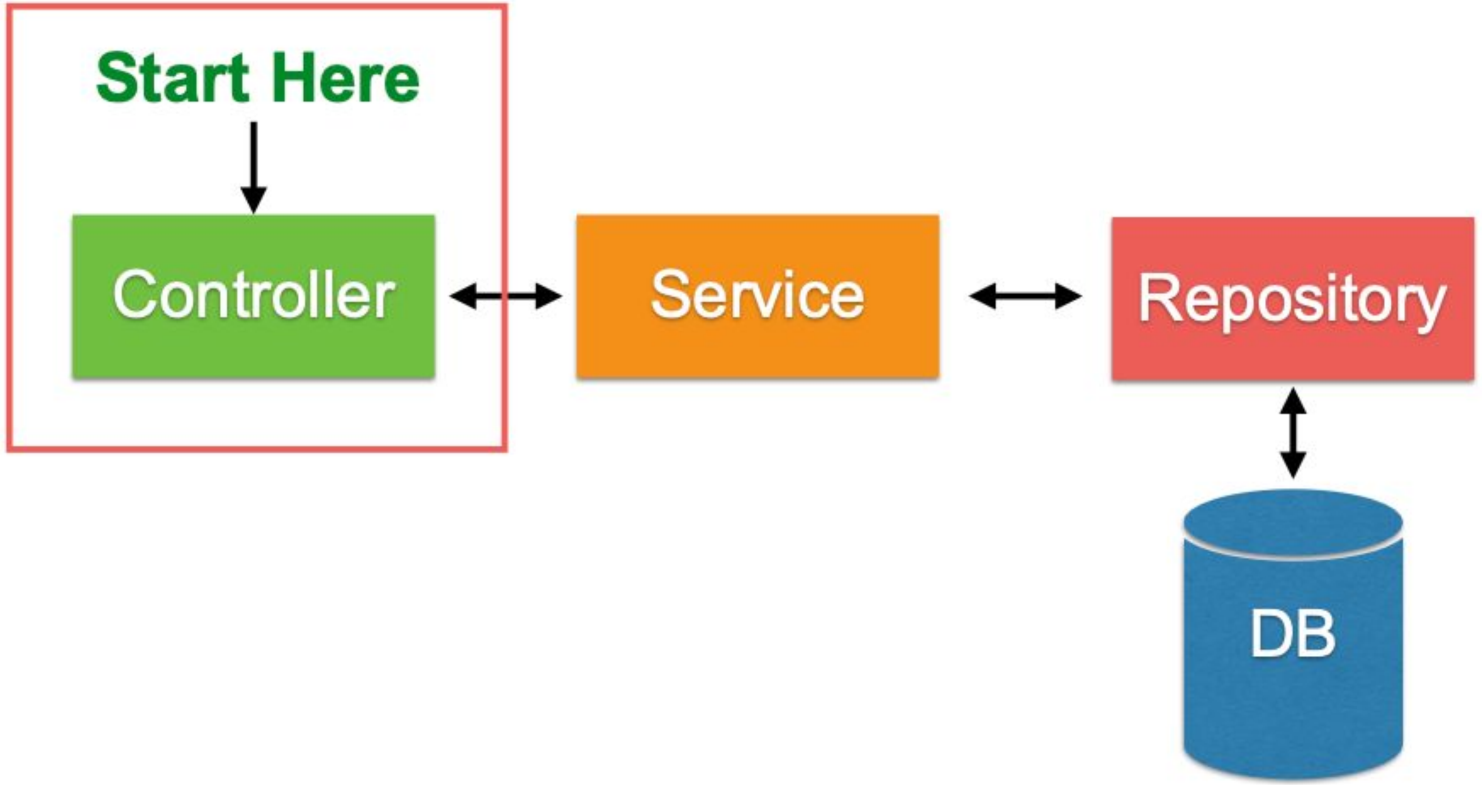
Spring Framework

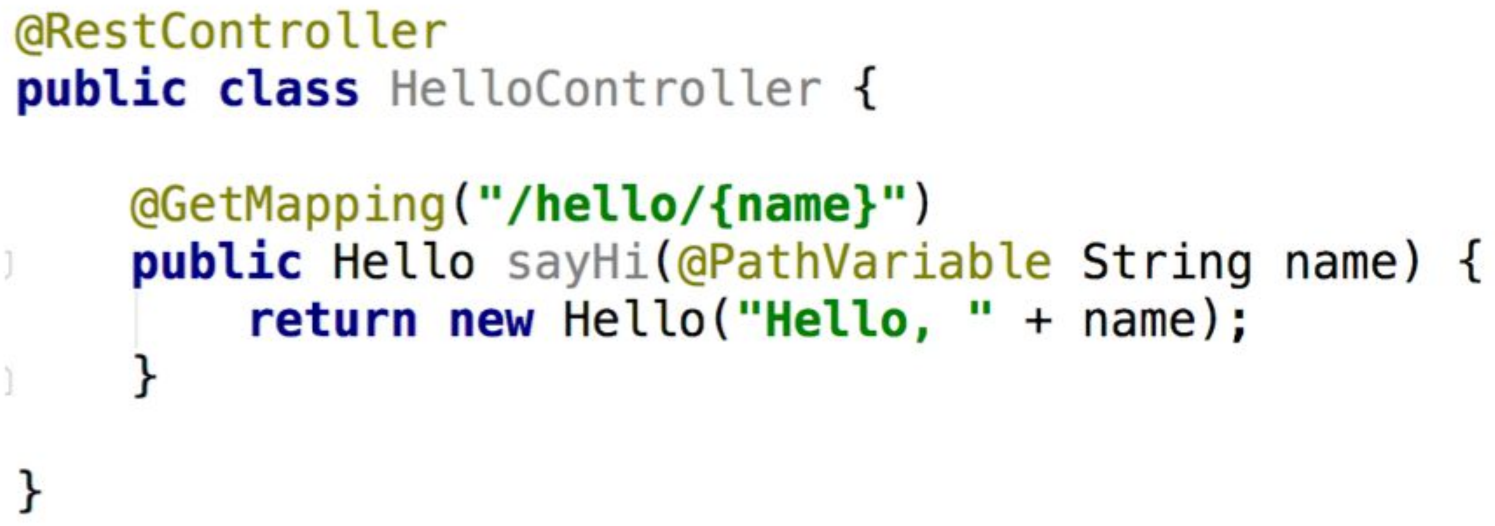
Create first RESTful API

Basic structure of Spring Boot



1. Create REST Controller

HelloController.java



```java
@RestController
public class HelloController {

    @GetMapping("/hello/{name}")
