структурованого матеріалу для вивченя Spring framework.

Було розглянуто та проаналізовано аналоги для онайлн навчання.Зроблені висновки та прийняте рішення в створенні власного тренажеру який буде відповідати всім вимогам.

Було створено дизайн та програмну реалізаціїї системи-тренажера для вивчення Spring framework a саме:

- Створено інструкція по встановленню необхідного оточення для створення продукту

- Створено набір завдання поетапне виконання яких дозволить студентам познайомитись з такими архітектурними рішеннями як: MVC, Builder,Proxy; та технологіями:Lombok,Spring Security,Spring data,SQL,AOP,OOP,DAO.
- Программно реалізовано Spring Security з використанням JWT токена.
- Програмно реалізовано функціонал заміру швидкості методи з використанням сучасного підходу для розробки AOP.
- Створено UML диаграмми класів.

На основі UML діаграм було побудовано вся структура проекту.

Для ручного тестування додатку було використано программу Postman.

Список літератури

1. Платформа Udemy [Електронний ресурс]. – Режим доступу: https://www.udemy.com/
2. Інтерактивний онлайн курс по програмуванню [Електронний ресурс]. – Режим доступу: https://javarush.ru/
3. Навчальна платформа coursera [Електронний ресурс]. – Режим доступу: https://ru.coursera.org/
4. Блог Baeldung [Електронний ресурс]. - Режим доступу:https://www.baeldung.com
5. Документація java [Електронний ресурс]. - Режим доступу: https://docs.oracle.com/
6. Документація Spring [Електронний ресурс]. - Режим доступу: https://docs.spring.io
7. Форум [Електронний ресурс]. - Режим доступу: https://habr.com/
8. Відео портал Youtube [Електронний ресурс]. - Режим доступу: https://www.youtube.com/?hl=ru
9. Craig Walls - Spring in Action, 5th Edition 2020
10. Christian Bauer - Hibernate in Action 2020
11. Thymeleaf [Електронний ресурс] : [Веб-сайт]. –Електронні дані.–Режим доступу: https://www.thymeleaf.org/

Додаток А

CurrentUserHandlerMethodArgumentResolver.class

```
@Component
public class CurrentUserHandlerMethodArgumentResolver implements
    HandlerMethodArgumentResolver {

  @Override
  public boolean supportsParameter(MethodParameter methodParameter) {
    return methodParameter.getParameterAnnotation(CurrentUser.class) != null
        && methodParameter.getParameterType().equals(User.class);
  }

  @Override
  public Object resolveArgument(MethodParameter methodParameter,
                        ModelAndViewContainer mavContainer, NativeWebRequest
webRequest,
                        WebDataBinderFactory binderFactory) throws Exception {

    Principal principal;
    if (supportsParameter(methodParameter) && (principal =
webRequest.getUserPrincipal()) != null) {
      return ((Authentication) principal).getPrincipal();
    }

    return null;
  }
```

JwtFilter.class

```
@AllArgsConstructor
public class JwtFilter extends OncePerRequestFilter {

   private final JwtProvider provider;

   @Override
   protected void doFilterInternal(
         HttpServletRequest request, HttpServletResponse response, FilterChain chain)
         throws ServletException, IOException {
      String token = provider.resolveToken(request);
      try {
         if (token != null && provider.validateToken(token)) {
            Authentication authentication
                  = new UsernamePasswordAuthenticationToken(
                  provider.getUsername(token), null, provider.getAuthorities(token)
            );
            SecurityContextHolder.getContext().setAuthentication(authentication);
         }
      } catch (JwtAuthorizationException ex) {
         SecurityContextHolder.clearContext();
         response.sendError(ex.getHttpCode(), ex.getMessage());
         return;
      }
      chain.doFilter(request, response); }}}
```

