Internet Applications Design and Implementation

(Lecture 7: Q&A)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Software Architecture

- What? Why? How?
- Why are Patterns important
- Why are Frameworks important

Services and MicroServices

- Main characteristics, features and motivations
- Exercise on modelling of MS architectures
- Why is OpenAPI important, what are its main features?
- Exercise on the definition of interfaces between services in practice

Architectural Patterns, Layered Architecture

- Purpose of each layered, rules to place software pieces, dependencies
- Data representation, DTO, DAO objects

REST architectural style

- Maturity Levels
- Properties of REST
- How to design an API with resources and subresources

Data Abstraction, Component Based-Programming

- Definition of JPA interfaces
- Custom Queries
- Pre-fetching

Assembling Applications, Testing with Mocks

- Dependencies
- Beans (@Service, @Component, @Bean)
- Mocks to test a component or a service (MVC, Repositories, Services)

Security of Internet Applications

- Basic Spring Security Configuration
- Access Control Models
- Information Flow
- Model-based access control (related to Attribute Access Control)
- How to implement a security (access control) policy that depends on the model. PreAuthorize + Service, PostAuthorize, PreFilter, PostFilter
- https://docs.spring.io/spring-security/reference/servlet/authorization/method-security.html
- JWT Tokens, Capabilities in Services

Sample Scenario

- Consider a headless server system for the management of train seat reservations in a railway company. So, the users of the system are either passengers or the railway inspectors. A passenger can list the trains that are available on a given date, the stations through which the trains go, and the seats that are available from a given starting station to a finishing station. In this scenario, the name train is used to signify a train trip, not the vehicle itself.
- The operations you need to account for are as follows: A passenger can reserve a seat in a train, indicating the train and the starting and finishing station. A passenger can also list their reservations and cancel or edit a reservation. There may be more than one passeger reserving one seat in a train, as long as they do not overlap in time (stations). A railway inspector can list all the passengers in their designated trains, they can edit the train schedule, adding or removing trains and stations.
- As further constraints to the systems, consider the following: A passenger can only see their own present and past reservations, and they can only cancel or edit a reservation up to one hour before the train leaves the starting station. Railway inspectors can edit trains up to one day before the train trip leaves.

REST

- Define a RESTful interface, Level 2 in the Richardson maturity scale, that defines the operations on trains and reservations. List all the endpoints by means of a Kotlin interface using data classes for DTO objects. No annotations and extra code is needed. Remember to use different levels of detail in different endpoints.
- Design the best EXTERNAL representation possible for the resources in the system. Note that the internal representation in the database is not necessarily the best to use in an API.

```
// General

GET /trains?date=XXX --> {id:number,date:Date,from:string,to:string,path:string[]}[]
GET /trains/{id}/path --> string[]
GET /trains/{id}/freeseats?from=XXX&to=XXX --> {id:number}[]

// passengers

POST /passengers/{username}/trips <<-- {train:number, from:string, to:string} -->>
number
GET /passengers/{username}/trips -->> {train:number, from:string, to:string}[]
DELETE /passengers/{username}/trips/{id}
PUT /passengers/{username}/trips/{id} <<-- {from:string, to:string}

// inspectors

GET /train/{id}/seats -->> {seat:number, passenger:string, from:string, to:string}[]
PUT /train/{id} <<-- {schedule:Date}
```

```
data class TrainDTO(val id:Long, val date:Date, val from:String, val to:String, val
path:String[])

data class SeatDTO(val id:Long)

data class TicketDTO{val id:Long, val train:Long, val from:String, val to:String}

data class TicketWIdDTO{val id:Long, val train:Long, val from:String, val to:String}

data class TicketUpdateDTO(val from:String, val to:String)

data class OccupancyDTO(val seat:Long, val passenger:String, val from:String, val
to:String)

data class ScheduleDTO(schedule:Date)
```

```
// General

GET /trains?date=XXX --> List<TrainDTO>
GET /trains/{id}/path --> List<String>
GET /trains/{id}/freeseats?from=XXX&to=XXX --> List<SeatDTO>

// passengers

POST /passengers/{username}/trips <<-- TicketDTO -->> Long
GET /passengers/{username}/trips -->> List<TicketWIdDTO>
DELETE /passengers/{username}/trips/{id}
PUT /passengers/{username}/trips/{id} <<-- TicketUpdateDTO

// inspectors

GET /trains/{id}/seats -->> List<OccupancyDTO>
PUT /trains/{id} <<-- ScheduleDTO
```

```
@RequestMapping("/api")
interface TrainController {

    // GET /trains?date=XXX
    @GetMapping("/trains")
    fun getTrainsByDate(@RequestParam date: String): List<TrainDTO>

    // GET /trains/{id}/path
    @GetMapping("/trains/{id}/path")
    fun getTrainPath(@PathVariable id: Long): List<String>

    // GET /trains/{id}/freeseats?from=XXX&to=XXX
    @GetMapping("/trains/{id}/freeseats")
    fun getFreeSeats(@PathVariable id: Long, @RequestParam from: String, @RequestParam to: String ): List<SeatDTO>

    // POST /passengers/{username}/trips
    @PostMapping("/passengers/{username}/trips")
    fun createPassengerTrip(@PathVariable username: String, @RequestBody ticketDTO: TicketDTO ): Long

    // GET /passengers/{username}/trips
    @GetMapping("/passengers/{username}/trips")
    fun getPassengerTrips(@PathVariable username: String): List<TicketWIdDTO>

    // DELETE /passengers/{username}/trips/{id}
    @DeleteMapping("/passengers/{username}/trips/{id}")
    fun deletePassengerTrip( @PathVariable username: String, @PathVariable id: Long ): Void

    // PUT /passengers/{username}/trips/{id}
    @PutMapping("/passengers/{username}/trips/{id}")
    fun updatePassengerTrip( @PathVariable username: String, @PathVariable id: Long, @RequestBody ticketUpdateDTO: TicketUpdateDTO ): Void

    // GET /trains/{id}/seats
    @GetMapping("/train/{id}/seats")
    fun getTrainOccupancy(@PathVariable id: Long): List<OccupancyDTO>

    // PUT /trains/{id}
    @PutMapping("/train/{id}")
    fun updateTrainSchedule( @PathVariable id: Long, @RequestBody scheduleDTO: ScheduleDTO ): Void
}
```

Datamodel

- Define the JPA (data) classes for the system described above and relate them correctly using JPA annotations. Design the best INTERNAL representation possible for the resources in the database. Note that the external representation in the API is not necessarily the best to use in the database.

Security

- Define two security policies, annotations, and corresponding services that regulate the access for reading, and also for changing reservations in the scenario above.

A passenger can only see their own present and past reservations, and they can only cancel or edit a reservation up to one hour before the train leaves the starting station. Railway inspectors can edit trains up to one day before the train trip leaves.