Overview

March 31

Session 1. Sharing experiences and QnA

Session 2. Spring framework fundamentals

Session 3. Installing required tools and bootstrapping hotel booking service

April 1

Session 4. Restful APIs and working with UI

Session 5. Data consistency with Spring

Session 1

Sharing experiences and QnA

About me


```
{
    "name": "Mashhur Sattorov",
    "profession": "Senior Software Engineer",
    "education": {
        "bachelor": "Tashkent IT Univ.",
        "master": "Mokwon Univ."
    },
    "workplaces": {
        "current": "Elastic (aka ElasticSearch)",
        "past": [
            "Amazon, Canada",
            "YogiOtte, South Korea",
            "TMON (Groupon), South Korea",
            "Unitech, South Korea"
        ]
    }
}
```

About me : business domain scope

- Defence Systems and Airport Safety
- E-Commerce applications:
  - tmon.co.kr: chatting platform, card and order service
  - goodchoice.kr: coupon microservice
  - amazon.com: vendor management service
- Data ingestion
  - Elasticsearch, Logstash, Elastic Agent, Kibana solutions

About me: Tech scope

Embedded Systems - 4 years & more

- FPGA (Verilog), Kernel level programming (C)

Building Microservices - 8 years & more

- Java (Spring), NodeJS, PHP Laravel

Distributed Databases - 1+ year

- Data ingestion with Elasticsearch and Logstash

Session 2

Spring framework fundamentals

Content

1. What is a Spring framework?
2. Why should I use framework?
3. Spring projects

1. Introduction

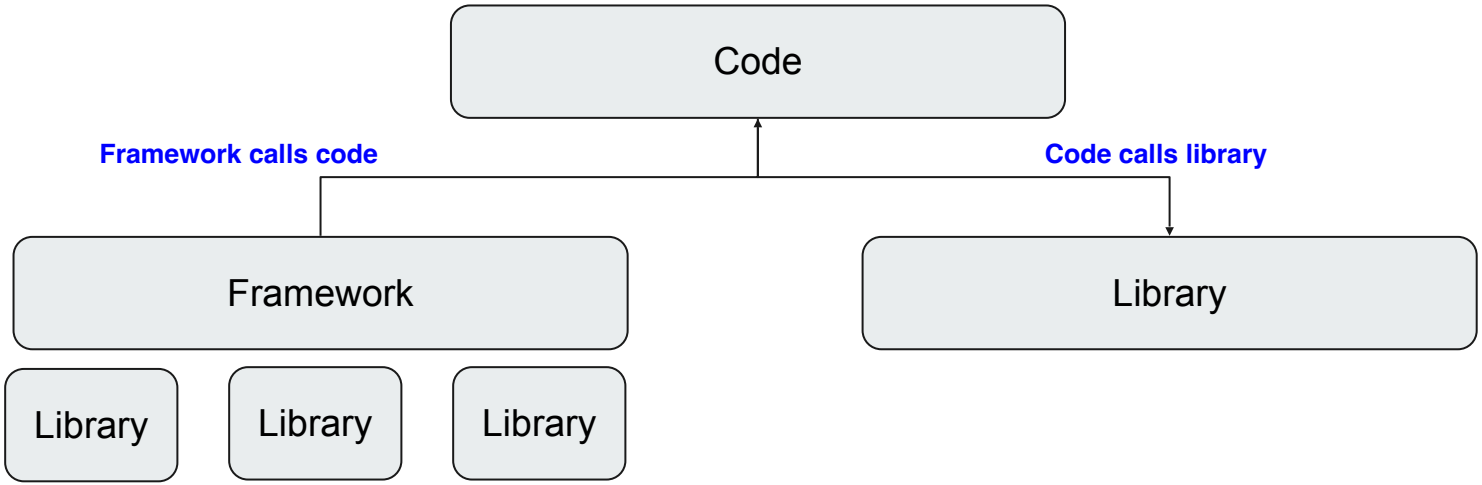
1. What is a Spring framework?
2. Why should I use framework?
3. Spring projects
4. What is IoC (Inversion of Control)?
5. What is DI (Dependency Injection)?
6. What are Beans?
7. Context
8. IoC Container

What is a Spring framework? - Intro

Spring is an **open source**, a lightweight Java framework which enhances Java EE features when developing Java applications.

1. Build *any* (batch processing, cloud, streaming, reactive/non-blocking) web applications fast;
2. Make developer life easier, focus only on business logics rather than managing modules;
    a. In ~2000 year, every developer had their own module (ex: *JavaThreadManager*)
3. Open source with large feedback supportive dev community, release iteration is smooth

What is a Spring framework? - Framework vs Library



Framework contains list of libraries

What is a Spring framework? - Java family clarifications

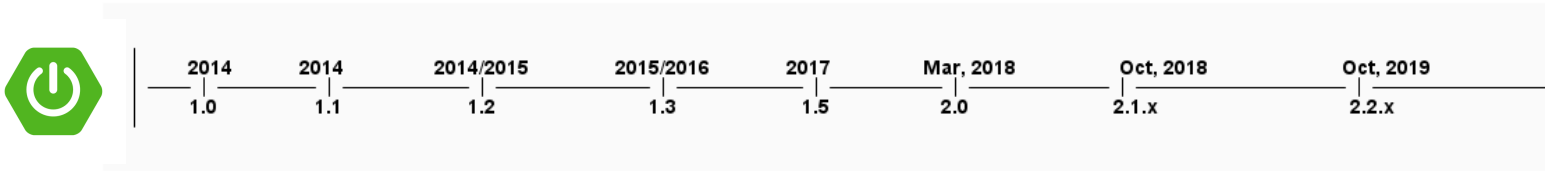
Java EE - Enterprise Edition (Java EE) is a collection of Java APIs owned by **Oracle**

Jakarta EE - open source alternative to Java EE owned by **Eclipse Foundation**

Java Spring - extends or leans on Java EE specifications (JSR) and open source developed by VMWare (who bought Tanzu, previously Spring developer group, key person: Josh Bloch)

Java is owned by Oracle.

What is a Spring framework? - Spring vs Spring boot versions



Why we should use Spring?

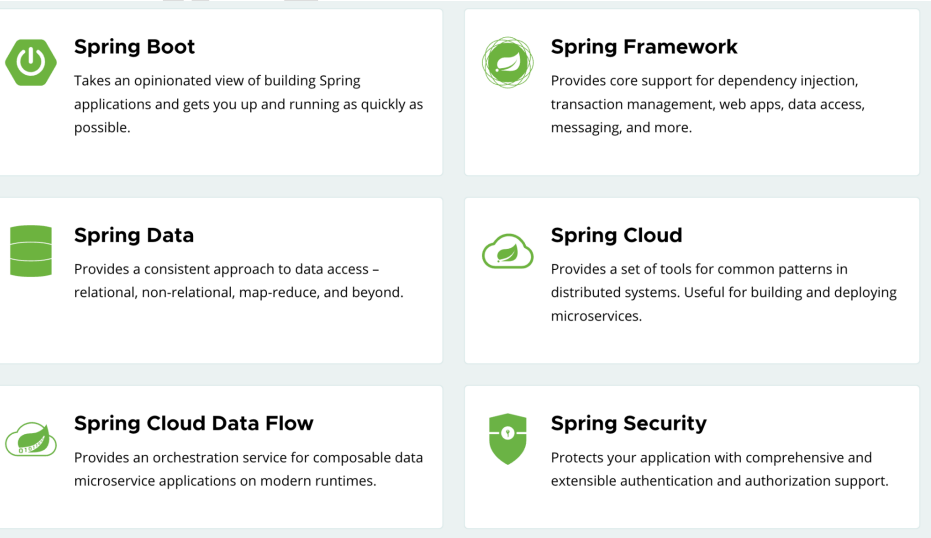
doesn't make tire &
glasses & seats

makes tire &
glasses & seats

- Focuses on quality/car preparation
- Can make more cars
- Can scale the business
- Less efforts, more results

- Takes much time
- May not get a consistent quality car
- Need to manage raw materials
- More work less result

Spring projects: spring.io/projects



Spring Boot

Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.

Spring Framework

Provides core support for dependency injection, transaction management, web apps, data access, messaging, and more.

Spring Data

Provides a consistent approach to data access – relational, non-relational, map-reduce, and beyond.

Spring Cloud

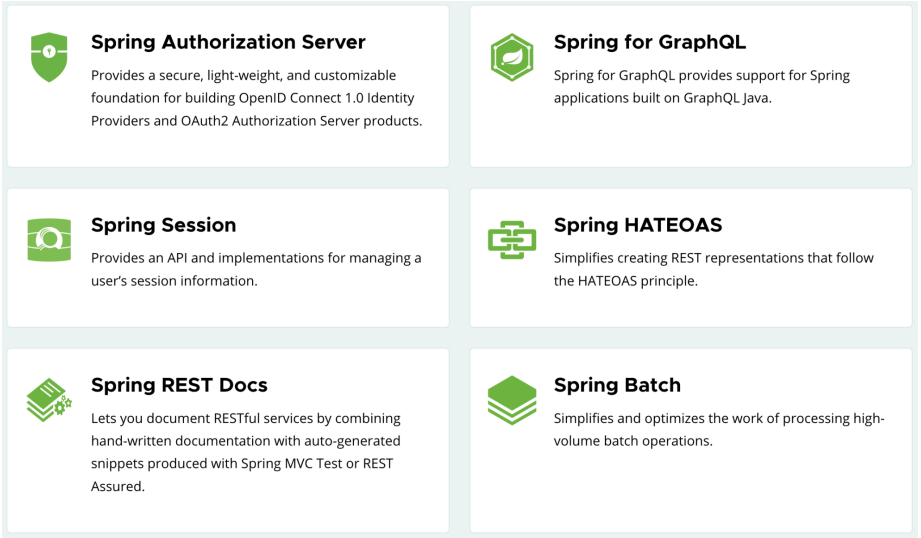
Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices.

Spring Cloud Data Flow

Provides an orchestration service for composable data microservice applications on modern runtimes.

Spring Security

Protects your application with comprehensive and extensible authentication and authorization support.



Spring Authorization Server

Provides a secure, light-weight, and customizable foundation for building OpenID Connect 1.0 Identity Providers and OAuth2 Authorization Server products.

Spring for GraphQL

Spring for GraphQL provides support for Spring applications built on GraphQL Java.

Spring Session

Provides an API and implementations for managing a user's session information.

Spring HATEOAS

Simplifies creating REST representations that follow the HATEOAS principle.

Spring REST Docs

Lets you document RESTful services by combining hand-written documentation with auto-generated snippets produced with Spring MVC Test or REST Assured.

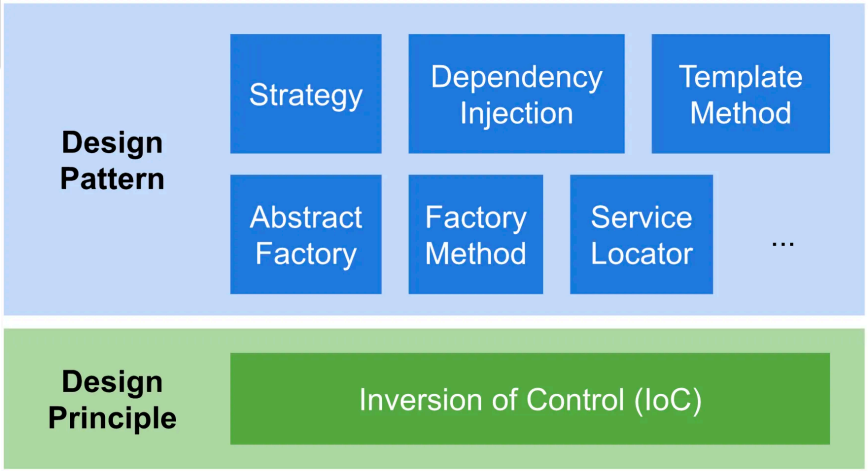
Spring Batch

Simplifies and optimizes the work of processing high-volume batch operations.