JwtProvider.class

```
@Component
public class JwtProvider {

   private static final String CLAIMS_ROLES_KEY = "auth";
   public static final String TOKEN_PREFIX = "Bearer ";
   public static final String TOKEN_HEADER = "Authorization";
   private static final String MESSAGE_WRONG_ROLE = "User must have role!";
   private String encodedSecretKey;

   @Value("${jwt.secret}")
   private String secretKey;

   @Value("${jwt.expiration.time}")
   private String expirationTimeHours;

   @PostConstruct
   private void init() {
      encodedSecretKey = Base64.getEncoder().encodeToString(secretKey.getBytes());
   }

   public String createToken(String email, Collection<String> roles) {
      Claims claims = Jwts.claims().setSubject(email);
      claims.put(CLAIMS_ROLES_KEY, roles.stream().reduce((role, prev) -> prev + ","
+ role).orElseThrow(() ->
            new IllegalArgumentException(MESSAGE_WRONG_ROLE)));

      return Jwts.builder()
```

```
            .setClaims(claims)
            .setIssuedAt(Timestamp.valueOf(LocalDateTime.now()))

.setExpiration(Timestamp.valueOf(LocalDateTime.now().plusSeconds(Long.parseLong(
expirationTimeHours))))
            .signWith(SignatureAlgorithm.HS512, encodedSecretKey)
            .compact();
   }

   public String resolveToken(HttpServletRequest req) {
      String bearerToken = req.getHeader(TOKEN_HEADER);
      if (bearerToken != null && bearerToken.startsWith(TOKEN_PREFIX)) {
         return bearerToken.substring(TOKEN_PREFIX.length());
      }
      return null;
   }

   public boolean validateToken(String token) {
      try {
         Jwts.parser().setSigningKey(encodedSecretKey).parseClaimsJws(token);
         return true;
      } catch (JwtException e) {
         throw new JwtAuthorizationException(e);
      } }
```

SecurityConfiguration.class

```
@AllArgsConstructor
```

```
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

   private final UserDetailsService userDetailsService;
   private final JwtProvider provider;

   @Override
   protected void configure(AuthenticationManagerBuilder auth) throws Exception {
      auth.userDetailsService(userDetailsService);
   }

   @Override
   public void configure(HttpSecurity http) throws Exception {
      http
              .csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
              .and()
              .authorizeRequests()
              .antMatchers("/api/auth/**","/dashboard","/login","/**").permitAll()
              .antMatchers("/api/client/**", "/api/driver/**", "/api/user/**",
"/api/car/**").hasAuthority("ROLE_ADMIN").anyRequest().authenticated()
              .and()
              .addFilterBefore(new JwtFilter(provider),
UsernamePasswordAuthenticationFilter.class)
```

```
            .cors();
    }

    @Bean
    public PasswordEncoder getPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    AuthenticationManager getAuthenticationManager() throws Exception {
        return authenticationManager();
    }

    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers(AUTH_WHITELIST);
    }
    private static final String[] AUTH_WHITELIST = {
            "/swagger-resources/**",
            "/swagger-ui.html",
            "/v2/api-docs",
            "/webjars/**",
            "/resources/"  };} public Collection<GrantedAuthority> getAuthorities(String
token) {
        String auth = getClaimsFromToken(token).get(CLAIMS_ROLES_KEY).toString();
        return Stream.of(auth.split(","))
                .map(SimpleGrantedAuthority::new)
```

```
            .collect(Collectors.toList());
    }

    public String getUsername(String token) {
        return getClaimsFromToken(token).getSubject();
    }

    private Claims getClaimsFromToken(String token) {
        return
Jwts.parser().setSigningKey(encodedSecretKey).parseClaimsJws(token).getBody();
    }
}
```

UserPrincipalDetailsService.class

```
@Service
@AllArgsConstructor
public class UserPrincipalDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) {
        Optional<User> user = userRepository.findByEmail(email);

        return user.map(this::toUserDetails)
            .orElseThrow(() -> new UsernameNotFoundException("User [" + email + "]
doesn't exist"));
    }
```

```
    private UserDetails toUserDetails(User user) {
        return org.springframework.security.core.userdetails.User
                .withUsername(user.getEmail())
                .password(user.getPassword())
                .roles(user.getRoles().stream().map(Enum::name).toArray(String[]::new))
                .build();
    }
}
```

