**German University in Cairo**
**Department of Computer Science**
**Assoc. Prof. Mervat Abu El-Kheir**

**Architecture of Massively Scalable Applications**, Spring 2025

# Mini Project 2

**Deadline is Friday 25/04/2025 11:59 PM**

## 1 Introduction

This project is a more advanced Spring Boot application featuring a dual-database design. You are required to use two data stores: PostgreSQL for relational data and MongoDB for NoSQL documents. The system models a ride-sharing service with entities such as **Captain**, **Customer**, **Trip**, and **Payment** as relational entities, while **Rating** is stored as a document in MongoDB. The aim of the project is to implement complete CRUD operations, custom query methods, and RESTful endpoints across multiple layers: model, repository, service, and controller.

## 2 Installation

Start a new Spring Boot project via the Spring Initializer at: `https://start.spring.io/` When setting up your Spring Boot project using Spring Initializer, you must include the essential dependencies in your `pom.xml` file to support both PostgreSQL and MongoDB. The following dependencies are recommended:

Listing 1: Essential Dependencies in pom.xml

```
<dependencies>
  <!-- Spring Boot starter web for building web applications -->
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <!-- Spring Boot starter data JPA for PostgreSQL integration -->
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <!-- PostgreSQL driver -->
  <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <scope>runtime</scope>
  </dependency>

  <!-- Spring Boot starter data MongoDB for MongoDB integration -->
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-mongodb</artifactId>
  </dependency>
</dependencies>
```

Set the following:

- **Project:** Maven
- **Language:** Java
- **Spring Boot:** 3.4.4
- **Packaging:** Jar
- **Java:** 23 (please note that if Java 23 is not in the options you can choose Java 24 and change it to 23 in pom.xml file)

Then configure the `docker-compose.yml` file (as previously shown in the lab manuals) to launch the required containers for PostgreSQL, MongoDB, and the Spring Boot application.

# 3 Project Structure

The project must follow a standard Spring Boot layered architecture with the package names as indicated below.

```
|- project-root/
|    |- src/
|    |    |- main/
|    |    |    |- java/com/example/miniapp
|    |    |    |    |- models
|    |    |    |    |    |- Captain.java       // Relational entity
|    |    |    |    |    |- Customer.java      // Relational entity
|    |    |    |    |    |- Trip.java          // Relational entity
|    |    |    |    |    |- Payment.java       // Relational entity
|    |    |    |    |    |- Rating.java        // MongoDB document
|    |    |    |    |- repositories
|    |    |    |    |    |- CaptainRepository.java
|    |    |    |    |    |- CustomerRepository.java
|    |    |    |    |    |- TripRepository.java
|    |    |    |    |    |- PaymentRepository.java
|    |    |    |    |    |- RatingRepository.java
|    |    |    |    |- services
|    |    |    |    |    |- CaptainService.java
|    |    |    |    |    |- CustomerService.java
|    |    |    |    |    |- TripService.java
|    |    |    |    |    |- PaymentService.java
|    |    |    |    |    |- RatingService.java
|    |    |    |    |- controllers
|    |    |    |         |- CaptainController.java
|    |    |    |         |- CustomerController.java
|    |    |    |         |- TripController.java
|    |    |    |         |- PaymentController.java
|    |    |    |         |- RatingController.java
|    |    |- resources
|    |         |- application.properties  // Contains DB settings for SQL and MongoDB
|- docker-compose.yml   // To launch PostgreSQL, MongoDB, and the Spring Boot app
```

# 4 Data

The application requires two different databases:

- **PostgreSQL**: Stores the relational entities (`Captain`, `Customer`, `Trip`, and `Payment`).

- **MongoDB**: Stores the `Rating` documents which are used to record feedback on trips, captains, or customers.

Database configuration (host, port, credentials, etc.) is managed within the `application.properties` file.

# 5 Models

Each model must include getters, setters, and at least three constructors (default, partial, full) as specified before in previous labs.

## 5.1 Captain

Represents a captain (driver) stored in PostgreSQL. Each captain has the following attributes:

- Long Unique ID (IDENTITY generation strategy)
- String Name
- String License Number
- Double AvgRatingScore

A captain may have many trips.

## 5.2 Customer

Represents a customer stored in PostgreSQL. Each customer has the following attributes:

- Long Unique ID (IDENTITY generation strategy)
- String Name
- String Email
- String PhoneNumber

A customer may be associated with multiple trips.

## 5.3 Trip

A trip represents a journey. It holds the following information:

- Long Unique ID (IDENTITY generation strategy)
- LocalDateTime TripDate
- String Origin
- String Destination
- Double TripCost

It has many-to-one relationships with both ***Captain*** and ***Customer***.