Spring Core Principles: what are IoC and DI?

IoC - Inversion of Control

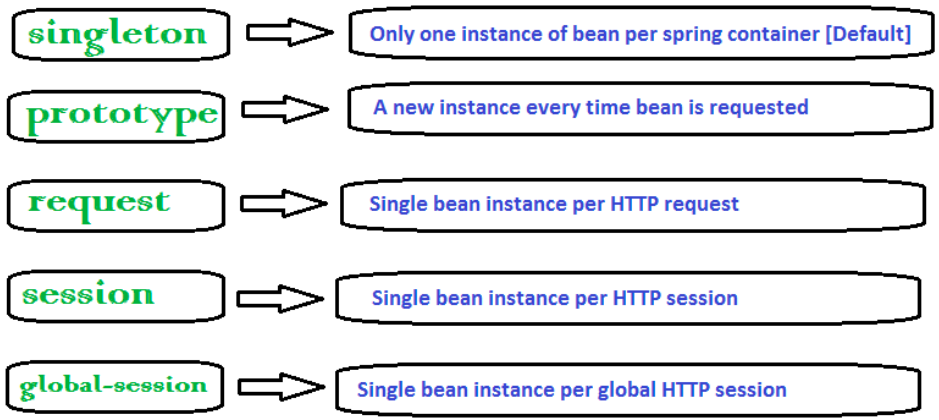
DI - Dependency Injection



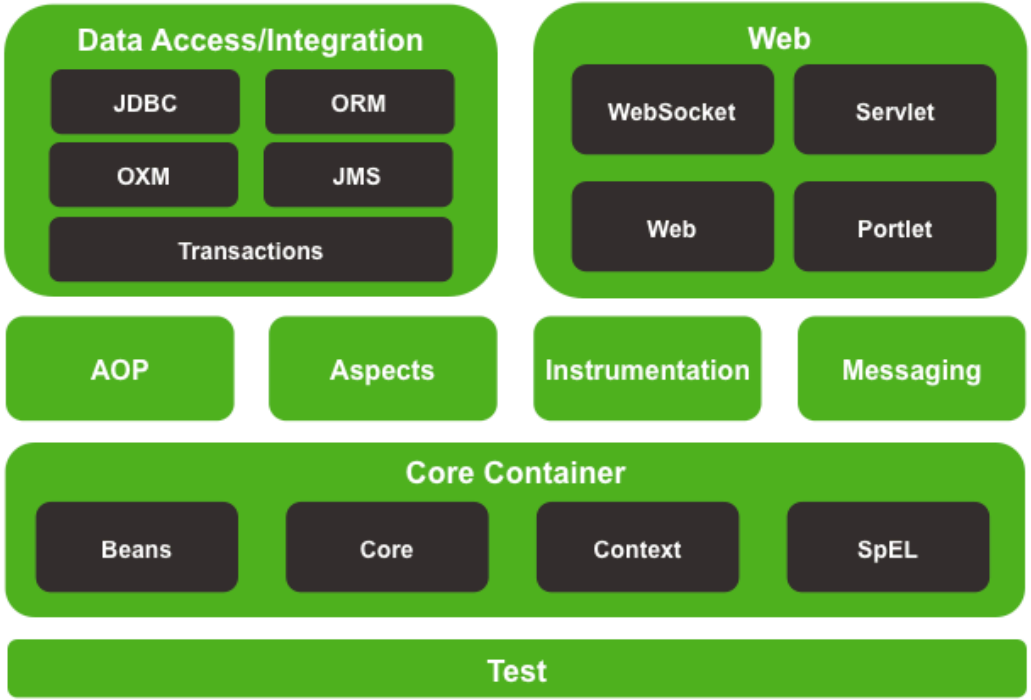
Spring Core Principles: What are Beans?

In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC *container* are called *beans*.

Beans are crated with @Bean and @Component (including stereotypes) annotations



Spring Core Principles: Spring runtime



- Spring ***contexts*** are responsible for instantiating, configuring, and assembling beans

- ***AOP*** (Aspect Oriented Programming) is a programming pattern that increases modularity by allowing the separation of the cross-cutting concern. @Around,@Before, @AfterThrowing and @AfterReturning advices

- ***SpEL*** (Spring Expression Language) Example <bean id="exampleBeanId"/>

- ***JPA*** (The Java Persistence API) is a specification of Java. It is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems.

Spring Core Principles: Spring MVC vs Spring boot

Spring MVC	Spring Boot
Spring MVC is a Model, View, and Controller based framework used to build web applications.	Spring Boot is one of the most widely used frameworks in the REST API development field. It is developed on top of the conventional spring framework. It is used to build stand-alone web spring applications.
This framework requires a lot of configurations, such as DispatcherServlet configurations and View Resolver configurations.	Spring Boot handles the configurations automatically with its Auto-configuration feature.
Every dependency needs to be specified separately for the features to run.	Spring Booth has the concept of starters, once it is added to the classpath, it will bring all the dependencies needed for developing a web application.
It does not provide powerful batch processing.	It provides powerful batch processing.

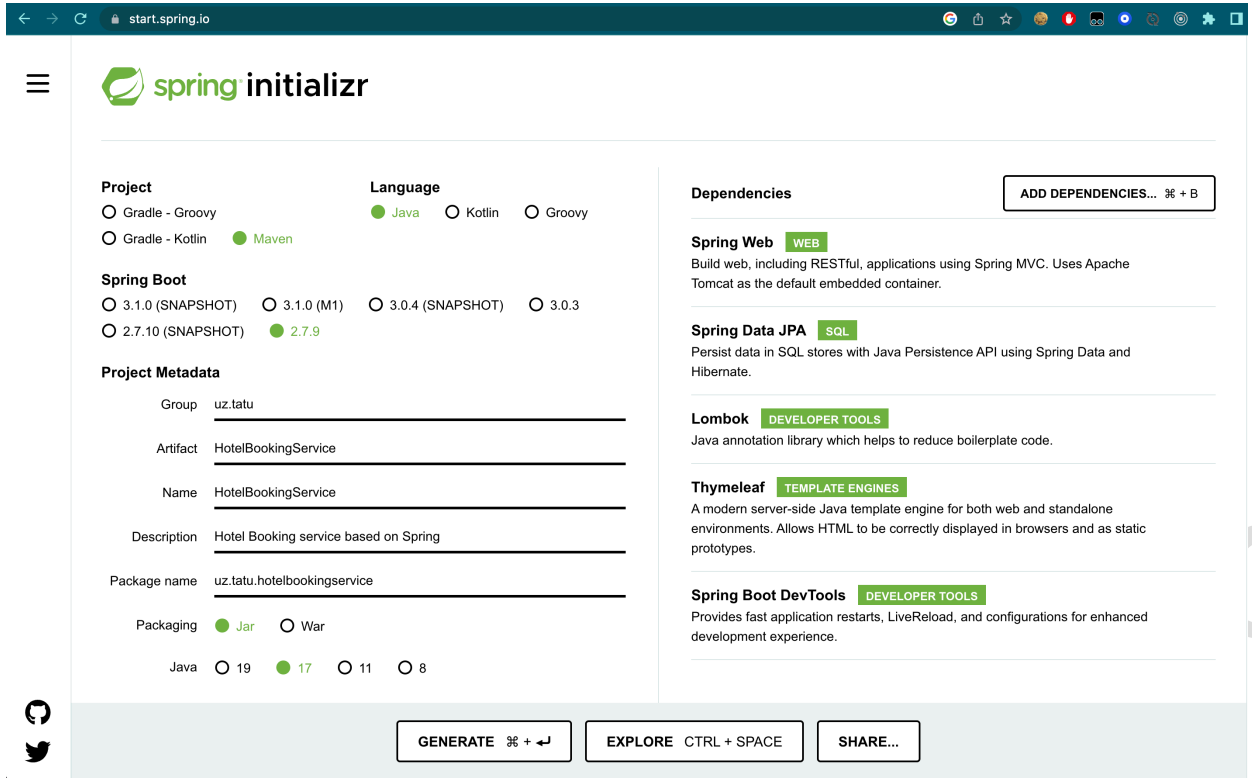
Session 3

Installing required tools
Bootstrapping hotel booking service

Required tools: Project

- Java (JDK 17): java.com/en/download/
- IntelliJ IDEA: jetbrains.com/idea/download
- MySQL: dev.mysql.com/downloads/mysql/
  - username: tatuParrot
  - password: parrotSaysHi
- MySQL Workbench: dev.mysql.com/downloads/workbench/
- Postman: postman.com/downloads
- Git (optional): github.com/git-guides/install-git

Required tools: Creating a Spring Application



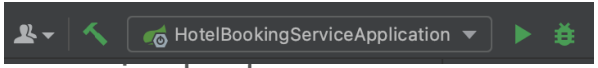
- Maven or Gradle
- Lombok
- Swagger
- Thymeleaf
- Project shared board: qtext.io/5z9s

Creating a Spring Application

- pom.xml overview

```
<dependencies>
    <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
    </dependency>
    …
</dependencies>
```

Running Spring Application

- Run the application
  - With IntelliJ IDEA 

  - With command: $mvn spring-boot:run

For now we get a following error. Let's configure DataSource to connect to MySQL (or any DB)
Change @SpringBootApplication annotation HotelBookingServiceApplication.java

ERROR: Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.

```
@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class })
```

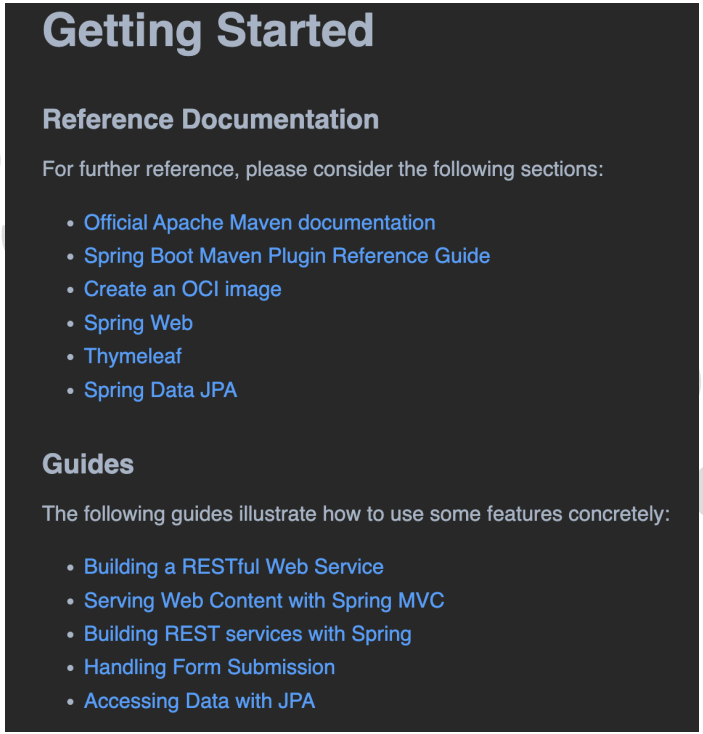
Running Spring Application

- Check if application is running

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.0.3)

