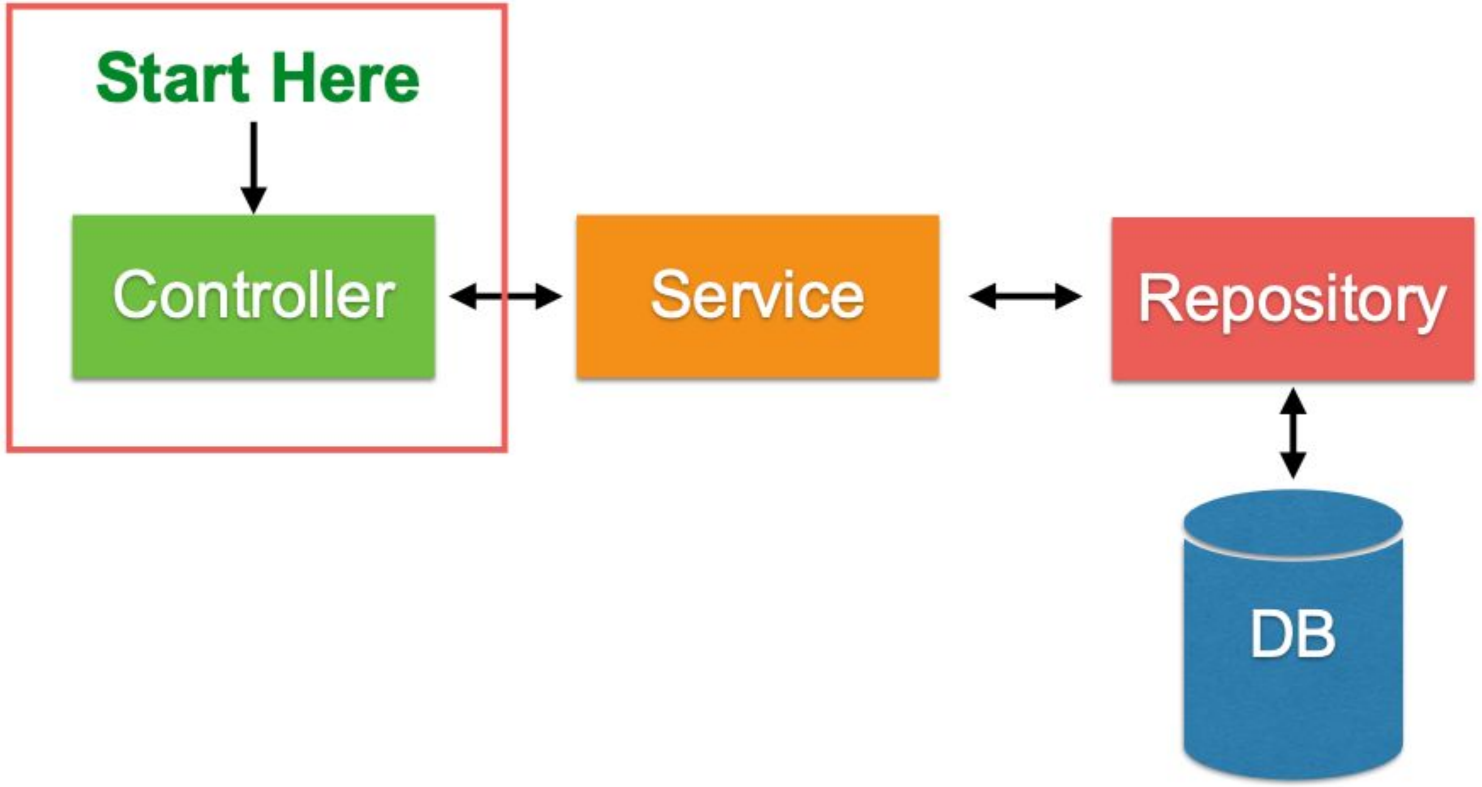
Spring Framework

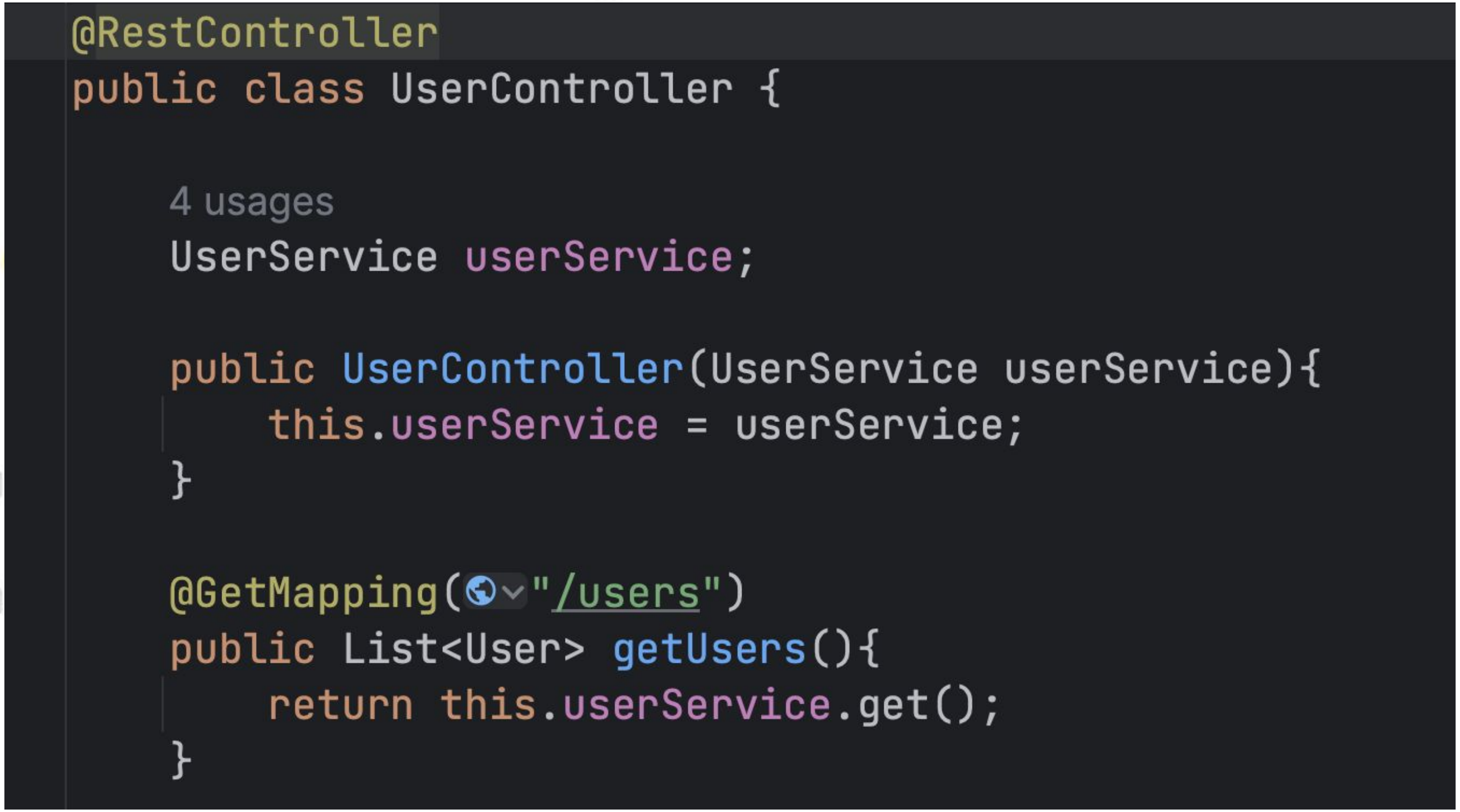
Start at 09:15

Create first RESTful API

Basic structure of Spring Boot

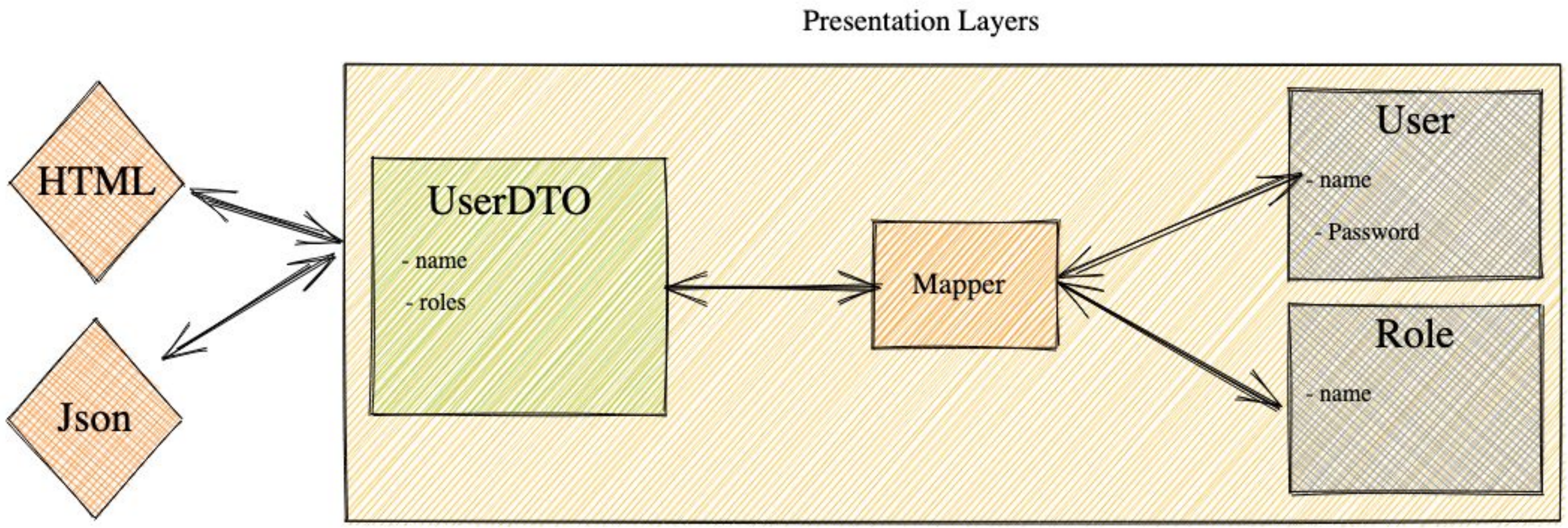