UserService.class

```
@RequiredArgsConstructor
@Slf4j
@Service
public class UserService {

    private static final String USER_IS_EMPTY = "User is Empty";
    private final UserRepository userRepository;
    private final UserMapper userMapper;
    private final AuthenticationManager authenticationManager;
    private final JwtProvider jwtProvider;
    private final PasswordEncoder passwordEncoder;


    @Transactional
    public String createUserToken(LoginDto loginDto) {
        String email = loginDto.getEmail();
```

```
        if (userRepository.findByEmail(email).isPresent()) {
            try {
                return getAuthenticationToken(loginDto);
            } catch (BadCredentialsException e) {
                throw new AuthException("Wrong Password");
            }
        } else {
            throw new AuthException("User [" + loginDto.getEmail() + "] doesn't exist");
        }
    }

    public UserDto findByUserId(Integer id) {
        return userRepository
                .findById(id)
                .map(userMapper::userToUserDto)
                .orElseThrow(() -> new EntityNotFoundException("User id = [" + id +
                        "] not found"));
    }

    public UserDto update(UserDto userDto) {
        Utils.isNull(userDto, USER_IS_EMPTY);
        return userRepository
                .findById(userDto.getId())
                .map(updateUser(userDto))
                .map(userMapper::userToUserDto)
                .orElseThrow(() -> new EntityNotFoundException("User with Id [" +
userDto.getId() + "] doesn't exist"));
```

```
    }

    public UserDto createUser(SignUpDto signUpDto) {
        if (!signUpDto.getPassword().equals(signUpDto.getRepeatPassword())) {
            throw new AuthException("Passwords must match");
        }

        if (userRepository.existsByEmail(signUpDto.getEmail())) {
            throw new EntityExistsException("User with email [" + signUpDto.getEmail() +
"] already exist");
        }

        User user = User.builder()
                .firstName(signUpDto.getFirstName())
                .lastName(signUpDto.getLastName())
                .email(signUpDto.getEmail())
                .password(passwordEncoder.encode(signUpDto.getPassword()))
                .roles(Collections.singleton(Role.CLIENT))
                .build();

        User save = userRepository.save(user);
        save.setPassword(null);

        return userMapper.userToUserDto(save);
    }

    public Page<UserDto> findAll(Pageable pageable) {
```

```
        Page<User> clientPage = userRepository.findAll(pageable);
        return clientPage.map(userMapper::userToUserDto);
    }

    public void delete(UserDto userDto) {
        userRepository.findByEmail(userDto.getEmail())
                .map(user -> {
                    userRepository.delete(user);
                    return user;
                })
                .orElseThrow(EntityNotFoundException::new);
    }

    public UserDto findByEmail(String email) {
        return userRepository.findByEmail(email)
                .map(userMapper::userToUserDto)
                .orElseThrow(EntityNotFoundException::new);
    }

    private String getAuthenticationToken(LoginDto loginDto) {
        String email = loginDto.getEmail();

        Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(email,
loginDto.getPassword()));

        Collection<String> roles = authentication.getAuthorities().stream()
```

```
        .map(GrantedAuthority::getAuthority).collect(Collectors.toSet());

    return jwtProvider.createToken(email, roles);
  }

  private Function<User, User> updateUser(UserDto userDto) {
    return user -> {
      User save = userMapper.userDtoToUser(userDto);

      userRepository.save(save);

      log.info("User with id [" + userDto.getId() + "] updated ");
      return save;

    };
  }}
```

TypeOfCarService.class

```
@RequiredArgsConstructor
@Service
@Slf4j
public class TypeOfCarService {
  private static final String TYPE_OF_CAR_IS_EMPTY = "Type of car is Empty";

  private final TypeOfCarRepository typeOfCarRepository;
  private final TypeOfCarMapper typeOfCarMapper;
```

```
    public TypeOfCarDto findById(Integer id) {
        return typeOfCarRepository
                .findById(id)
                .map(typeOfCarMapper::typeOfCarToTypeOfCarDto)
                .orElseThrow(EntityNotFoundException::new);
    }

    public TypeOfCarDto create(TypeOfCarDto typeOfCarDto) {
        Utils.isNull(typeOfCarDto, TYPE_OF_CAR_IS_EMPTY);

        if (typeOfCarRepository.findByName(typeOfCarDto.getName()).isPresent()) {
            log.error("Type of car already exist");
            throw new EntityExistsException("Type of car already exist");
        }

        TypeOfCar typeOfCar =
typeOfCarMapper.typeOfCarDtoToTypeOfCar(typeOfCarDto);

        return
typeOfCarMapper.typeOfCarToTypeOfCarDto(typeOfCarRepository.save(typeOfCar));
    }

    public TypeOfCarDto update(TypeOfCarDto typeOfCarDto) {
        Utils.isNull(typeOfCarDto, TYPE_OF_CAR_IS_EMPTY);

        return typeOfCarRepository
```

```
            .findById(typeOfCarDto.getId())
            .map(typeOfCar -> {
                typeOfCar =
typeOfCarMapper.typeOfCarDtoToTypeOfCar(typeOfCarDto);
                typeOfCarRepository.save(typeOfCar);
                return typeOfCarDto;
            })
            .orElseThrow(EntityNotFoundException::new);
    }

    public void delete(TypeOfCarDto typeOfCarDto) {
        Utils.isNull(typeOfCarDto, TYPE_OF_CAR_IS_EMPTY);

        typeOfCarRepository.findById(typeOfCarDto.getId())
            .map(typeOfCar -> {
                typeOfCarRepository.delete(typeOfCar);
                log.info("Type of Car  " + typeOfCar + " deleted");
                return typeOfCarDto;
            })
            .orElseThrow(EntityNotFoundException::new);
    }

}
```