2023-03-03T13:53:34.933+05:00  INFO 39473 --- [           main] u.t.h.HotelBookingServiceApplication     : Starting HotelBookingServiceApp
2023-03-03T13:53:34.934+05:00  INFO 39473 --- [           main] u.t.h.HotelBookingServiceApplication     : No active profile set, falling
2023-03-03T13:53:35.292+05:00  INFO 39473 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s)
2023-03-03T13:53:35.296+05:00  INFO 39473 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-03-03T13:53:35.297+05:00  INFO 39473 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache
2023-03-03T13:53:35.328+05:00  INFO 39473 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded We
2023-03-03T13:53:35.328+05:00  INFO 39473 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: ini
2023-03-03T13:53:35.443+05:00  WARN 39473 --- [           main] ion$DefaultTemplateResolverConfiguration : Cannot find template location:
2023-03-03T13:53:35.474+05:00  INFO 39473 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080
2023-03-03T13:53:35.479+05:00  INFO 39473 --- [           main] u.t.h.HotelBookingServiceApplication     : Started HotelBookingServiceAppli
```

- Go to localhost:8080

Running Spring Application: Help.md overview



Getting Started

Reference Documentation

For further reference, please consider the following sections:

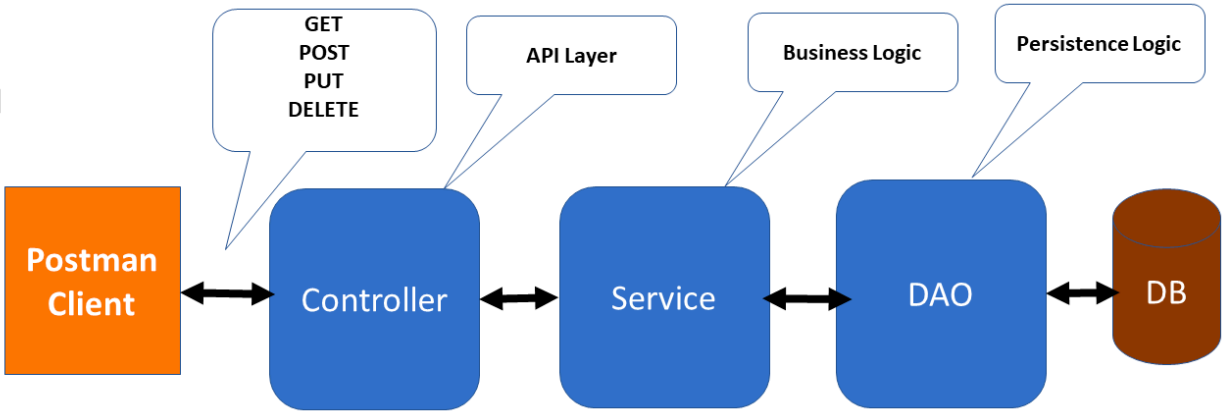
- Official Apache Maven documentation
- Spring Boot Maven Plugin Reference Guide
- Create an OCI image
- Spring Web
- Thymeleaf
- Spring Data JPA

Guides

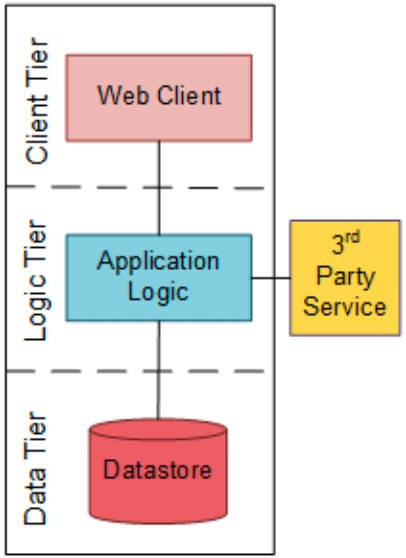
The following guides illustrate how to use some features concretely:

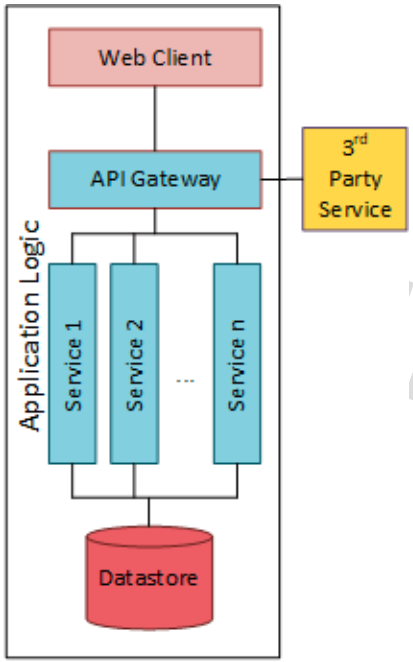
- Building a RESTful Web Service
- Serving Web Content with Spring MVC
- Building REST services with Spring
- Handling Form Submission
- Accessing Data with JPA

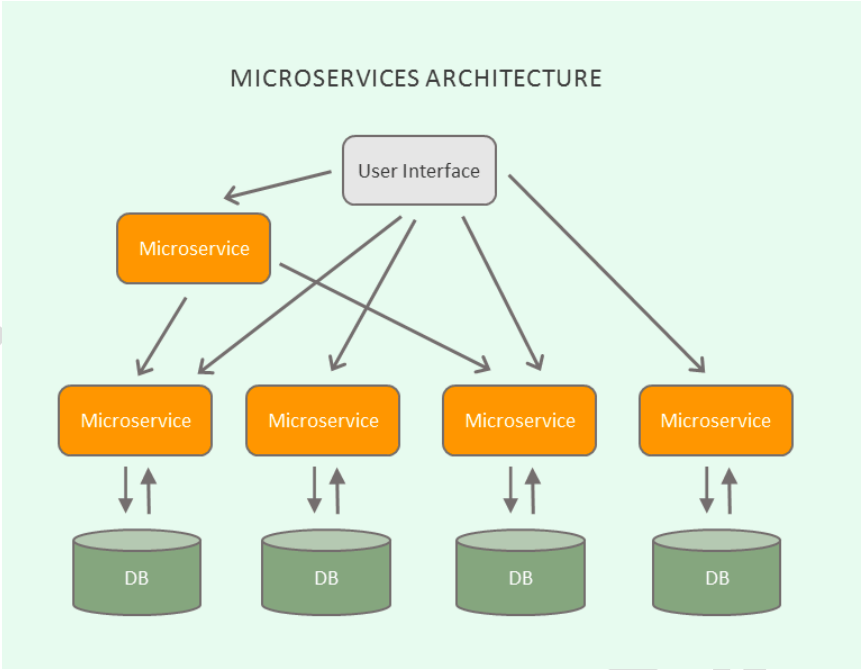
Hotel Booking Service Architecture overview



Monolith vs Microservice







Hotel Booking Service: Controller

- Create a `controller` folder under `hotelBookingService` root folder
- Create a `HotelBookingServiceRestController.java` and `HotelBookingServiceViewController.java` files
- Run application

HotelBookingServiceRestController.java

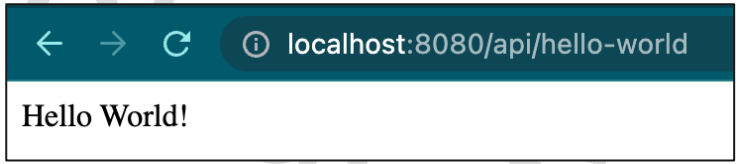
```java
package uz.tatu.hotelbookingservice.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api")
public class HotelBookingServiceRestController {

    @GetMapping("hello-world")
    public String helloWorld() {
        return "Hello World!";
    }
}
```

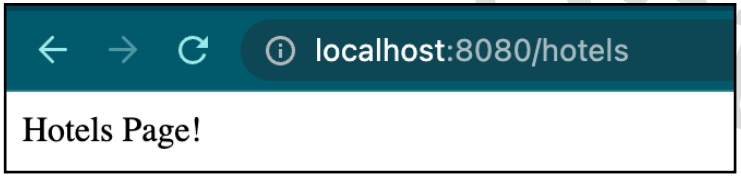
HotelBookingServiceViewController.java

```java
package uz.tatu.hotelbookingservice.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HotelBookingServiceViewController {

    @GetMapping("/hotels")
    public ModelAndView hotels() {
        return new ModelAndView( viewName: "hotels");
    }
}
```

Hotel Booking Service: Controller

- Create `hotels.html` under resources/templates folder and add “Hotels Page!” In body section
- Access localhost:8080/api/hello-world
- Access localhost:8080/hotels





Session 4

Restful APIs and working with UI

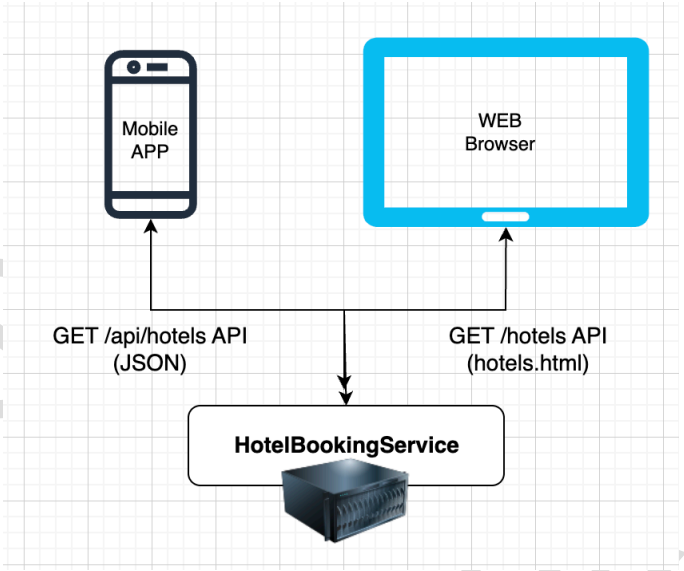
What are APIs (Application Programming Interface)?

- Add /hotels API to return hotel list to Rest Controller

```
@GetMapping("/hotels")
public List<String> hotels() {
    List<String> hotels = new ArrayList<>();
    hotels.add("Inter Continental");
    hotels.add("Sheraton");
    return hotels;
}
```

- Update hotels.html

```
<body>
  Hotels Page!
    <ul>
        <li>Inter Continental</li>
        <li>Sheraton</li>
    </ul>
</body>
```



What are Restful APIs?

REST stands for Representational State Transfer.

RESTful API is an interface that two computer systems use to exchange information over the internet.

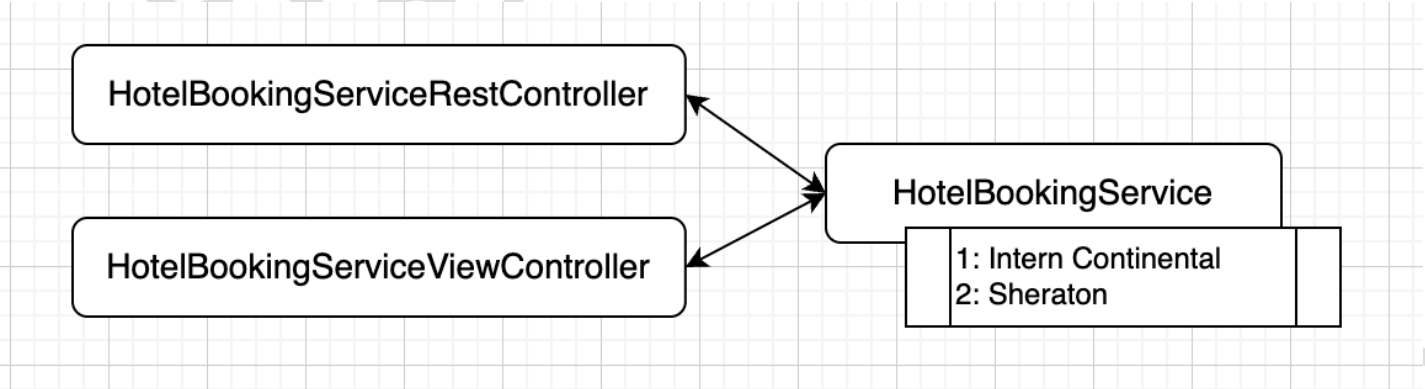
Method	Description
GET	Retrieve information about the REST API resource
POST	Create a REST API resource
PUT	Update a REST API resource
DELETE	Delete a REST API resource or related component

Hotel Booking Service Restful APIs

Method	Description
GET - /hotels	Get list of available hotels
POST - /hotels	Create list of hotels
PUT - /hotels/{id}	Update {id} hotel resource
DELETE - /hotels/{id}	Delete {id} hotel resource

Hotel Booking Business Logic

Service logic should contain
- Available hotels;
- Book (or reserve);
- Add/Update/Delete



Hotel Booking Business Logic

- Create `service` folder under `hotelbookingservice` package
- Create HotelBookingService.java
- Add following logic:

Map is not SAFE, what we can use instead?