### 5.4 Payment

Handles payment data for each trip. It contains the following information:

- Long Unique ID (IDENTITY generation strategy)
- Double amount
- String paymentMethod (e.g., *card*, *cash*)
- Boolean paymentStatus; (*True* if paid)

The Payment is linked to a specific trip.

### 5.5 Rating

The `Rating` document is stored in MongoDB and provides feedback with a score and comment on an entity. It is represented using the following attributes:

- String ID
- Long entityId (ID of the entity rated (*captain*, *customer*, or *trip*))
- String entityType (*captain*, *customer*, or *trip*)
- Integer score (Rating score (1-5))
- String comment
- LocalDateTime ratingDate

---

## 6 Relations

### 6.1 One-To-Many

a) ***Captain*** – ***Trip***:

- A single Captain can be responsible for multiple Trips.
- The ***Trip*** entity is the owner

b) ***Customer*** – ***Trip***:

- A single Customer can take multiple Trips.
- The ***Trip*** entity is again the owner

### 6.2 One-To-One

a) ***Trip*** – ***Payment***:

- Each Trip has one Payment record associated with it, and each Payment record is linked with a single Trip.
- The ***Payment*** entity is the owner

---

# 7 Repositories

Implement repository interfaces for both SQL and NoSQL operations.

## 7.1 SQL Repositories

Each relational entity (Captain, Customer, Trip, Payment) should have a dedicated repository.

### 7.1.1 CaptainRepository

Manage queries for captain data, and should include:

- Query to find all captains with a rating above a threshold.
- Query to locate a captain by license number.

### 7.1.2 CustomerRepository

Should include the following queries:

- Finding customers by their email domain.
- Finding customers by phone prefix.

### 7.1.3 TripRepository

Should include the following queries:

- Retrieving trips within a specified date range.
- Filtering trips by captain ID.

### 7.1.4 PaymentRepository

Should include the following queries:

- Finding payments by trip ID.
- Finding payments with an amount above a set threshold.

## 7.2 MongoDB Repository

### 7.2.1 RatingRepository

This repository deals with `Rating`, and have the following queries:

- Find ratings for a given entity (by ID and type).
- Retrieve ratings with a score above a specified value.

# 8 Services

All business logic is handled by the services. Remember, the services act as the intermediary layer between the controllers and the repositories. You should create a dedicated service for each model.

## 8.1 CaptainService

The service responsible for handling business logic related to the `Captain` entity.

### 8.1.1 Class Definition

```
@Service
public class CaptainService {
    // Dependency Injection
    private final CaptainRepository captainRepository;

    public CaptainService(CaptainRepository captainRepository) {
        this.captainRepository = captainRepository;
    }
}
```

### 8.1.2 Required Methods

#### 8.1.2.1 Add Captain

To add a new captain into the system.

```
public Captain addCaptain(Captain captain);
```

#### 8.1.2.2 Get All Captains

To retrieve all captains in the system.

```
public List<Captain> getAllCaptains();
```

#### 8.1.2.3 Get Captain By ID

To retrieve a specific captain by their ID.

```
public Captain getCaptainById(Long id);
```

#### 8.1.2.4 Filter by Rating

To retrieve all captains with a rating greater than a given threshold.

```
public List<Captain> getCaptainsByRating(Double ratingThreshold);
```

#### 8.1.2.5 Filter by License Number

To retrieve a captain by their specific license number.

```
public Captain getCaptainByLicenseNumber(String licenseNumber);
```

---

## 8.2 CustomerService

The service handling the functionalities related to the **Customer**.

### 8.2.1 Class Definition

```
@Service
public class CustomerService {

    private final CustomerRepository customerRepository;

    @Autowired
    public CustomerService(CustomerRepository customerRepository) {
        this.customerRepository = customerRepository;
    }

    // Business logic methods here...
}
```

### 8.2.2 Required Methods

#### 8.2.2.1 Add Customer

To add a new customer to the system.

```
public Customer addCustomer(Customer customer);
```

#### 8.2.2.2 Get All Customers

To retrieve all customers.

```
public List<Customer> getAllCustomers();
```

#### 8.2.2.3 Get Customer By ID

To retrieve a specific customer by their ID.

```
public Customer getCustomerById(Long id);
```

#### 8.2.2.4 Update Customer

To update a customer's information.

```
public Customer updateCustomer(Long id, Customer customer);
```

#### 8.2.2.5 Delete Customer

To delete a customer record from the system.

```
public void deleteCustomer(Long id);
```

#### 8.2.2.6 Find Customers By Email Domain

To retrieve all customers whose email addresses end with a specific domain.

```
public List<Customer> findCustomersByEmailDomain(String domain);
```

#### 8.2.2.7 Find Customers By Phone Prefix

To retrieve all customers whose phone numbers start with a specific prefix.

```
public List<Customer> findCustomersByPhonePrefix(String prefix);
```

---

## 8.3 TripService

The service handling the functionalities related to the **Trip**.