TripService.class

```
@RequiredArgsConstructor
@Slf4j
@Service
```

```
public class TripService {

    private static final String TRIP_IS_EMPTY = "Trip is empty";
    private static final Integer averageSpeed = 80;
    private static final int HOURS_FOR_REST = 2;

    private final ClientService clientService;
    private final TripRepository tripRepository;
    private final TripMapper tripMapper;

    public TripDto findById(Integer id) {
        return tripRepository.findById(id)
                .map(tripMapper::tripToTripDto)
                .orElseThrow(() -> new EntityNotFoundException("Trip  [" + id + "] doesn't
exist"));
    }

    public TripDto create(TripDto tripDto) {
        Utils.isNull(tripDto, TRIP_IS_EMPTY);
        Trip trip = tripMapper.tripDtoToTrip(tripDto);
        Integer distance = trip.getDistance();
        calculateTimeDeparture(trip, distance);

        checkPossibilityOfCreatingOrder(trip);

        calculatePrice(trip, distance);
```

```
        return tripMapper.tripToTripDto(tripRepository.save(trip));

    }

    public TripDto registerForTrip(TripDto tripDto, String clientEmail) {
        TripDto trip = tripRepository.findById(tripDto.getId())
                .map(tripMapper::tripToTripDto)
                .orElseThrow(EntityNotFoundException::new);
        ClientDto client = clientService.findClientByUserEmail(clientEmail);
        CarDto carDto = trip.getDriverDto().getCarDto();

        if (carDto.getMaxCountOfPlaces() - carDto.getCurrentCountOfPlaces() < 1) {
            log.error("Car is Full");
            throw new RuntimeException("Car is Full");
        }
        trip.setClientDto(client);

        return
tripMapper.tripToTripDto(tripRepository.save(tripMapper.tripDtoToTrip(trip)));
    }

    public TripDto update(TripDto tripDto) {
        Utils.isNull(tripDto, TRIP_IS_EMPTY);
        return tripRepository
                .findById(tripDto.getId())
                .map(updateOrder(tripDto))
                .map(tripMapper::tripToTripDto)
```

```
                .orElseThrow(() -> new EntityNotFoundException("Trip with Id [" +
tripDto.getId() + "] doesn't exist"));

    }

    public Page<TripDto> findAll(Pageable pageable) {
        Page<Trip> tripDtos = tripRepository.findAll(pageable);
        return tripDtos.map(tripMapper::tripToTripDto);
    }

    public void delete(TripDto tripDto) {
        Utils.isNull(tripDto, TRIP_IS_EMPTY);
        tripRepository
                .findById(tripDto.getId())
                .map(deleteOrder(tripDto))
                .map(tripMapper::tripDtoToTrip)
                .orElseThrow(() -> new EntityNotFoundException("Trip with Id [" +
tripDto.getId() + "] doesn't exist"));

    }

    private Function<Trip, TripDto> deleteOrder(TripDto tripDto) {
        return tripDto1 -> {
            tripRepository.delete(tripDto1);
            return tripDto;
        };
    }
```

```
    private void calculateTimeDeparture(Trip trip, Integer distance) {
        int timeTrip = distance / averageSpeed;
        LocalDateTime departureTime = trip.getDepartureTime();
        trip.setDepartureTime(departureTime.plusHours(timeTrip +
HOURS_FOR_REST));
    }

    private void calculatePrice(Trip trip, Integer distance) {
        Double transportationPricePerKm =
trip.getDriver().getCar().getTypeOfCar().getTransportationPricePerKm();
        trip.setPrice(transportationPricePerKm * distance);
    }

    private Function<Trip, Trip> updateOrder(TripDto tripDto) {
        return trip -> {
            Trip save = tripMapper.tripDtoToTrip(tripDto);
            return tripRepository.save(save);

        };
    }

    private void checkPossibilityOfCreatingOrder(Trip trip) {
        boolean periodIsEmpty = tripRepository.tripBetween(
                trip.getArrivalTime(),
                trip.getDepartureTime(),
                trip.getDriver().getId())
```

```
            .isEmpty();
        if (!periodIsEmpty) {
            log.warn("Cannot create trip.Driver " + trip.getDriver() + " is busy");
            throw new RuntimeException("Cannot create trip.Driver "
                    + trip.getDriver().getUser().getFirstName() + " "
                    + trip.getDriver().getUser().getLastName()
                    + " is busy"); } }}
```