```
@Service
public class HotelBookingService {

    // Map is not safe?
    private final Map<Integer, String> hotels = new HashMap<>();

    {
        hotels.put(1, "Inter Continental");
        hotels.put(2, "Sheraton");
    }

    public Map<Integer, String> availableHotels() {
        return hotels;
    }
}
```

Call Hotel Booking service layer

- Inject and call `HotelBookingService` in Rest Controller
- Question: why shouldn't we just use field auto wire, like
  - @Autowired private HotelBookingService hotelBookingService;

```
private final HotelBookingService hotelBookingService;

@Autowired
public HotelBookingServiceRestController(HotelBookingService hotelBookingService) {
    this.hotelBookingService = hotelBookingService;
}

@GetMapping("/hotels")
public Map<Integer, String> hotels() {
    return hotelBookingService.availableHotels();
}
```

Hotel CRUD operations: Get

- Get hotel

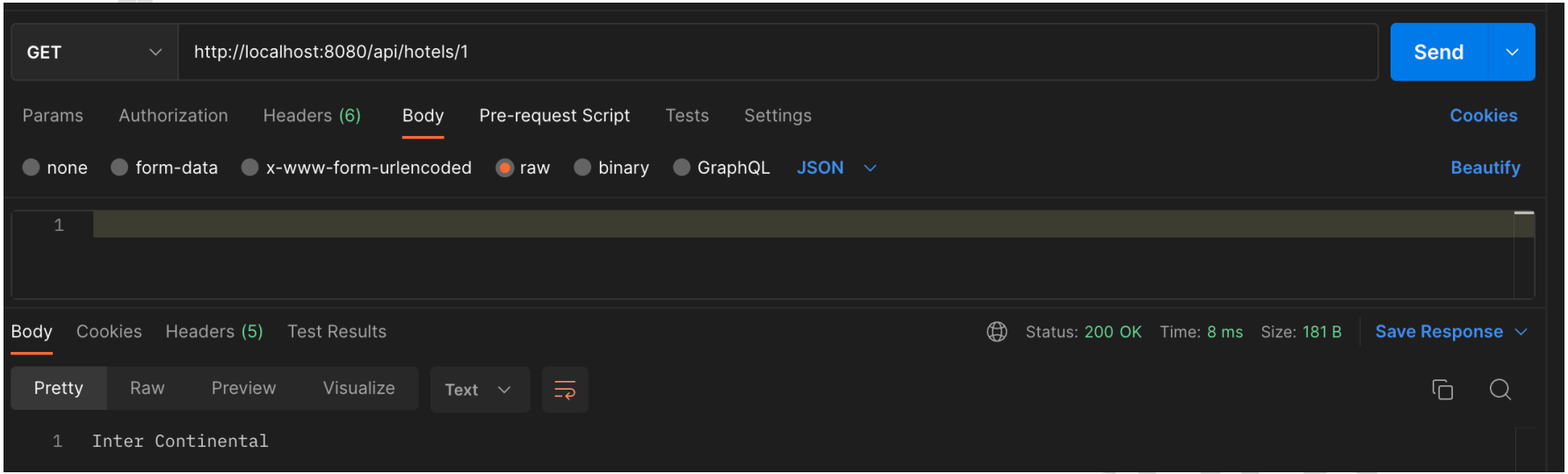
// HotelBookingServiceRestController.java

```
@GetMapping("/hotels/{id}")
public String hotel(@PathVariable int id) {
    return hotelBookingService.hotel(id);
}
```

// HotelBookingService.java

```
public String hotel(int id) {
    if (Objects.isNull(hotels.get(id))) {
        throw new NoHotelFoundException(id);
    }

    return this.hotels.get(id);
}
```

Hotel CRUD operations: Get

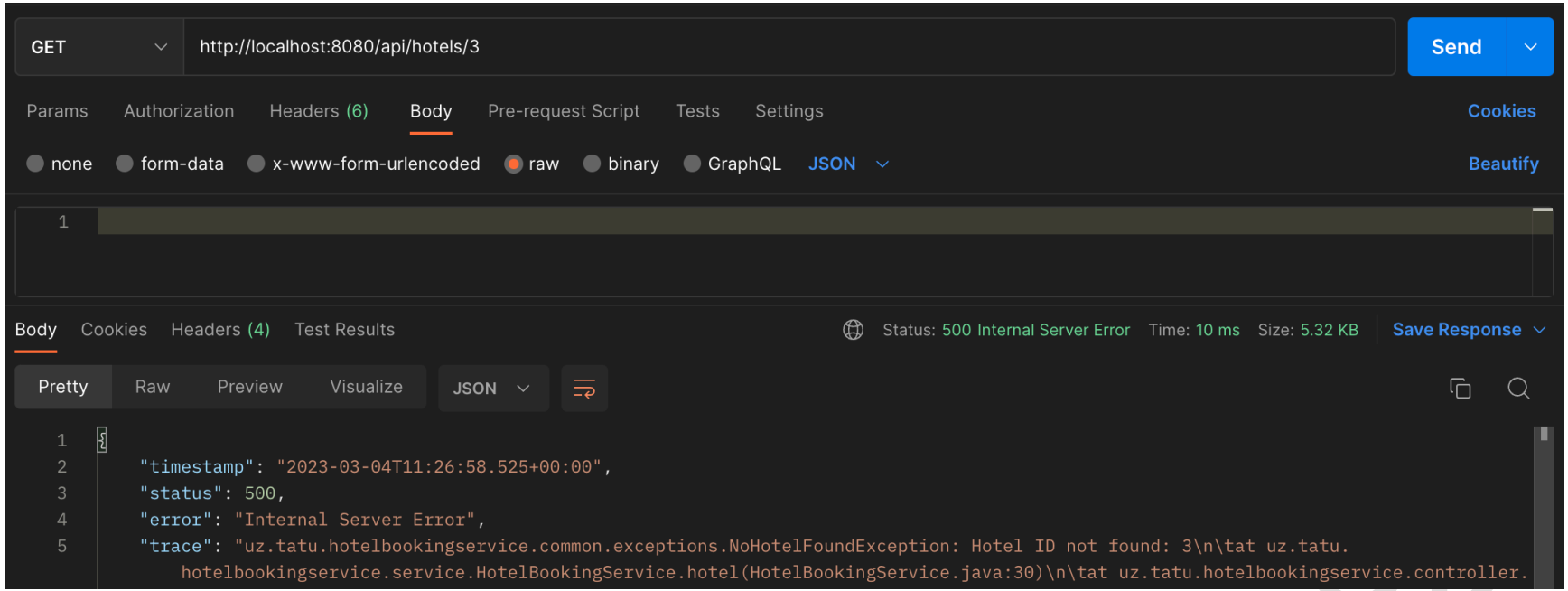


Hotel CRUD operations: Get, NoHotelFoundException

- Create a `common` folder under `hotelbookingservice` folder and create `exceptions` folder under common
- Create `NoHotelFoundException.java`

```
package uz.tatu.hotelbookingservice.common.exceptions;

public class NoHotelFoundException extends RuntimeException {

    public NoHotelFoundException(int hotelId) {
        super("Hotel ID not found: " + hotelId);
    }
}
```

Hotel CRUD operations: Get, NoHotelFoundException



Hotel CRUD operations: Create

- Create hotels

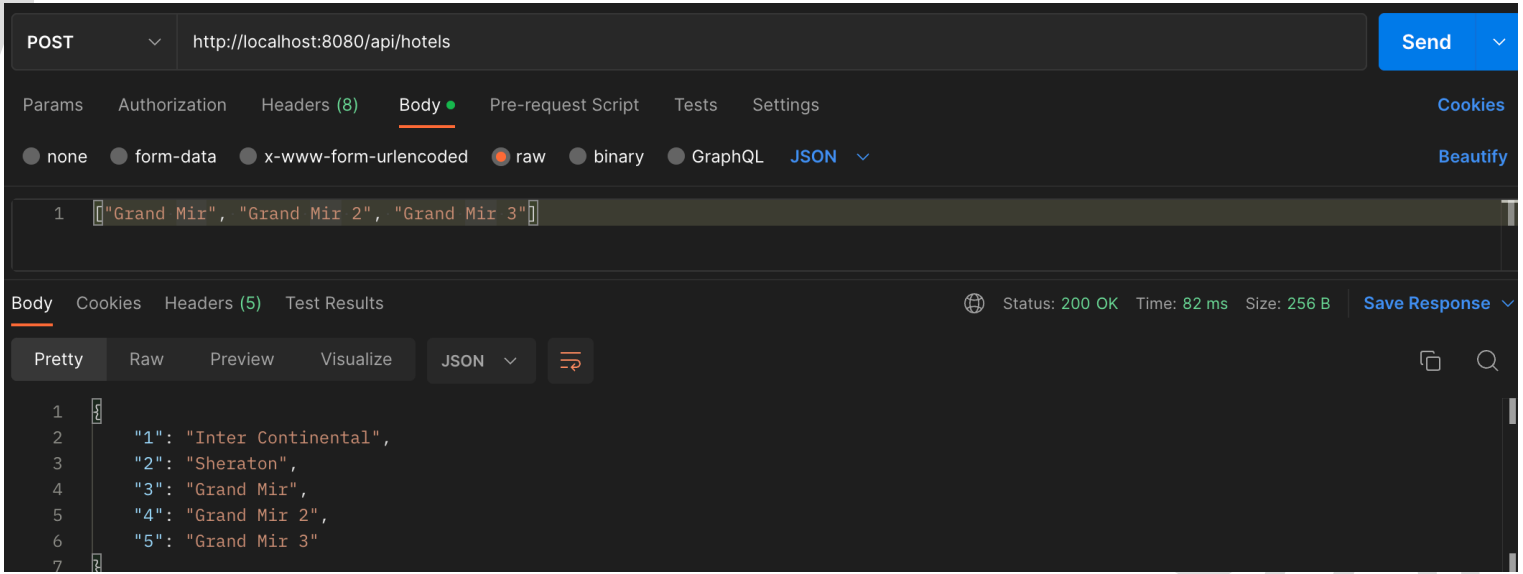
// HotelBookingServiceRestController.java

```
@PostMapping("/hotels")
public Map<Integer, String> hotels(@NonNull List<String> hotels) {
    return hotelBookingService.addHotels(hotels);
}
```

// HotelBookingService.java

```
public Map<Integer, String> addHotels(List<String> hotels) {
    hotels.forEach((name) -> this.hotels.put(this.hotels.size() + 1, name));
    return this.hotels;
}
```

Hotel CRUD operations: Create



Hotel CRUD operations: Update

- Update hotel

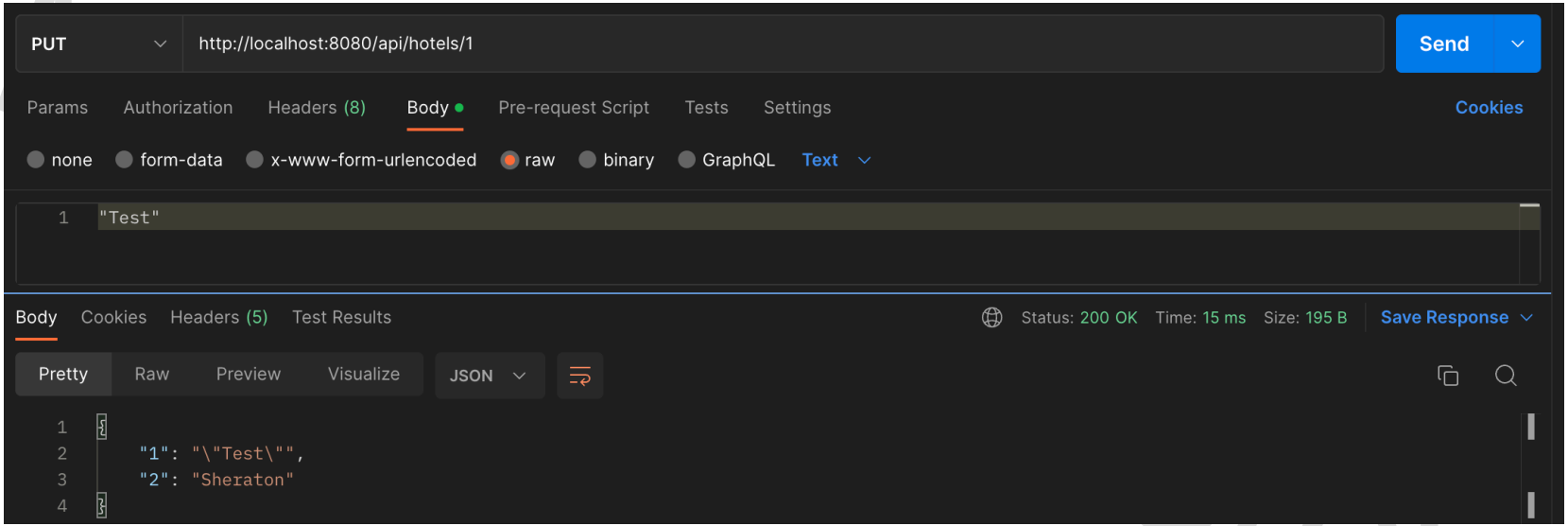
// HotelBookingServiceRestController.java