### 8.3.1 Class Definition

```
@Service
public class TripService {

    private final TripRepository tripRepository;

    @Autowired
    public TripService(TripRepository tripRepository) {
        this.tripRepository = tripRepository;
    }

    // Business logic methods here...
}
```

### 8.3.2 Required Methods

#### 8.3.2.1 Add Trip

To create and schedule a new trip.

```
public Trip addTrip(Trip trip);
```

#### 8.3.2.2 Get All Trips

To retrieve all trips in the system.

```
public List<Trip> getAllTrips();
```

#### 8.3.2.3 Get Trip By ID

To retrieve a specific trip by its ID.

```
public Trip getTripById(Long id);
```

#### 8.3.2.4 Update Trip

To update trip details such as origin, destination, or cost.

```
public Trip updateTrip(Long id, Trip trip);
```

#### 8.3.2.5 Delete Trip

To delete a trip from the system.

```
public void deleteTrip(Long id);
```

#### 8.3.2.6 Find Trips Within a Date Range

To retrieve all trips that occurred between two specific dates.

```
public List<Trip> findTripsWithinDateRange(LocalDateTime startDate, LocalDateTime endDate);
```

#### 8.3.2.7 Find Trips By Captain ID

To retrieve all trips associated with a specific captain's ID.

```
public List<Trip> findTripsByCaptainId(Long captainId);
```

---

## 8.4 PaymentService

The service handling the functionalities related to the **Payment**.

### 8.4.1 Class Definition

```
@Service
public class PaymentService {

    private final PaymentRepository paymentRepository;

    @Autowired
    public PaymentService(PaymentRepository paymentRepository) {
        this.paymentRepository = paymentRepository;
    }

    // Business logic methods here...
}
```

### 8.4.2 Required Methods

#### 8.4.2.1 Add Payment

To record a new payment for a trip.

```
public Payment addPayment(Payment payment);
```

#### 8.4.2.2 Get All Payments

To retrieve all payment records.

```
public List<Payment> getAllPayments();
```

#### 8.4.2.3 Get Payment By ID

To retrieve a specific payment by its ID.

```
public Payment getPaymentById(Long id);
```

#### 8.4.2.4 Update Payment

To update payment details (for example, the payment status).

```
public Payment updatePayment(Long id, Payment payment);
```

#### 8.4.2.5 Delete Payment

To remove a payment record from the system.

```
public void deletePayment(Long id);
```

#### 8.4.2.6 Find Payments By Trip ID

To retrieve all payments associated with a specific trip ID.

```
public List<Payment> findPaymentsByTripId(Long tripId);
```

#### 8.4.2.7 Find Payments With an Amount Greater Than a Threshold

To retrieve all payments where the amount exceeds a specified threshold.

```
public List<Payment> findByAmountThreshold(Double threshold);
```

## 8.5 RatingService

The service handling the functionalities related to the **Rating** (stored in MongoDB).

### 8.5.1 Class Definition

```
@Service
public class RatingService {

    private final RatingRepository ratingRepository;

    @Autowired
    public RatingService(RatingRepository ratingRepository) {
        this.ratingRepository = ratingRepository;
    }

    // Business logic methods here...
}
```

### 8.5.2 Required Methods

#### 8.5.2.1 Add Rating

To add a new rating.

```
public Rating addRating(Rating rating);
```

#### 8.5.2.2 Update Rating

To update an existing rating's score or comments.

```
public Rating updateRating(String id, Rating updatedRating);
```

#### 8.5.2.3 Delete Rating

To delete a rating by its ID.

```
public void deleteRating(String id);
```

#### 8.5.2.4 Get Ratings By Entity

To retrieve the ratings linked to a specific entity by its ID and type.

```
public List<Rating> getRatingsByEntity(Long entityId, String entityType);
```

#### 8.5.2.5 Find Ratings Above a Specific Value

To retrieve all ratings where the score is greater than a specified value.

```
public List<Rating> findRatingsAboveScore(int minScore);
```

---

# 9 Controllers

You should implement RESTful api endpoints for each entity mentioned above.

## 9.1 CaptainController

To handle the endpoints related to the **Captain** entity.