    public Hello sayHi(@PathVariable String name) {
        return new Hello("Hello, " + name);
    }

}
```

2. Create model class

Hello.java

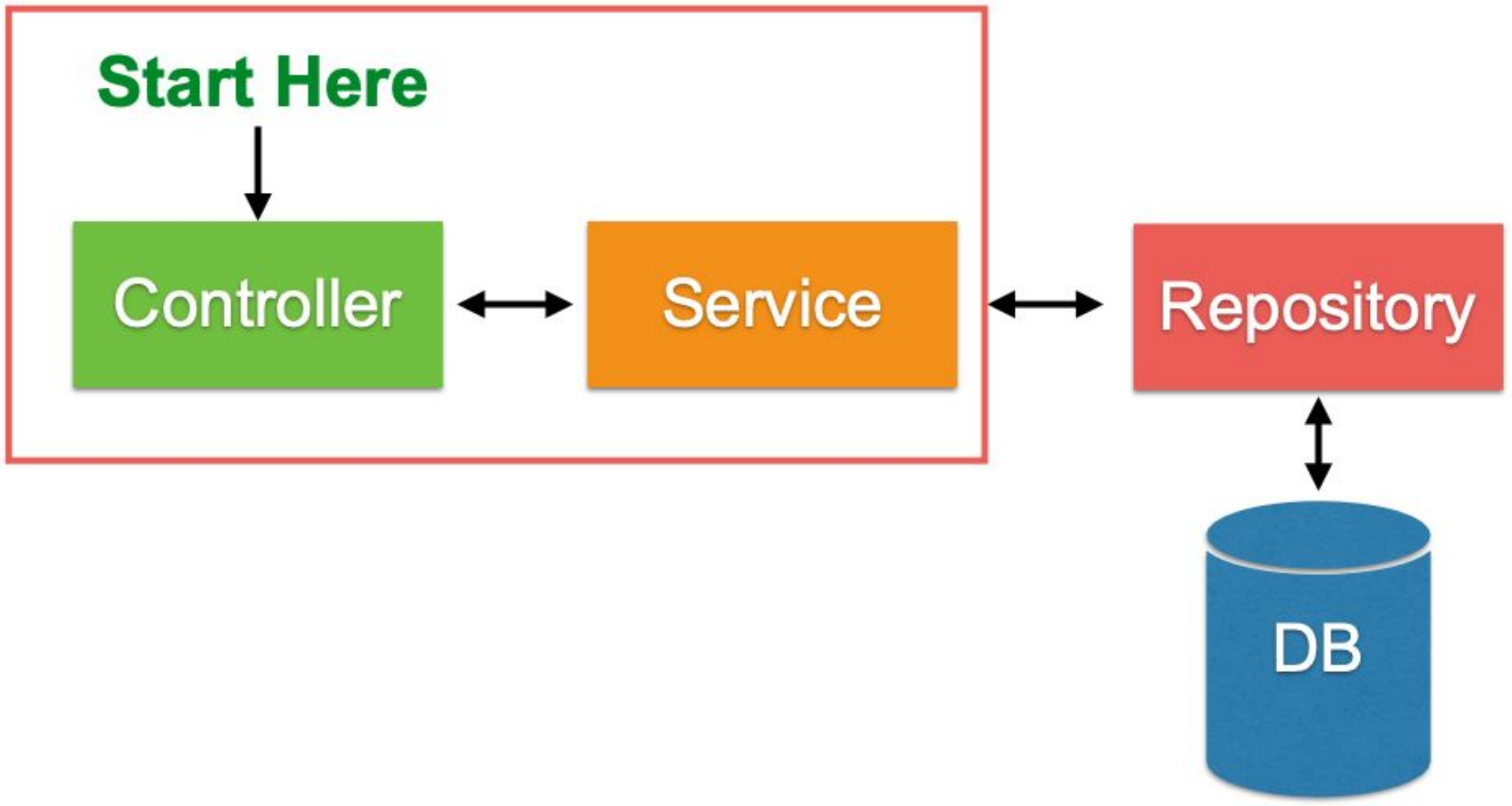
```java
public class Hello {

    private String message;

    Hello(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

3. Compile and Packaging