```
@PutMapping("/hotels/{id}")
public Map<Integer, String> updateHotel(@PathVariable int id,
                                        @RequestBody @NonNull String hotelName) {
    return hotelBookingService.updateHotel(id, hotelName);
}
```

// HotelBookingService.java

```
public Map<Integer, String> updateHotel(int id, String name) {
    if (Objects.isNull(hotels.get(id))) {
        throw new NoHotelFoundException(id);
    }

    hotels.put(id, name);
    return this.hotels;
}
```

Hotel CRUD operations: Update



Hotel CRUD operations: Delete

- Delete hotel

// HotelBookingServiceRestController.java

```
@DeleteMapping("/hotels/{id}")
public void deleteHotel(@PathVariable int id) {
    hotelBookingService.deleteHotel(id);
}
```

// HotelBookingService.java

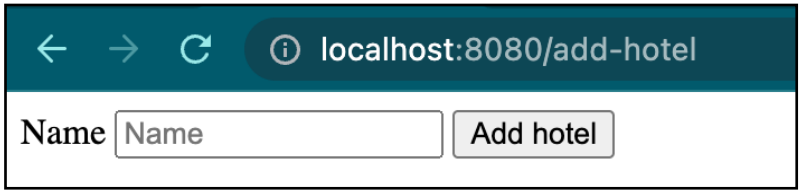
```
public void deleteHotel(int id) {
    if (Objects.isNull(hotels.get(id))) {
        throw new NoHotelFoundException(id);
    }

    hotels.remove(id);
}
```

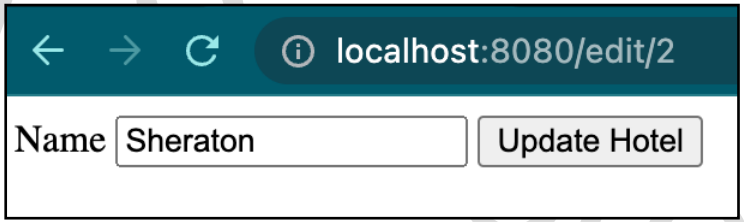
Serving UI with Spring

1. Thymeleaf

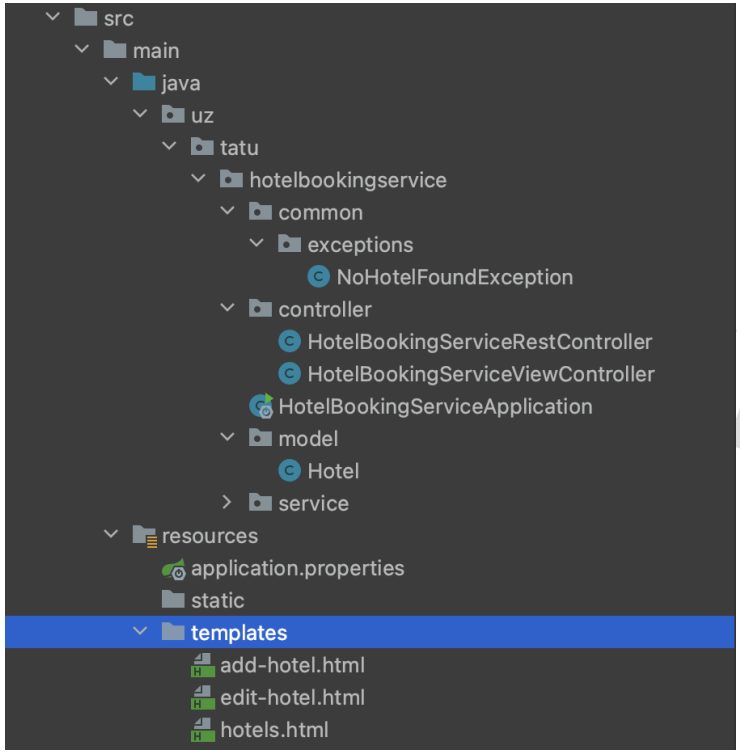
2. How do backend, frontend (SPA) and mobile work all together?

[Thymeleaf] The UIs we are going to build





Current project structure



controller ->
- HotelBookingServiceViewController.java

model ->
- Hotel.java

resources/templates ->
- add-hotel.html
- edit-hotel.html
- hotels.html

Hotel.java model

```java
package uz.tatu.hotelbookingservice.model;

//import lombok.Builder;
import lombok.Getter;
import lombok.Setter;

// TODO: we will cover when we talk about Lombok
//       @Builder(toBuilder = true)
@Getter
@Setter
public class Hotel {

    public int id;

    public String name;
}
```

HotelBookingServiceViewController.java

```java
package uz.tatu.hotelbookingservice.controller;

import lombok.NonNull;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import uz.tatu.hotelbookingservice.model.Hotel;
import uz.tatu.hotelbookingservice.service.HotelBookingService;

import java.util.ArrayList;
import java.util.List;

@Controller
public class HotelBookingServiceViewController {

    private final HotelBookingService hotelBookingService;

    @Autowired
    public HotelBookingServiceViewController(HotelBookingService hotelBookingService) {
        this.hotelBookingService = hotelBookingService;
    }
```

HotelBookingServiceViewController.java (continue)

```java
@GetMapping("/hotels")
public ModelAndView hotels(Model model) {
    model.addAttribute("hotels", hotelBookingService.availableHotels());
    return new ModelAndView("hotels");
}

@GetMapping("/add-hotel")
public String addHotel(Hotel hotel) {
    return "add-hotel";
}

@PostMapping("/add-hotel")
public String addHotel(@RequestParam @NonNull String name) {
    List<String> hotelCandidates = new ArrayList<>();
    hotelCandidates.add(name);
    hotelBookingService.addHotels(hotelCandidates);
    return "redirect:/hotels";
}
```

HotelBookingServiceViewController.java (continue)

```java
    @GetMapping("/edit/{id}")
    public String getUpdateForm(@PathVariable("id") int id, Model model) {
        String hotelName = hotelBookingService.hotel(id);

        Hotel hotel = new Hotel();
        hotel.setId(id);
        hotel.setName(hotelName);
        model.addAttribute("hotel", hotel);
        return "edit-hotel";
    }

    @PostMapping("/edit/{id}")
    public String updateHotel(@PathVariable("id") int id, @RequestParam @NonNull String name) {
        hotelBookingService.updateHotel(id, name);
        return "redirect:/hotels";
    }

    @GetMapping("/delete/{id}")
    public String deleteUser(@PathVariable("id") int id, Model model) {
        hotelBookingService.deleteHotel(id);
        return "redirect:/hotels";
    }
}
```

add-hotel.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Add hotel</title>
</head>
<body>
    <form action="#" th:action="@{/add-hotel}" th:object="${hotel}" method="post">
        <label for="name">Name</label>
        <input type="text" th:field="*{name}" id="name" placeholder="Name">
        <span th:if="${#fields.hasErrors('name')}" th:errors="*{name}"></span>
        <input type="submit" value="Add hotel">
    </form>
</body>
</html>
```

edit-hotel.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Update Hotel</title>
</head>
<body>
  <form action="#"
        th:action="@{/edit/{id}(id=${hotel.id})}"
        th:object="${hotel}"
        method="post">
    <label for="name">Name</label>
    <input type="text" th:field="*{name}" id="name" placeholder="Name">
    <span th:if="${#fields.hasErrors('name')}" th:errors="*{name}"></span>
    <input type="submit" value="Update Hotel">
  </form>
</body>
</html>
```

hotels.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Book Hotels!</title>
</head>
<body>
  <div style="text-align: center;">
    <h1>Hotels Page!</h1>
      <!--<ul>
          <li>Inter Continental</li>
          <li>Sheraton</li>
      </ul>-->
      <div th:switch="${hotels}">
          <h2 th:case="null">No hotel yet!</h2>
          <div th:case="*">
              <h2>Available Hotels</h2>
              <table style="margin-left: auto; margin-right: auto;">
                  <thead>
                  <tr>
                      <th>Id</th>
                      <th>Name</th>
                      <th>Edit</th>
                      <th>Delete</th>
                  </tr>
```

hotels.html (continue)

```
                    </thead>
                    <tbody>
                    <tr th:each="hotel : ${hotels}">
                        <td th:text="${hotel.key}"></td>
                        <td th:text="${hotel.value}"></td>
                        <td><a th:href="@{/edit/{id}(id=${hotel.key})}">Edit</a></td>
                        <td><a th:href="@{/delete/{id}(id=${hotel.key})}">Delete</a></td>
                    </tr>
                    </tbody>
                </table>
            </div>
            <p><a href="/add-hotel">Add a new hotel</a></p>
        </div>
    </div>
</body>
</html>
```

Run, test and improve!

- Let’s discuss any issues and questions.
- ***Add reservation feature!***

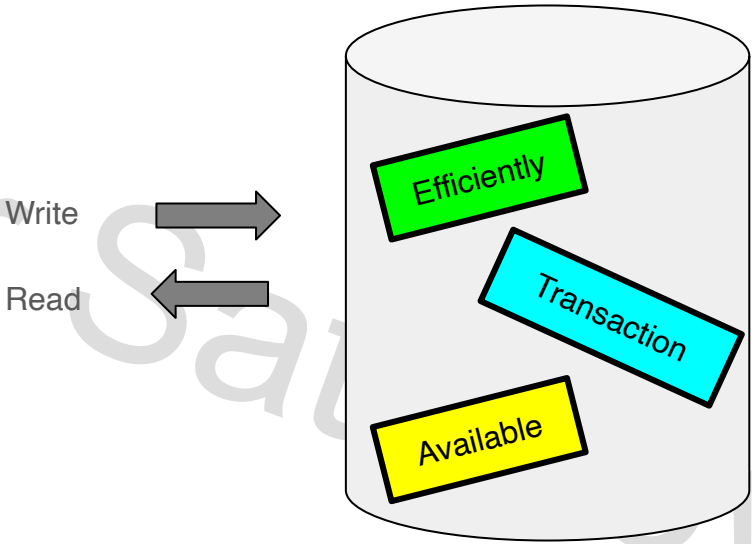
Session 5

Data consistency with Spring

What are Databases?

Is a storage where we have to write and read data from:

- Write and Read efficiently
- Provide Transaction
- Highly Available



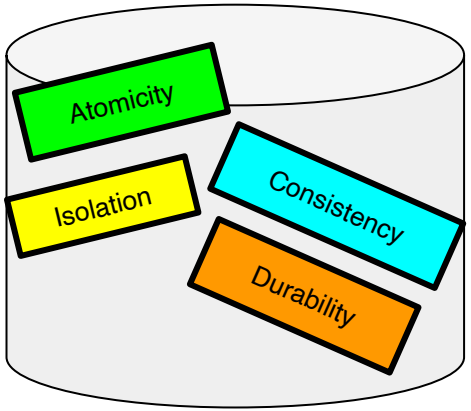
What is ACID?