### 9.1.1 Class Definition

```
@RestController
@RequestMapping("/captain")
public class CaptainController {

    private final CaptainService captainService;

    @Autowired
    public CaptainController(CaptainService captainService) {
        this.captainService = captainService;
    }

    // RESTful endpoint methods...
}
```

#### 9.1.2 Required Endpoints

##### 9.1.2.1 Add Captain

Post request to add a new captain.

```
@PostMapping("/addCaptain")
public Captain addCaptain(@RequestBody Captain captain);
```

##### 9.1.2.2 Get All Captains

Get request to retrieve all captains.

```
@GetMapping("/allCaptains")
public List<Captain> getAllCaptains();
```

##### 9.1.2.3 Get Specific Captain

Get request to retrieve a captain by ID.

```
@GetMapping("/{id}")
public Captain getCaptainById(@PathVariable Long id);
```

##### 9.1.2.4 Filter Captains By Rating

Get request to retrieve captains whose rating exceeds a specific threshold.

```
@GetMapping("/filterByRating")
public List<Captain> getCaptainsByRating(@RequestParam Double ratingThreshold);
```

##### 9.1.2.5 Filter Captain By License Number

Get request to retrieve a captain with a specific license number.

```
@GetMapping("/filterByLicenseNumber")
public Captain getCaptainByLicenseNumber(@RequestParam String licenseNumber);
```

---

### 9.2 CustomerController

To handle the endpoints related to the **Customer** entity.

#### 9.2.1 Class Definition

```
@RestController
@RequestMapping("/customer")
public class CustomerController {

    private final CustomerService customerService;

    @Autowired
    public CustomerController(CustomerService customerService) {
        this.customerService = customerService;
    }
```

```
    // RESTful endpoint methods...
}
```

### 9.2.2 Required Endpoints

#### 9.2.2.1 Add Customer

Post request to add a new customer.

```
@PostMapping("/addCustomer")
public Customer addCustomer(@RequestBody Customer customer);
```

#### 9.2.2.2 Get All Customers

Get request to retrieve all customers.

```
@GetMapping("/allCustomers")
public List<Customer> getAllCustomers();
```

#### 9.2.2.3 Get Specific Customer

Get request to retrieve a customer by ID.

```
@GetMapping("/{id}")
public Customer getCustomerById(@PathVariable Long id);
```

#### 9.2.2.4 Update Customer

Put request to update customer information.

```
@PutMapping("/update/{id}")
public Customer updateCustomer(@PathVariable Long id, @RequestBody Customer customer);
```

#### 9.2.2.5 Delete Customer

Delete request to remove a customer.

```
@DeleteMapping("/delete/{id}")
public String deleteCustomer(@PathVariable Long id);
```

#### 9.2.2.6 Find Customers By Email Domain

Get request to retrieve customers whose email address ends with a specific domain.

```
@GetMapping("/findByEmailDomain")
public List<Customer> findCustomersByEmailDomain(@RequestParam String domain);
```

#### 9.2.2.7 Find Customers By Phone Prefix

Get request to retrieve customers whose phone number starts with a specific prefix.

```
@GetMapping("/findByPhonePrefix")
```

```
public List<Customer> findCustomersByPhonePrefix(@RequestParam String prefix);
```

---

## 9.3 TripController

To handle endpoints related to the **Trip** entity.

### 9.3.1 Class Definition

```
@RestController
@RequestMapping("/trip")
public class TripController {

    private final TripService tripService;

    @Autowired
    public TripController(TripService tripService) {
        this.tripService = tripService;
    }

    // RESTful endpoint methods...
}
```

### 9.3.2 Required Endpoints

#### 9.3.2.1 Add Trip

Post request to schedule a new trip.

```
@PostMapping("/addTrip")
public Trip addTrip(@RequestBody Trip trip);
```

#### 9.3.2.2 Get All Trips

Get request to retrieve all trips.

```
@GetMapping("/allTrips")
public List<Trip> getAllTrips();
```

#### 9.3.2.3 Get Specific Trip

Get request to retrieve a trip by ID.

```
@GetMapping("/{id}")
public Trip getTripById(@PathVariable Long id);
```

#### 9.3.2.4 Update Trip

Put request to update the details of a trip.

```
@PutMapping("/update/{id}")
public Trip updateTrip(@PathVariable Long id, @RequestBody Trip trip);
```

#### 9.3.2.5 Delete Trip

Delete request to cancel a trip.

```
@DeleteMapping("/delete/{id}")
public String deleteTrip(@PathVariable Long id);
```

Updated Method

#### 9.3.2.6 Find Trips Within a Date Range

Get request to retrieve trips that occurred between two dates.

```
@GetMapping("/findByDateRange")
public List<Trip> findTripsWithinDateRange(@RequestParam LocalDateTime startDate, @RequestParam
    LocalDateTime endDate);
```

#### 9.3.2.7 Find Trips By Captain ID

Get request to retrieve trips associated with a specific captain.

```
@GetMapping("/findByCaptainId")
public List<Trip> findTripsByCaptainId(@RequestParam Long captainId);
```

---

## 9.4 PaymentController

To handle endpoints related to the **Payment** entity.