```
$mvnw clean package
```

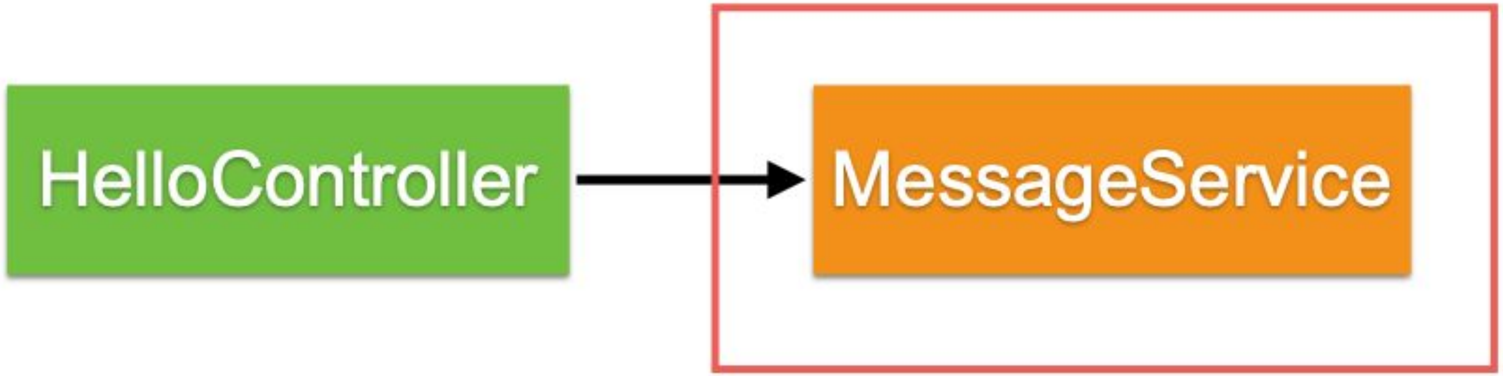
Move business logic to service

Working with service

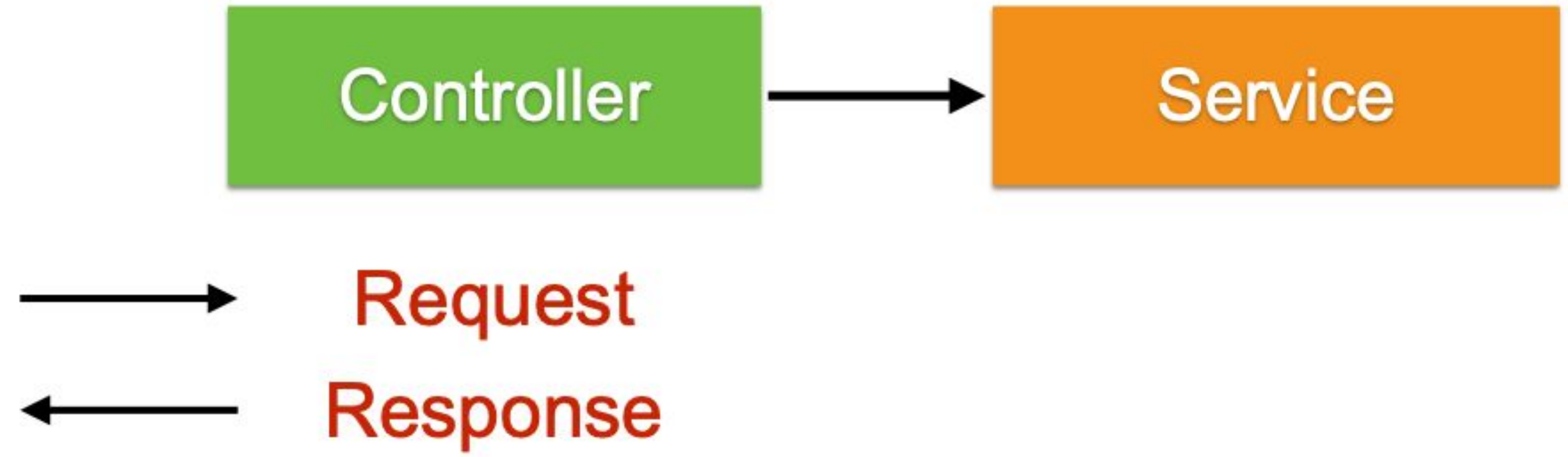


Move business logic to service