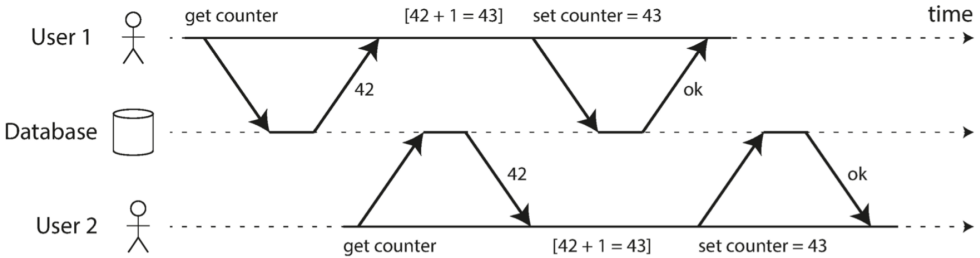
Atomicity - handling faults (All changes takes effect or it nothing does)

Consistency - ensures that a transaction can only bring the database from one valid state to another

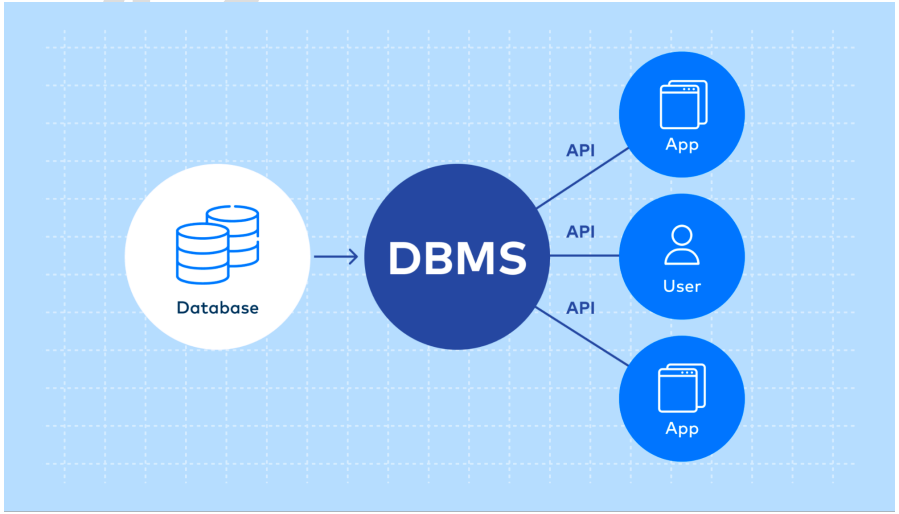
Isolation - result of running transactions concurrently has a same result as they were running in a sequential order

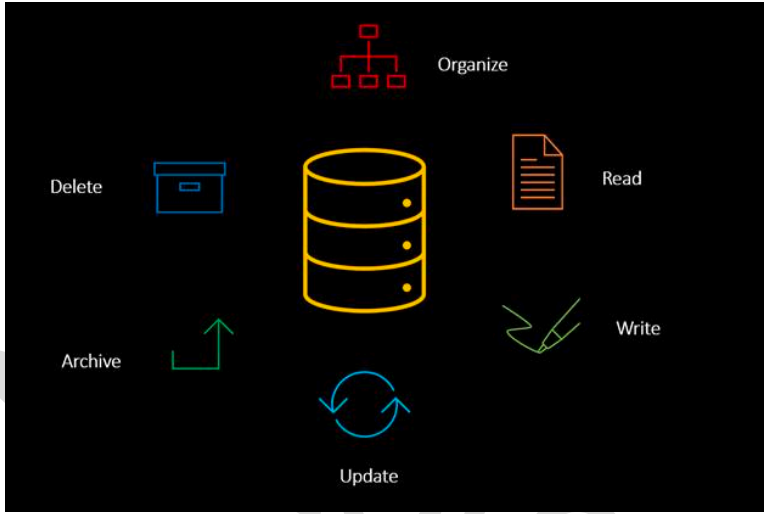
Durability - data is stored on hard disk and not disappeared after reboot



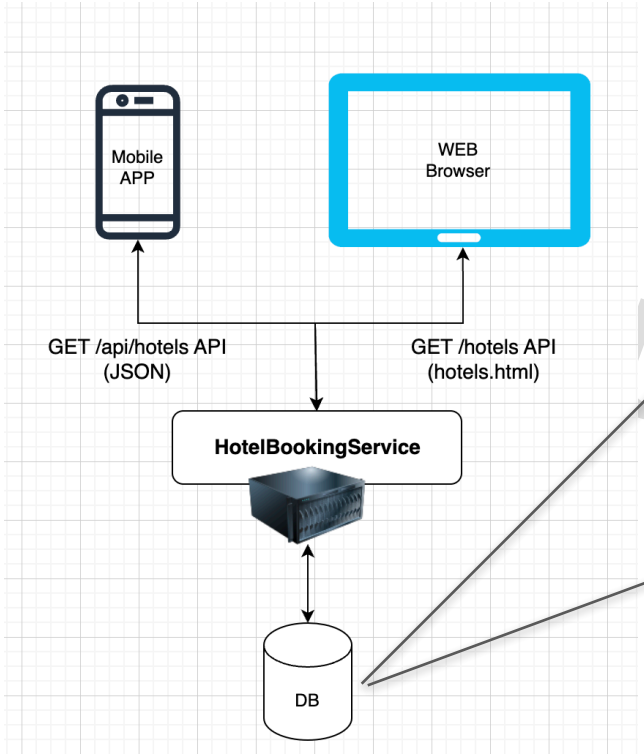


What is a Database Management System?





Hotel Booking Service overall architecture



hotels

id	name
1	Inter Continental
2	Sheraton

reserved_hotels

reserved_date	hotel_id
17-03-2023	2
18-03-2023	1

Spring data JPA (Java Persistent API)

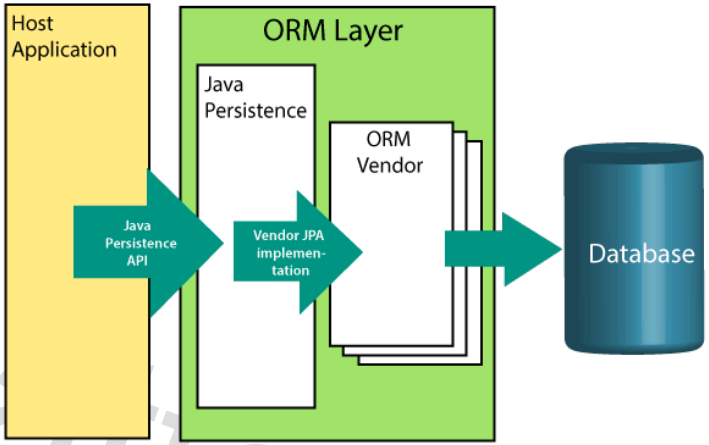
Spring Boot JPA is a Java specification for managing relational data in Java applications.

It allows us to access and persist data between Java object/ class and relational database.

JPA follows Object-Relation Mapping (ORM).

It is a set of interfaces. It also provides a runtime EntityManager API for processing queries and transactions on the objects against the database.

It uses a platform-independent object-oriented query language JPQL (Java Persistent Query Language).



Hibernate

The implementation of JPA specification are provided by many vendors such as:

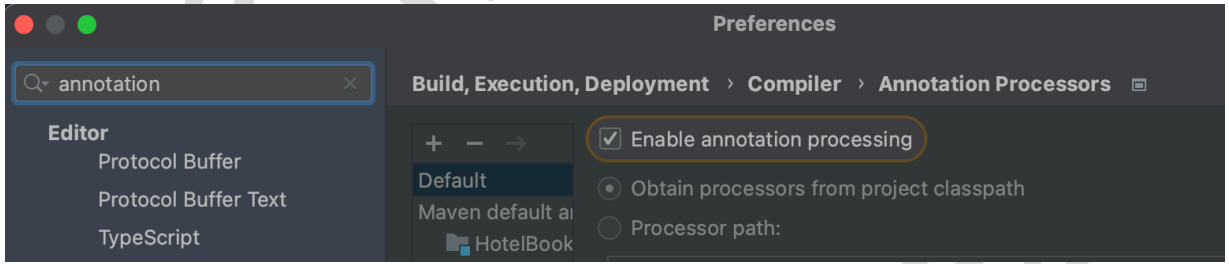
- Hibernate
- Toplink
- iBatis
- OpenJPA etc.

Lombok library and its annotations

- Getter
- Setter
- @NoArgsConstructor
- @AllArgsConstructor
- @Builder(toBuilder = true)
- @NonNull
- etc…

Don’t forget to add Lombok dependency and enable annotation processing in IntelliJ IDEA

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```



[Let’s implement] Database configuration

Remove excluding data source from `HotelBookingServiceApplication.java`

```
(exclude = { DataSourceAutoConfiguration.class })
```

Add data source configuration to `application.properties`

```
spring.datasource.url=jdbc:mysql://localhost/hotel_booking?useSSL=false
spring.datasource.username=yourUserName
spring.datasource.password=yourPassword
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto=none
spring.jpa.show-sql=false
```

Run your MySQL server and connect to it on Workbench.

Update Hotel entity, add @Entity & @Id annotations

```
@Entity
@NoArgsConstructor
public class Hotel {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public int id;

    public String name;
}
```

Create a repository

Create a `repository` folder under `hotelbookingservice` package and add followings

```
package uz.tatu.hotelbookingservice.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import uz.tatu.hotelbookingservice.model.Hotel;

@Repository
public interface HotelBookingServiceRepository extends JpaRepository<Hotel, Integer> {

}
```

Reimplement hotel booking service

```java
@Service
@RequiredArgsConstructor
public class HotelBookingService {

    private final HotelBookingServiceRepository hotelBookingServiceRepository;

    public List<Hotel> availableHotels() {
        return hotelBookingServiceRepository.findAll();
    }

    public Hotel hotel(int id) {
        Optional<Hotel> hotel = hotelBookingServiceRepository.findById(id);
        if (hotel.isEmpty()) {
            throw new NoHotelFoundException(id);
        }
        return hotel.get();
    }

    public List<Hotel> addHotels(List<Hotel> hotels) {
        return hotelBookingServiceRepository.saveAll(hotels);
    }
```

Reimplement hotel booking service (continue)

```
    public List<Hotel> updateHotels(List<Hotel> hotels) {
        List<Hotel> modifiedHotels = new ArrayList<>();
        for (Hotel hotel: hotels) {
            Optional<Hotel> hotelInDb = hotelBookingServiceRepository.findById(hotel.getId());
            if (hotelInDb.isPresent()) {
                Hotel modifiedHotel = hotelBookingServiceRepository.save(hotel);
                modifiedHotels.add(modifiedHotel);
            }
        }
        return modifiedHotels;
    }
  public void deleteHotel(int id) {
        hotelBookingServiceRepository.deleteById(id);
    }
}
```