Location.service

```
@RequiredArgsConstructor
@Service
@Slf4j
public class LocationService {

    private static final String ADDRESS_IS_EMPTY = "Location is empty";

    private final LocationRepository locationRepository;
    private final LocationMapper locationMapper;



    public LocationDto findById(Integer id) {
        return locationRepository
                .findById(id)
                .map(locationMapper::locationToLocationDto)
                .orElseThrow(EntityNotFoundException::new);
    }

    public List<LocationDto> findAll() {
```

```
        List<Location> all = locationRepository.findAll();
        return all.stream()
                .map(locationMapper::locationToLocationDto)
                .sorted(Comparator.comparing(LocationDto::getLocationName))
                .collect(Collectors.toList());

    }

    public Page<LocationDto> findAll(Pageable pageable) {
        Page<Location> clientPage = locationRepository.findAll(pageable);

        return clientPage.map(locationMapper::locationToLocationDto);
    }

    public LocationDto create(LocationDto locationDto) {
        Utils.isNull(locationDto, ADDRESS_IS_EMPTY);
        Location location = locationMapper.locationDtoToLocation(locationDto);

        if
(locationRepository.findByLocationName(location.getLocationName()).isPresent()) {
            log.error("Location  [" + location.getLocationName() + "] already exist");
            throw new EntityExistsException("Location [" + location.getLocationName() +
"] already exist");
        }

        return locationMapper.locationToLocationDto(locationRepository.save(location));
    }
```

```
    public LocationDto update(LocationDto locationDto) {
        Utils.isNull(locationDto, ADDRESS_IS_EMPTY);

        return locationRepository.findById(locationDto.getId())
                .map(location -> {
                    location = locationMapper.locationDtoToLocation(locationDto);
                    locationRepository.save(location);
                    return locationDto;
                }).orElseThrow(EntityNotFoundException::new);
    }

    public void delete(LocationDto locationDto) {
        Utils.isNull(locationDto, ADDRESS_IS_EMPTY);
        locationRepository.findById(locationDto.getId())
                .map(location -> {
                    locationRepository.delete(location);
                    log.info("Location  " + location.getLocationName() + " deleted");
                    return locationDto;
                })
                .orElseThrow(EntityNotFoundException::new);
    }
}
```

Driverservice.class

```
@RequiredArgsConstructor
```

```
@Service
@Slf4j
public class DriverService {
   private static final String DRIVER_IS_EMPTY = "Driver is Empty";
   private final DriverRepository driverRepository;
   private final DriverMapper driverMapper;

   public DriverDto findById(Integer id) {
      return driverRepository
            .findById(id)
            .map(driverMapper::driverToDriverDto)
            .orElseThrow(EntityNotFoundException::new);
   }

   public DriverDto update(DriverDto driverDto) {
      Utils.isNull(driverDto, DRIVER_IS_EMPTY);

      return driverRepository.findById(driverDto.getId())
            .map(driver -> {
               driver = driverMapper.driverDtoToDriver(driverDto);
               driverRepository.save(driver);
               return driverDto;
            }).orElseThrow(EntityNotFoundException::new);
   }

   public DriverDto create(DriverDto driverDto) {
      Utils.isNull(driverDto, DRIVER_IS_EMPTY);
```

```
    Driver driver = driverMapper.driverDtoToDriver(driverDto);

    if (driverRepository.findByUser(driver.getUser()).isPresent()) {
      log.error("Driver  Already exist");
      throw new EntityExistsException("Driver  Already exist");
    }
    return driverMapper.driverToDriverDto(driverRepository.save(driver));
  }

  public void delete(DriverDto driverDto) {
    Utils.isNull(driverDto, DRIVER_IS_EMPTY);
    driverRepository.findById(driverDto.getId())
        .map(driver -> {
          driverRepository.delete(driver);
          log.info("Driver " + driver + " deleted");
          return driverDto;
        })
        .orElseThrow(EntityNotFoundException::new);
  }

  public Page<DriverDto> findAll(Pageable pageable) {
    Page<Driver> clientPage = driverRepository.findAll(pageable);
    return clientPage.map(driverMapper::driverToDriverDto);
  }
}
```