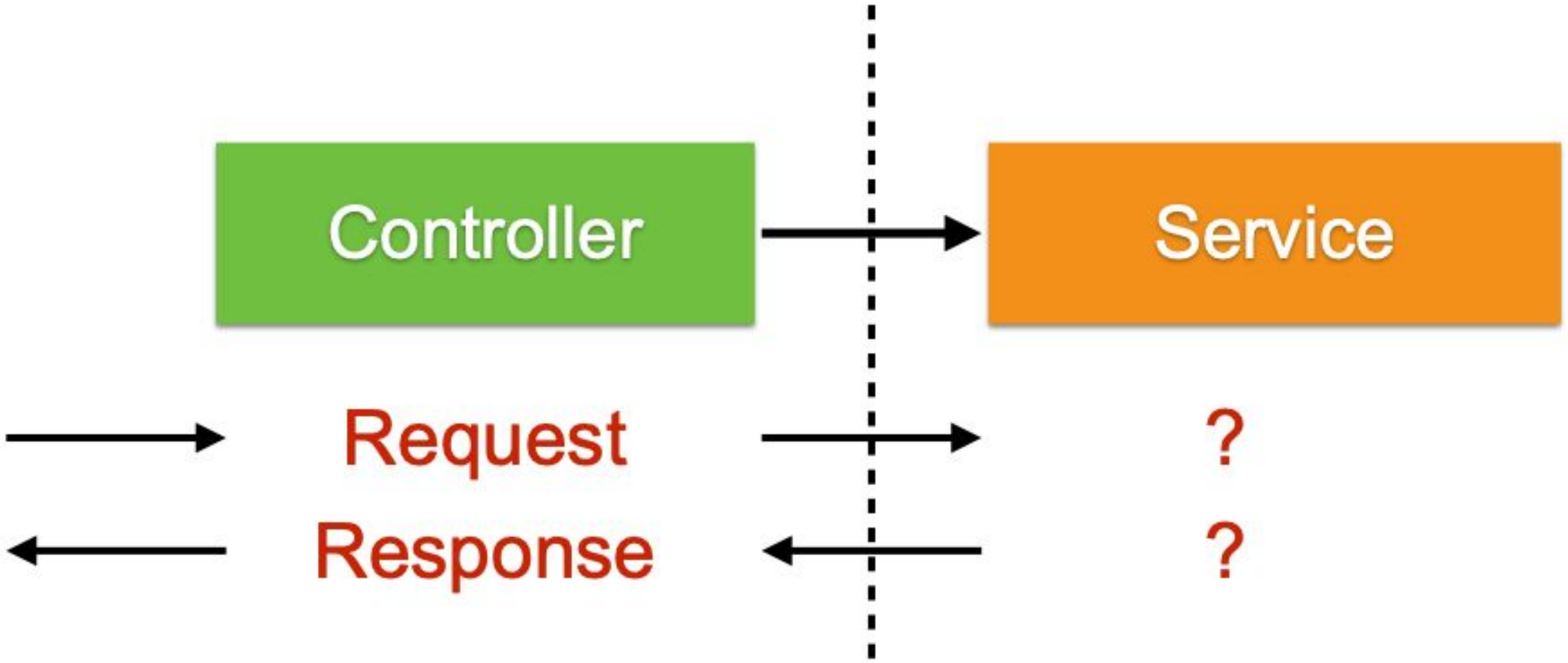
Service class or interface ?



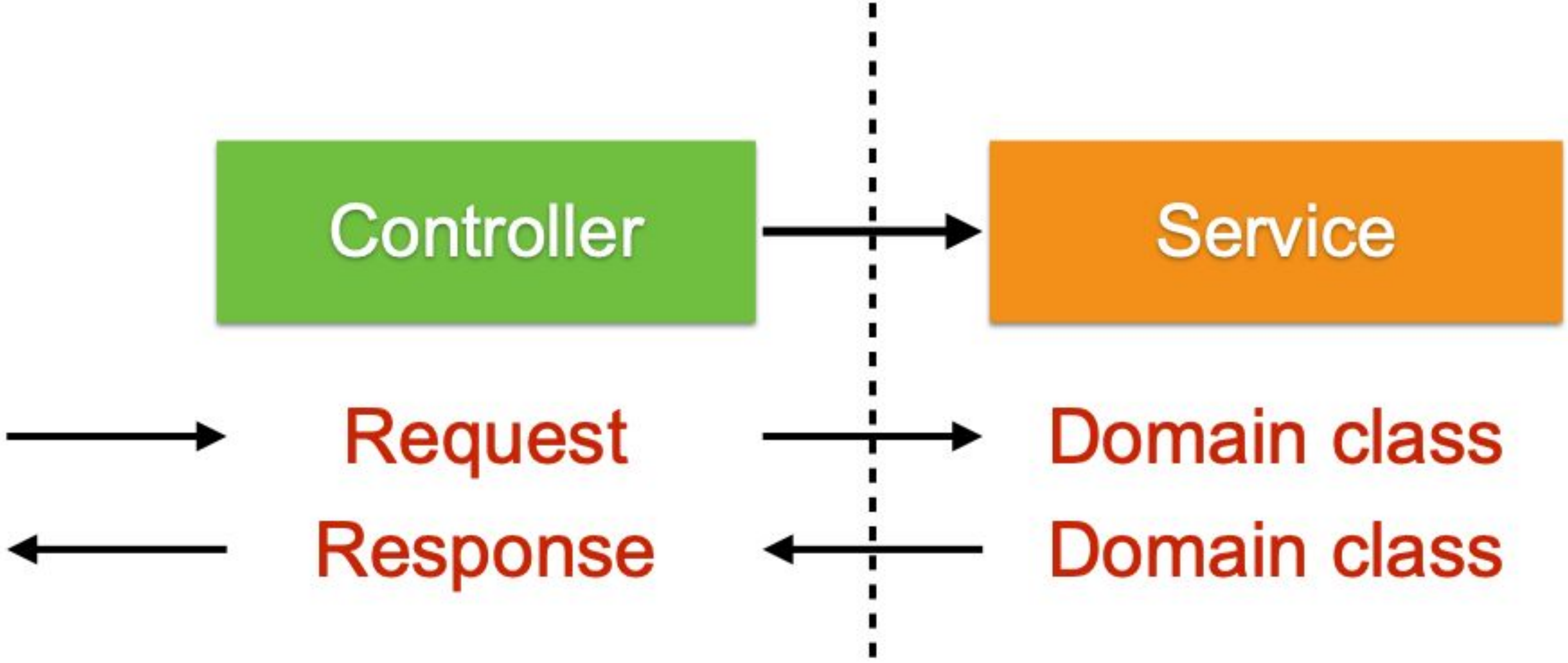
Data Model for service ?



Data Model ?