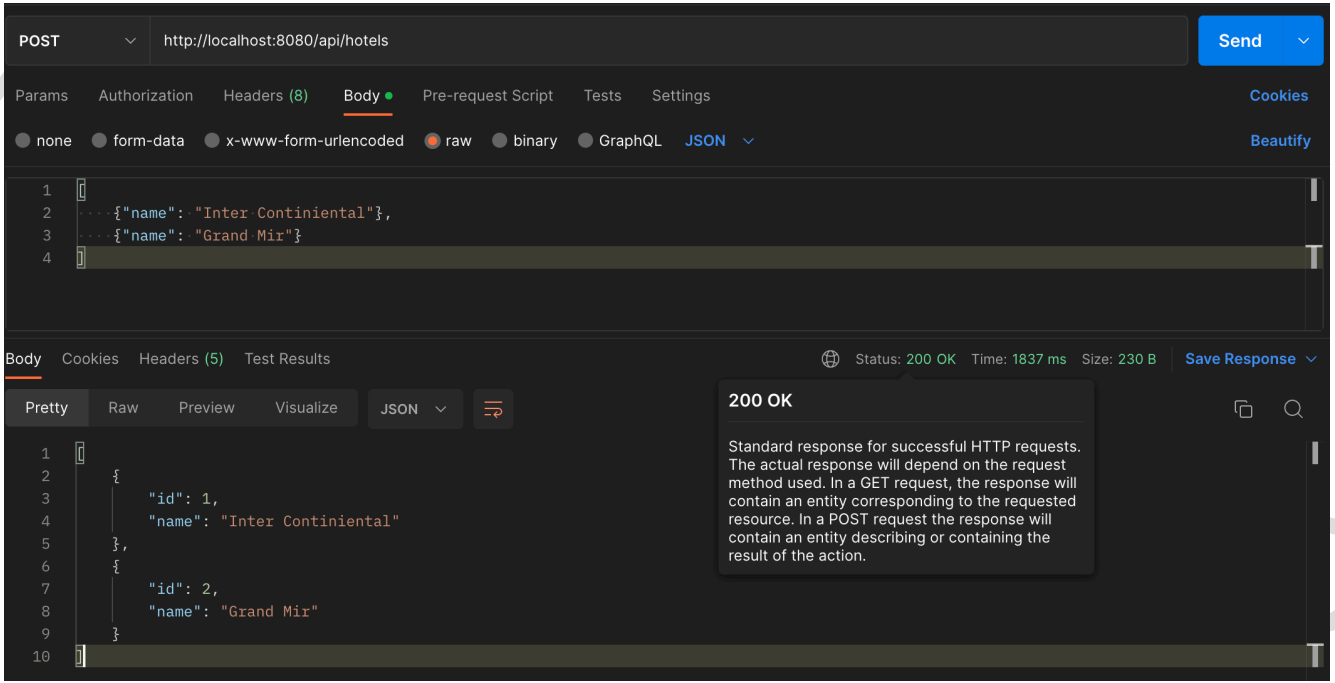
Update controller

```java
@GetMapping("/hotels")
public ResponseEntity<List<Hotel>> hotels() {
    List<Hotel> hotels = hotelBookingService.availableHotels();
    HttpHeaders headers = new HttpHeaders();
    headers.add("Custom-Header", "HOTEL-BOOKING-SERVICE-HEADER");

    // we can also create our own Generic Response entity
    return new ResponseEntity<>(hotels, headers, HttpStatus.OK);
}

@GetMapping("/hotels/{id}")
public ResponseEntity<Hotel> hotel(@PathVariable int id) {
    return new ResponseEntity<>(hotelBookingService.hotel(id), HttpStatus.OK);
}

@PostMapping("/hotels")
public ResponseEntity<List<Hotel>> hotels(@NonNull @RequestBody List<Hotel> hotels) {
    List<Hotel> createdHotels = hotelBookingService.addHotels(hotels);
    return new ResponseEntity<>(createdHotels, HttpStatus.OK);
}
```

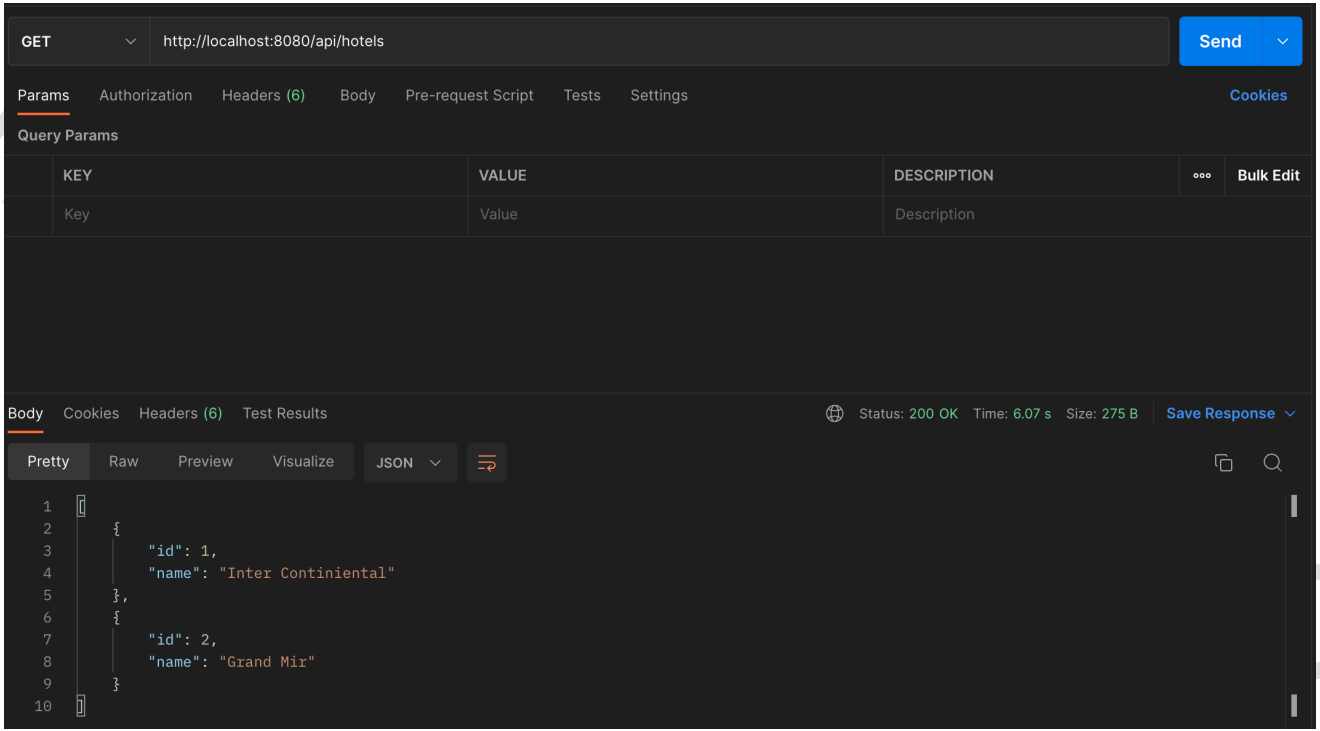
Update controller (continue)

```java
@PutMapping("/hotels/{id}")
public ResponseEntity<List<Hotel>> updateHotels(@RequestBody @NonNull List<Hotel> hotels) {
    List<Hotel> updatedHotels = hotelBookingService.updateHotels(hotels);
    return new ResponseEntity<>(updatedHotels, HttpStatus.OK);
}

@DeleteMapping("/hotels/{id}")
public void deleteHotel(@PathVariable int id) {
    hotelBookingService.deleteHotel(id);
}
```