ClientService.class

```
@RequiredArgsConstructor
```

```
@Service
@Slf4j
public class ClientService {

    public static final String CLIENT_IS_EMPTY = "Client is Empty";
    private final ClientRepository clientRepository;
    private final ClientMapper clientMapper;
    private final UserMapper userMapper;

    public ClientDto findById(Integer id) {
        return clientRepository
                .findById(id)
                .map(clientMapper::clientToClientDto)
                .orElseThrow(EntityNotFoundException::new);
    }

    public ClientDto findClientByUserEmail(String email) {
        return clientRepository.findClientByUserEmail(email)
                .map(clientMapper::clientToClientDto)
                .orElseThrow(EntityNotFoundException::new);
    }

    public Page<ClientDto> findAll(Pageable pageable) {
        Page<Client> clientPage = clientRepository.findAll(pageable);
        return clientPage.map(clientMapper::clientToClientDto);
    }
```

```
public ClientDto update(ClientDto clientDto) {
    Utils.isNull(clientDto, CLIENT_IS_EMPTY);

    return clientRepository
            .findById(clientDto.getId())
            .map(updateUser(clientDto))
            .orElseThrow(EntityNotFoundException::new);
}

public ClientDto create(ClientDto clientDto) {
    Utils.isNull(clientDto, CLIENT_IS_EMPTY);

    Client client = clientMapper.clientDtoToClient(clientDto);

    if (clientRepository.findByUser(client.getUser()).isPresent()) {
        log.error("Client  Already exist");
        throw new EntityExistsException("Client  Already exist");
    }

    return clientMapper.clientToClientDto(clientRepository.save(client));

}

public void delete(ClientDto clientDto) {
    Utils.isNull(clientDto, CLIENT_IS_EMPTY);
    clientRepository.findById(clientDto.getId())
            .map(client -> {
```

```
                clientRepository.delete(client);
                log.info("Client " + client + " deleted");
                return clientDto;
            })
            .orElseThrow(EntityNotFoundException::new);
    }

    private Function<Client, ClientDto> updateUser(ClientDto clientDto) {
        return client -> {
            client.setId(clientDto.getId());
            client.setUser(userMapper.userDtoToUser(clientDto.getUserDto()));
            clientRepository.save(client);
            log.info("Client" + client + " updated");
            return clientDto;

        };
    }
}
```

CarService.class

```
@RequiredArgsConstructor
@Service
@Slf4j
public class CarService {
    private static final String CAR_IS_EMPTY = "Car is Empty";
    private final CarRepository carRepository;
    private final CarMapper carMapper;
```

```
public CarDto findById(Integer id) {
    return carRepository.findById(id)
            .map(carMapper::carToCarDto)
            .orElseThrow(() -> new EntityNotFoundException("Car with id [" + id +
                    "] not found"));

}

public CarDto create(CarDto carDto) {
    Utils.isNull(carDto, CAR_IS_EMPTY);

    Car car = carMapper.carDtoToCar(carDto);

    return carMapper.carToCarDto(carRepository.save(car));

}
public Page<CarDto> findAll(Pageable pageable) {
    Page<Car> clientPage = carRepository.findAll(pageable);
    return clientPage.map(carMapper::carToCarDto);
}

public void delete(CarDto carDto) {
    Utils.isNull(carDto, CAR_IS_EMPTY);

    carRepository
            .findById(carDto.getId())
```

```
            .map(deleteCar(carDto))
            .orElseThrow(() -> new EntityNotFoundException("Car with Id [" +
carDto.getId() + "] doesn't exist"));

    }

    public CarDto update(CarDto carDto) {
        Utils.isNull(carDto, CAR_IS_EMPTY);
        isNotExistThrowException(carDto);

        Car car = carMapper.carDtoToCar(carDto);

        return carMapper.carToCarDto(carRepository.save(car));
    }


    private void isNotExistThrowException(CarDto carDto) {
        if (!carRepository.findById(carDto.getId()).isPresent()) {
            log.error("Car with Id [" + carDto.getId() + "] doesn't exist");
            throw new EntityNotFoundException("Car with Id [" + carDto.getId() + "]
doesn't exist");
        }
    }

    private Function<Car, CarDto> deleteCar(CarDto carDto) {
        return order -> {
            carRepository.delete(order);
```

```
            return carDto;
        };
    }

}
```