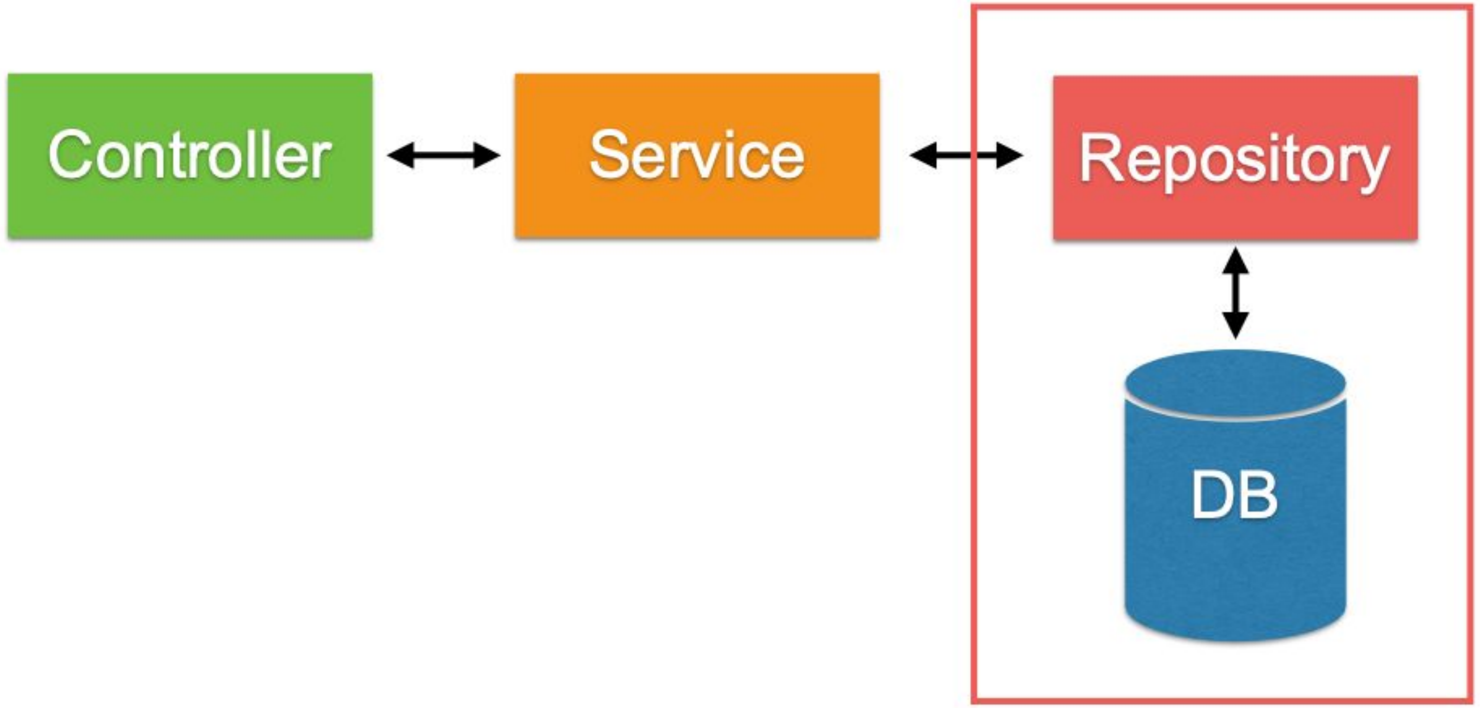


Data Model ?

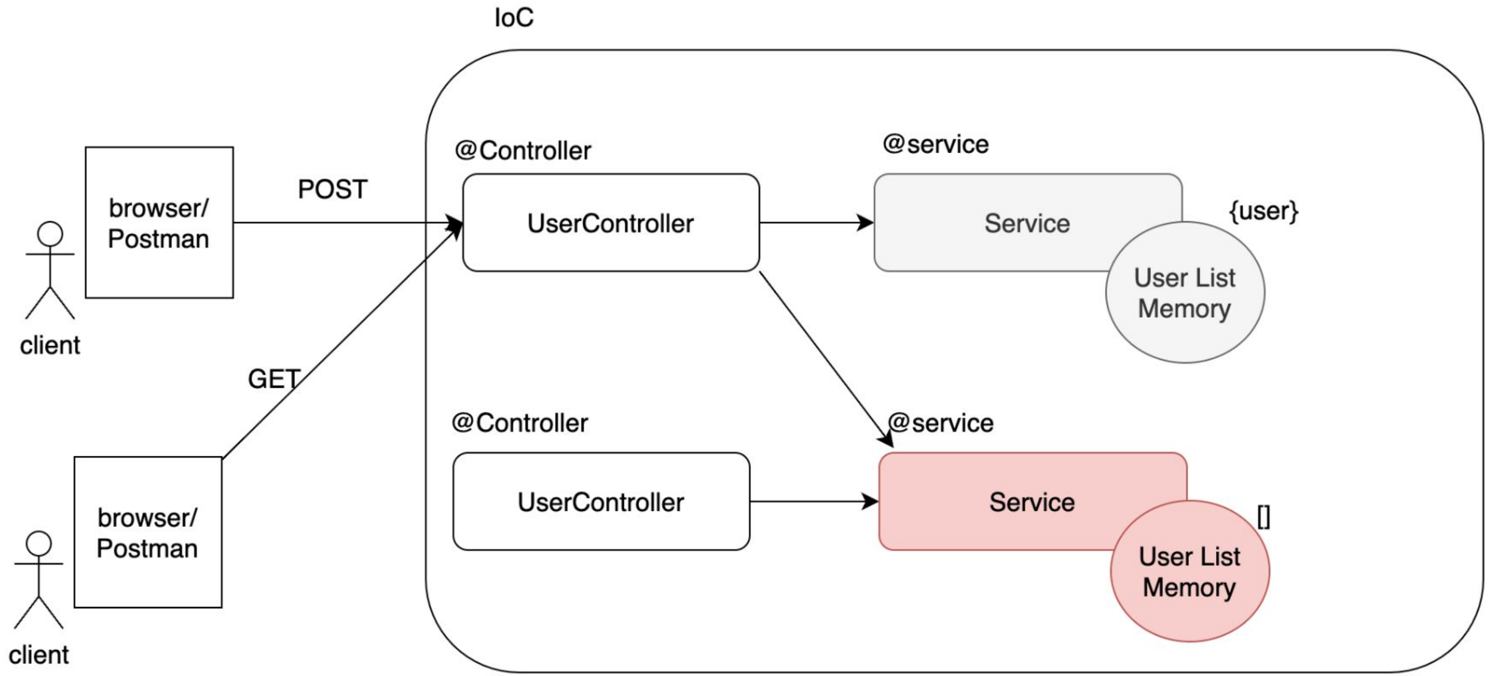

Controller
Service
Request
Domain class
Response
Domain class

Working with Repository