### 9.4.1 Class Definition

```
@RestController
@RequestMapping("/payment")
public class PaymentController {

    private final PaymentService paymentService;

    @Autowired
    public PaymentController(PaymentService paymentService) {
        this.paymentService = paymentService;
    }

    // RESTful endpoint methods...
}
```

### 9.4.2 Required Endpoints

#### 9.4.2.1 Add Payment

Post request to record a new payment.

```
@PostMapping("/addPayment")
public Payment addPayment(@RequestBody Payment payment);
```

#### 9.4.2.2 Get All Payments

Get request to retrieve all payment records.

```
@GetMapping("/allPayments")
public List<Payment> getAllPayments();
```

#### 9.4.2.3 Get Specific Payment

Get request to retrieve a payment by ID.

```
@GetMapping("/{id}")
public Payment getPaymentById(@PathVariable Long id);
```

#### 9.4.2.4 Update Payment

Put request to update payment details.

```
@PutMapping("/update/{id}")
public Payment updatePayment(@PathVariable Long id, @RequestBody Payment payment);
```

#### 9.4.2.5 Delete Payment

Delete request to remove a payment.

```
@DeleteMapping("/delete/{id}")
public String deletePayment(@PathVariable Long id);
```

#### 9.4.2.6 Find Payments By Trip ID

Get request to retrieve payments associated with a specific trip ID.

```
@GetMapping("/findByTripId")
public List<Payment> findPaymentsByTripId(@RequestParam Long tripId);
```

#### 9.4.2.7 Find Payments With an Amount Greater Than a Threshold

Get request to retrieve payments where the amount exceeds a given threshold.

```
@GetMapping("/findByAmountThreshold")
public List<Payment> findPaymentsWithAmountGreaterThan(@RequestParam Double threshold);
```

---

## 9.5 RatingController

To handle the endpoints related to the **Rating** entity (stored in MongoDB).

### 9.5.1 Class Definition

```
@RestController
@RequestMapping("/rating")
public class RatingController {
```

```
    private final RatingService ratingService;

    @Autowired
    public RatingController(RatingService ratingService) {
        this.ratingService = ratingService;
    }

    // RESTful endpoint methods...
}
```

### 9.5.2 Required Endpoints

#### 9.5.2.1 Add Rating

Post request to add a new rating document.

```
@PostMapping("/addRating")
public Rating addRating(@RequestBody Rating rating);
```

#### 9.5.2.2 Update Rating

Put request to update a rating's score or comments.

```
@PutMapping("/update/{id}")
public Rating updateRating(@PathVariable String id, @RequestBody Rating updatedRating);
```

#### 9.5.2.3 Delete Rating

Delete request to remove a rating by its ID.

```
@DeleteMapping("/delete/{id}")
public String deleteRating(@PathVariable String id);
```

#### 9.5.2.4 Find Ratings For a Specific Entity

Get request to retrieve ratings linked to a specific entity by ID and type.

```
@GetMapping("/findByEntity")
public List<Rating> findRatingsByEntity(@RequestParam Long entityId, @RequestParam String
    entityType);
```

#### 9.5.2.5 Find Ratings Above a Specific Value

Get request to retrieve ratings with a score greater than a specific threshold.

```
@GetMapping("/findAboveScore")
public List<Rating> findRatingsAboveScore(@RequestParam int minScore);
```

# 10 Docker

A `docker-compose.yml` file should be provided to orchestrate the application along with its databases:

- PostgreSQL for relational data.

- MongoDB for NoSQL data.
- Your Spring Boot application container.

Ensure that the Dockerfile and environment variables reference the correct paths for configuration files and data mounting.

---

# 11 Test Cases

The public test cases file will be provided in the middle of the week to test your project.

---

# 12 Submission

Create a repository for your Project with separate branches for each team member with the Main branch for integrating your work. Your repository name should be Mini2 - Your Team Number - your Team name (e.g. ***Mini2-100-Random_ 01***). Then add ****Scalable-Submissions**** OR ****Scalable2025@gmail.com**** as colaborator to your repository. Your history of commits will be monitored as an indication for the participation for all the team members so make sure you divide the work equally among you. If a branch was missing for a team member, it will result in a ***zero*** for this team member.