BrandService.class

```
@RequiredArgsConstructor
@Service
@Slf4j
public class BrandService {
    private static final String BRAND_IS_EMPTY = "Brand is Empty";

    private final BrandRepository brandRepository;
    private final BrandMapper brandMapper;

    public BrandDto findById(Integer id) {
        return brandRepository
                .findById(id)
                .map(brandMapper::brandToBrandDto)
                .orElseThrow(EntityNotFoundException::new);
    }

    public BrandDto create(BrandDto brandDto) {
        Utils.isNull(brandDto, BRAND_IS_EMPTY);

        Brand brand = brandMapper.brandDtoToBrand(brandDto);
```

```
        if
(brandRepository.findByNameContainingIgnoreCase(brand.getName()).isPresent()) {
            log.error("Brand [" + brand.getName() + "] already exist");
            throw new EntityExistsException("Brand [" + brand.getName() + "] already
exist");
        }

        return brandMapper.brandToBrandDto(brandRepository.save(brand));
    }

    public BrandDto update(BrandDto brandDto) {
        Utils.isNull(brandDto, BRAND_IS_EMPTY);
        return brandRepository
                .findById(brandDto.getId())
                .map(updateBrand(brandDto))
                .orElseThrow(EntityNotFoundException::new);

    }

    public void delete(BrandDto brandDto) {
        Utils.isNull(brandDto, BRAND_IS_EMPTY);

        brandRepository
                .findById(brandDto.getId())
                .map(deleteBrand())
                .orElseThrow(EntityNotFoundException::new);
    }
```

```
    private Function<Brand, Brand> deleteBrand() {
        return brand -> {
            brandRepository.delete(brand);
            log.info("Brand " + brand + " deleted");
            return brand;
        };
    }

    private Function<Brand, BrandDto> updateBrand(BrandDto brandDto) {
        return brand -> {
            brand.setId(brandDto.getId());
            brand.setName(brandDto.getName());
            brandRepository.save(brand);
            log.info("Brand [" + brand + "]  saved");
            return brandDto;
        };
    }

}
```

ExceptionHandler.class

```
@Slf4j
@ControllerAdvice
public class ExceptionHandler {
    @org.springframework.web.bind.annotation.ExceptionHandler(value =
AuthException.class)
    public ResponseEntity<ErrorResponse> authException(AuthException exception) {
```

```
        return new ResponseEntity<>(
                createErrorResponse(exception.getMessage()),
                HttpStatus.UNAUTHORIZED);
    }

    @org.springframework.web.bind.annotation.ExceptionHandler(value =
JwtAuthorizationException.class)
    public ResponseEntity<ErrorResponse>
jwtAuthorizationException(JwtAuthorizationException exception) {
        return new ResponseEntity<>(
                createErrorResponse(exception.getMessage()),
                HttpStatus.UNAUTHORIZED);
    }

    @org.springframework.web.bind.annotation.ExceptionHandler(value =
EntityExistsException.class)
    public ResponseEntity<ErrorResponse>
entityAlreadyExistsException(EntityExistsException exception) {
        return new ResponseEntity<>(
                createErrorResponse(exception.getMessage()),
                HttpStatus.PRECONDITION_FAILED);
    }

    @org.springframework.web.bind.annotation.ExceptionHandler(value =
EntityNotFoundException.class)
    public ResponseEntity<ErrorResponse>
entityNotFoundException(EntityNotFoundException exception) {
```

```
        return new ResponseEntity<>(
                createErrorResponse(exception.getMessage()),
                HttpStatus.NOT_FOUND);
    }

    @org.springframework.web.bind.annotation.ExceptionHandler(value =
AccessDeniedException.class)
    public ResponseEntity<ErrorResponse>
accessDeniedException(AccessDeniedException exception) {
        exception.printStackTrace();
        return new ResponseEntity<>(
                createErrorResponse(exception.getMessage()),
                HttpStatus.FORBIDDEN);
    }
    private ErrorResponse createErrorResponse(String errorMessage) {
        return ErrorResponse.builder()
                .message(errorMessage) .build();  }}
```