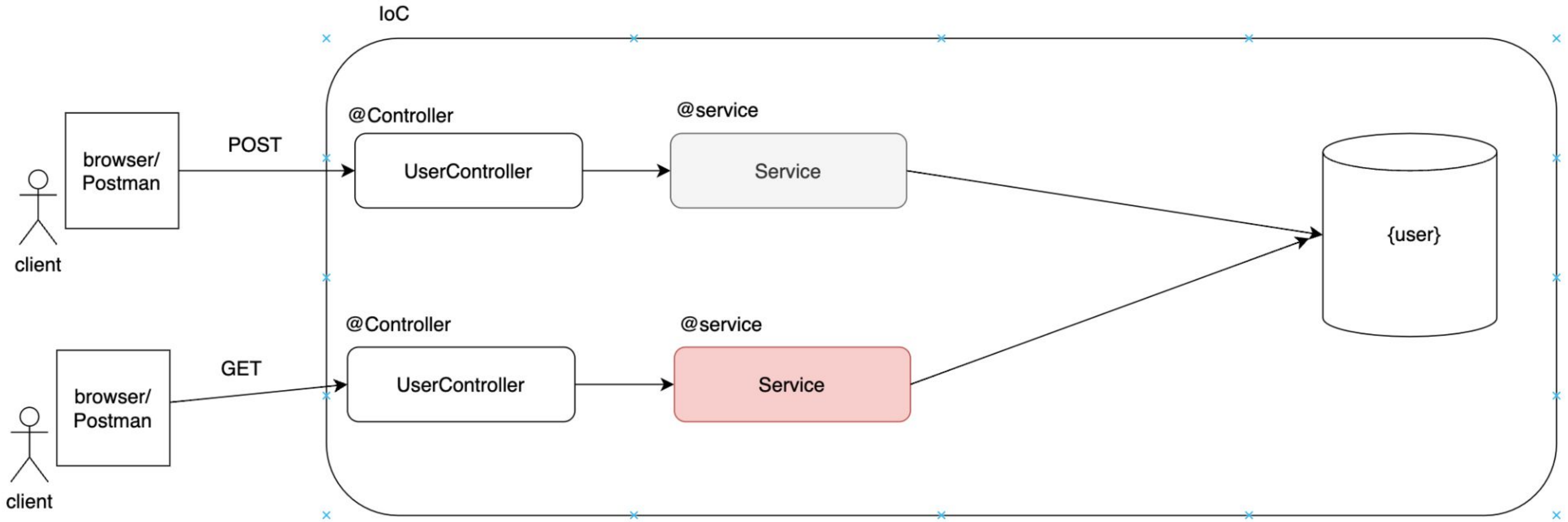
Working with repository



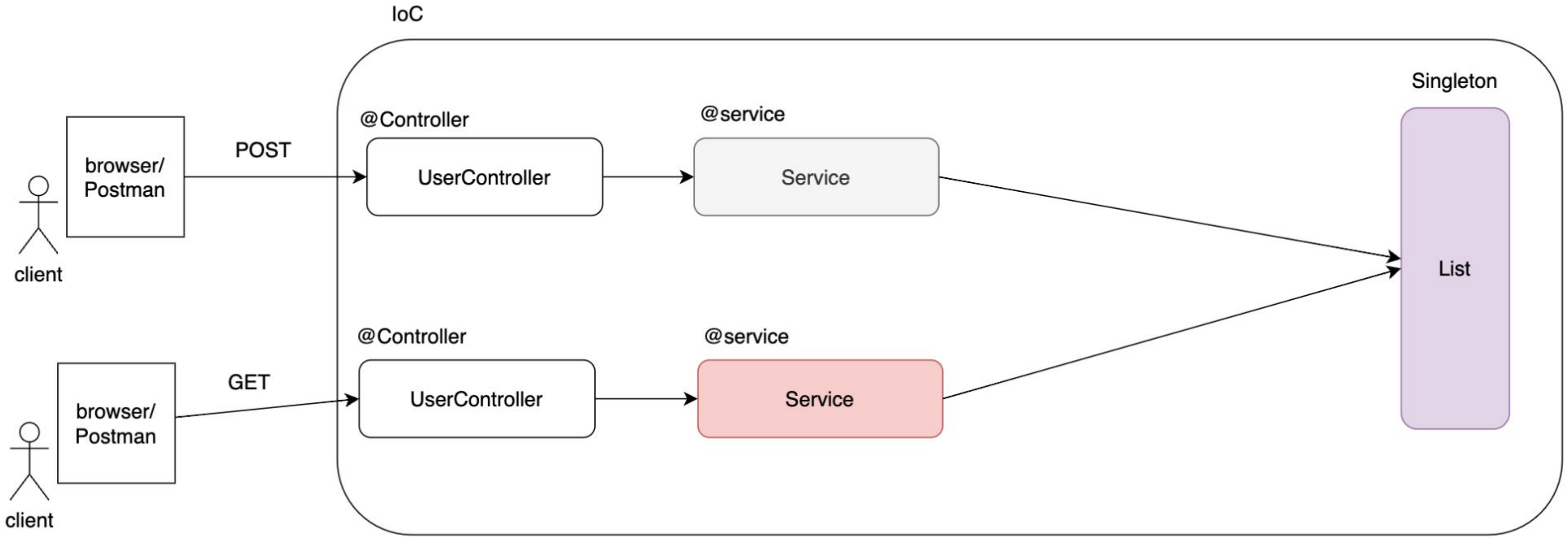
```java
@Service
public class UserService {
    List users = new ArrayList<String>();