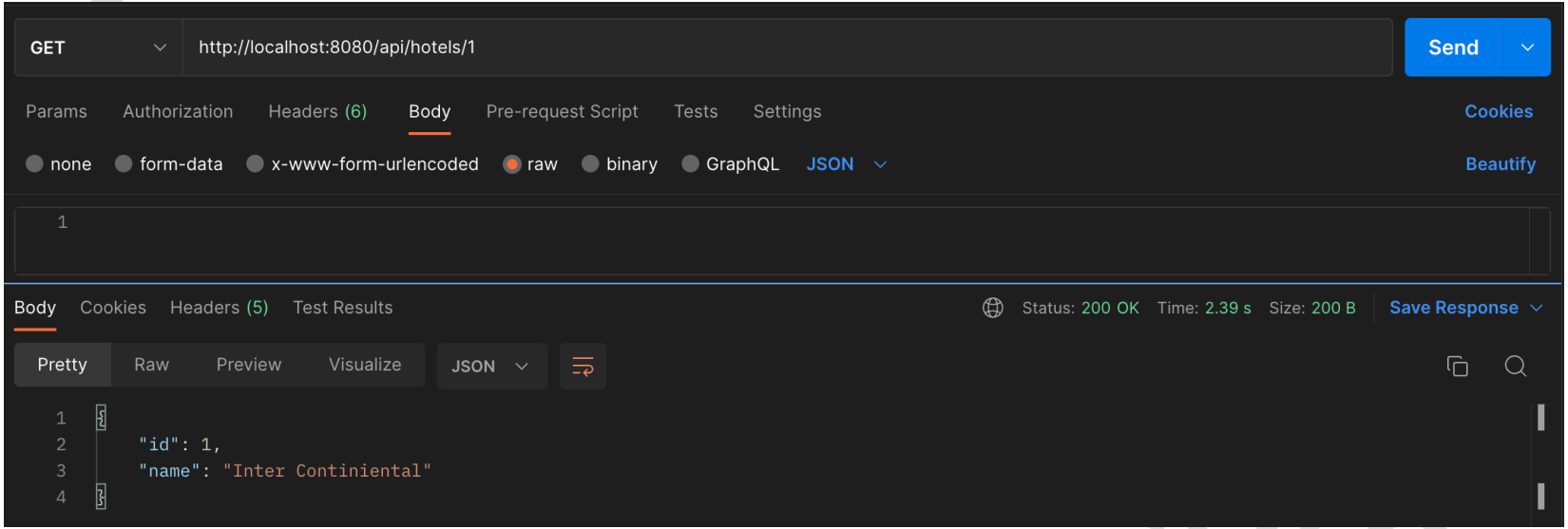
[Test] Add hotels



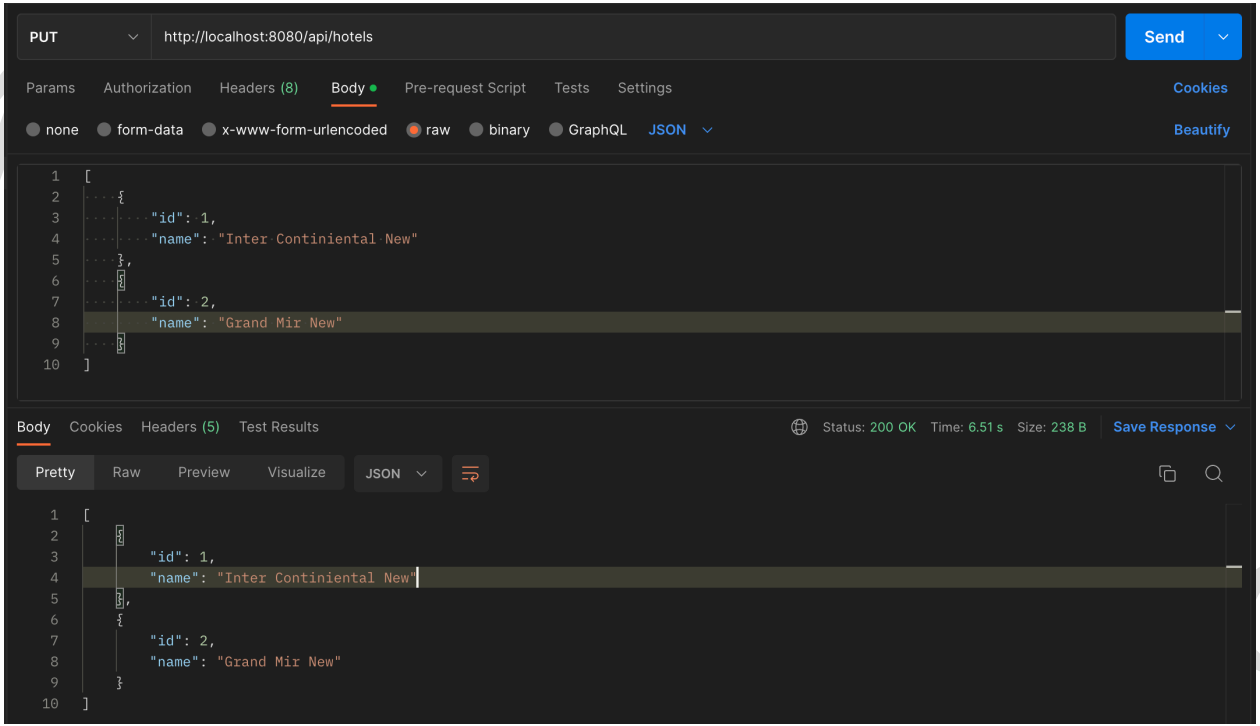
[Test] Get hotels



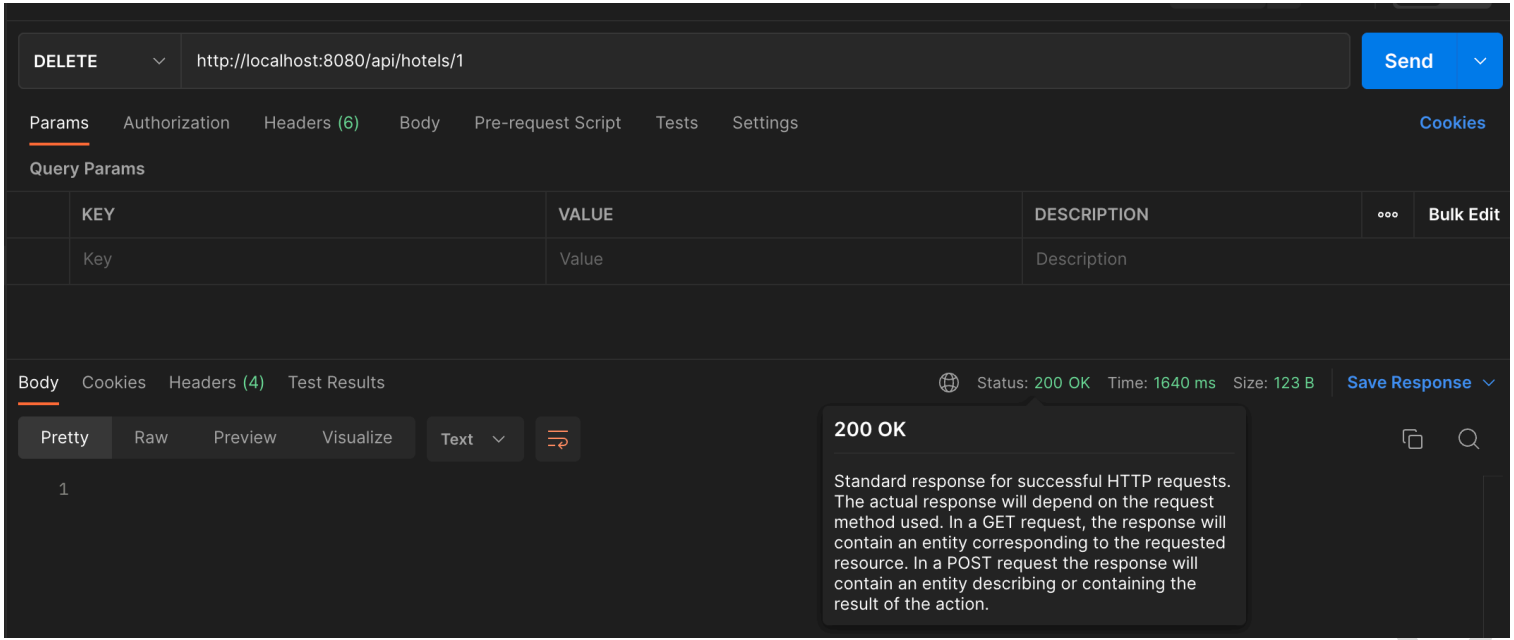
[Test] Get single hotel



[Test] Update hotels



[Test] Delete hotel



Custom JPA query

Update repository with native MySQL query

```
@Repository
public interface HotelBookingServiceRepository extends JpaRepository<Hotel, Integer> {

    @Query(value = "select * from hotel where name like CONCAT('%', :name, '%')",
     nativeQuery = true)
    List<Hotel> searchByName(@Param(value = "name") String name);
}
```

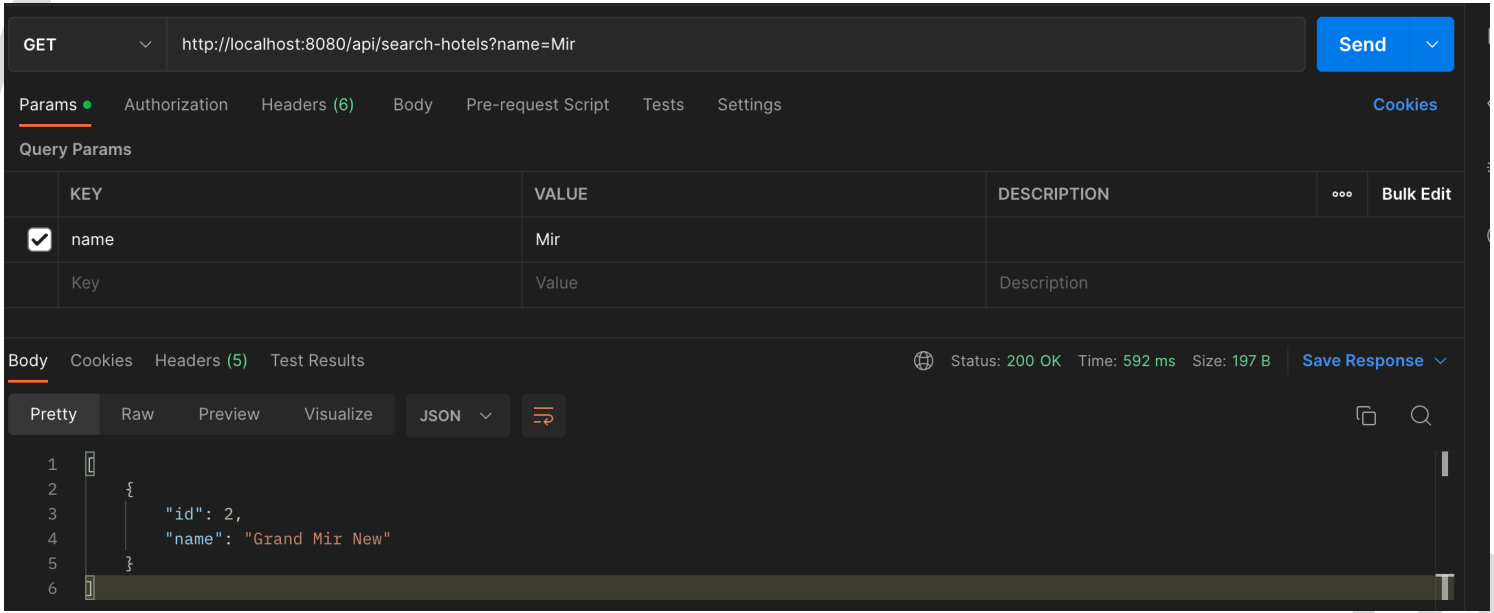
Update service

```
public List<Hotel> search(@NonNull String nameLookLike) {
    return hotelBookingServiceRepository.searchByName(nameLookLike);
}
```

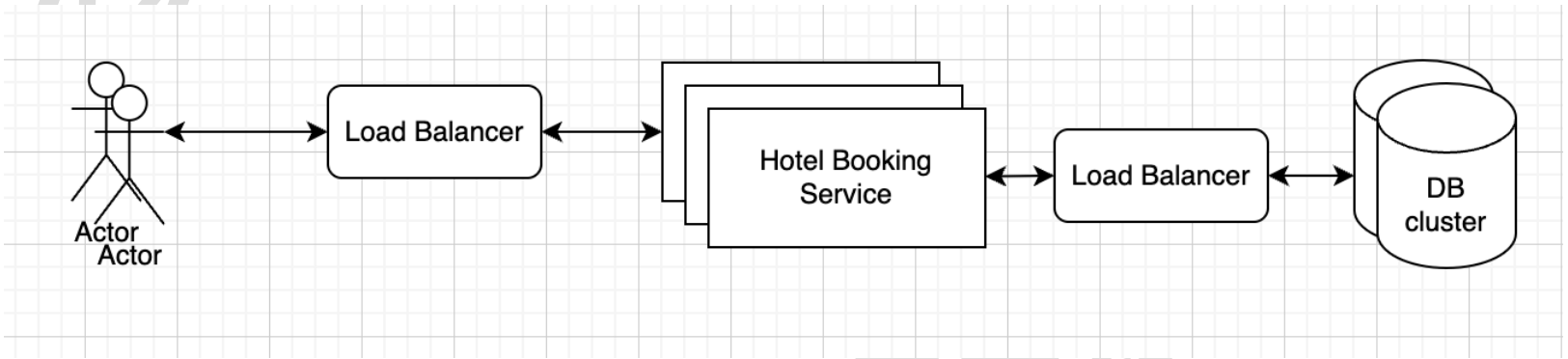
Add controller

```
@GetMapping("/search-hotels")
public ResponseEntity<List<Hotel>> hotels(@RequestParam String name) {
    List<Hotel> hotels = hotelBookingService.search(name);
    return new ResponseEntity<>(hotels, HttpStatus.OK);
}
```

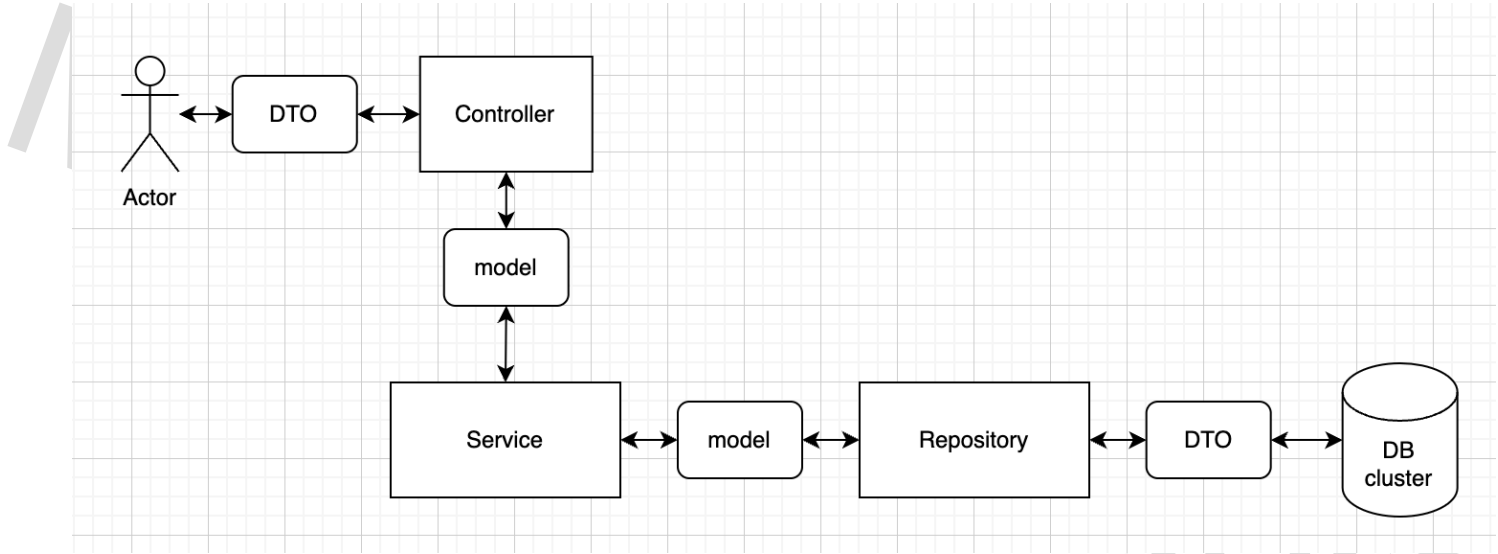
Search hotel test



How real applications System Architecture look like?



How real applications S/W architecture look like?



Useful Spring tools

1. Logging log4j2
2. Actuator
3. Swagger
4. H2 database