}
```



```java
@Service
public class UserService {
    List users = new ArrayList<String>();

}
```



```java
@Service
public class UserService {
    List users = new ArrayList<String>();

}
```



@Configuration

Indicates that the class can be used by the Spring IoC container as a source of bean definitions.

classes are processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

```
public class MySingletonBean {
    public void logMessage(String message) {
        System.out.println("Message: " + message);
    }
}
```

Singleton

Ensures that a bean is instantiated only once in the Spring container.

Singleton beans are created only once and the same instance is returned for each subsequent request in the same Spring container.

```
@Configuration
public class AppConfig {
    @Bean
    public MySingletonBean mySingletonBean() {
        return new MySingletonBean();
    }
}
```

10 minutes

Break 10:35

User story

As a User,
I want User service API
So that I can a CRUD operation to user data

Design RESTFul APIs

Method	Path	Description
GET	/users	Get all users
GET	/users/{id}	Get user by id
POST	/users	Create new user
PUT	/users/{id}	Update user
DELETE	/users/{id}	Delete user by id

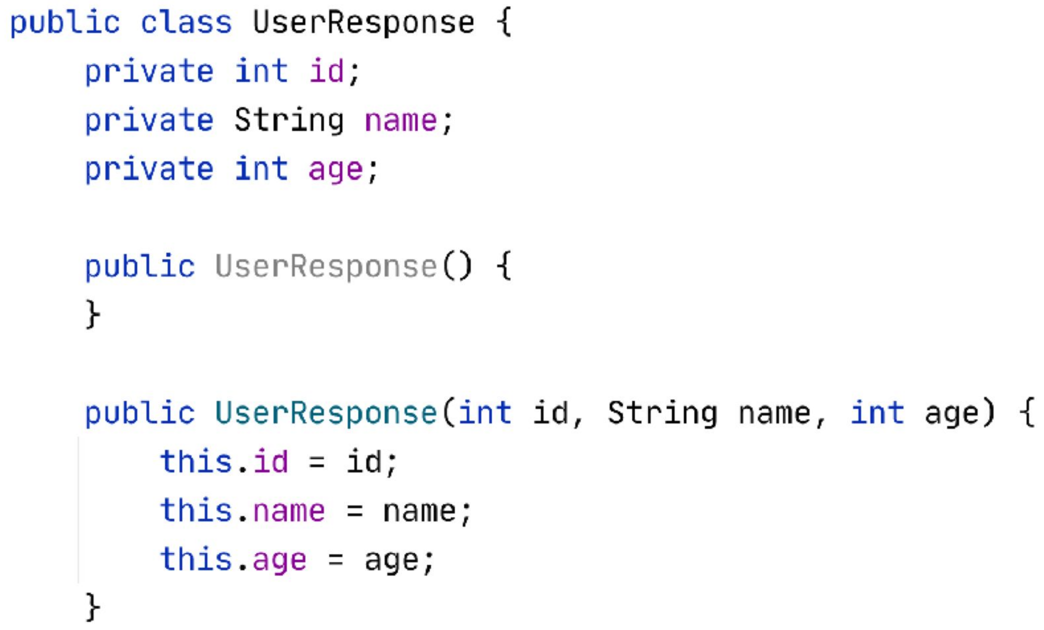
1. Create REST Controller

UserController.java

```
@RestController
public class UserController {

    @GetMapping("/users")
    public List<UserResponse> getAllUsers() {
        List<UserResponse> userResponseList = new ArrayList<>();
        userResponseList.add(new UserResponse(1,"demo 1", 30));
        userResponseList.add(new UserResponse(2,"demo 2", 35));
        return userResponseList;
    }
```

2. Create model class

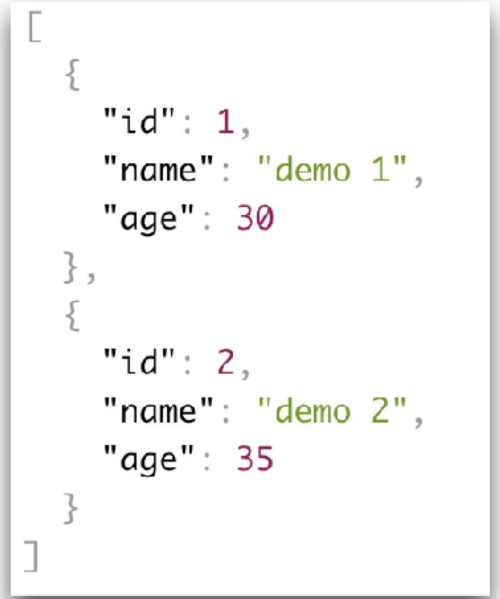
UserResponse.java



```java
public class UserResponse {
    private int id;
    private String name;
    private int age;

    public UserResponse() {
    }

    public UserResponse(int id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }
```

Open in browser

http://localhost:8080/users



```
[
  {
    "id": 1,
    "name": "demo 1",
    "age": 30
  },
  {
    "id": 2,
    "name": "demo 2",
    "age": 35
  }
]
```

Create more APIs

Get user by id

Create a new user

Update user by id

Delete user by id

Get user by id

GET /users/{id}

```java
@GetMapping("/users/{id}")
public UserResponse getUserById(@PathVariable int id) {
    UserResponse userResponse = new UserResponse(id, "Demo", 40);
    return userResponse;
}
```

Create a new user

POST /users

```
@PostMapping("/users")
public UserResponse createNewUser(@RequestBody UserRequest newUser)
{
    UserResponse newUserResponse = new UserResponse(
            1,
            newUser.getName(),
            newUser.getAge());
    return newUserResponse;
}
```

Update user by id

PUT /users/{id}

```java
@PutMapping("/users/{id}")
public UserResponse updateUser(@RequestBody UserRequest newUser,
                               @PathVariable int id) {
    // TODO
    // 1. find by id
    // 2. found => update user
    // 3. not found => ?? (create ? or throw error)
    UserResponse updatedUserResponse = new UserResponse(
            id,
            newUser.getName(),
            newUser.getAge());
    return updatedUserResponse;
}
```

Delete user by id

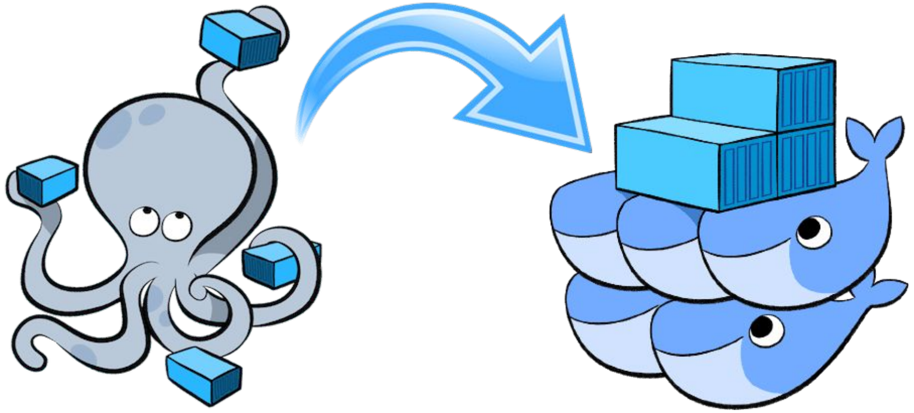
DELETE /users/{id}

```
@DeleteMapping("/users/{id}")
public void deleteUser(@PathVariable int id) {
    // TODO
}
```

Postgres

```
services:
  postgres:
    image: postgres:15.3-alpine
    container_name: postgresql
    volumes:
      - ./postgresdata:/data/db
    ports:
      - ${POSTGRES_PORT}:5432
    environment:
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_USERNAME=${POSTGRES_USERNAME}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
```

docker-compose.yml



Take a break [Lunch]