1. Create REST Controller



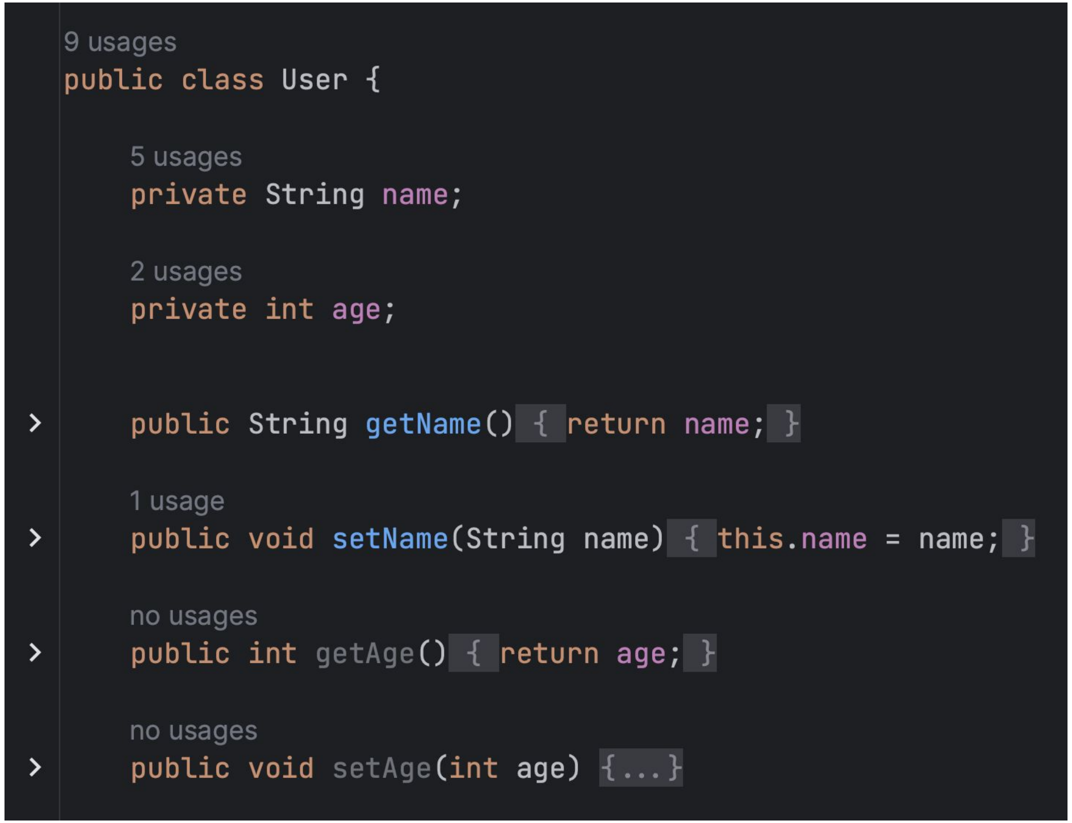
```
@RestController
public class UserController {

    4 usages
    UserService userService;

    public UserController(UserService userService){
        this.userService = userService;
    }

    @GetMapping("/users")
    public List<User> getUsers(){
        return this.userService.get();
    }
```

DTO



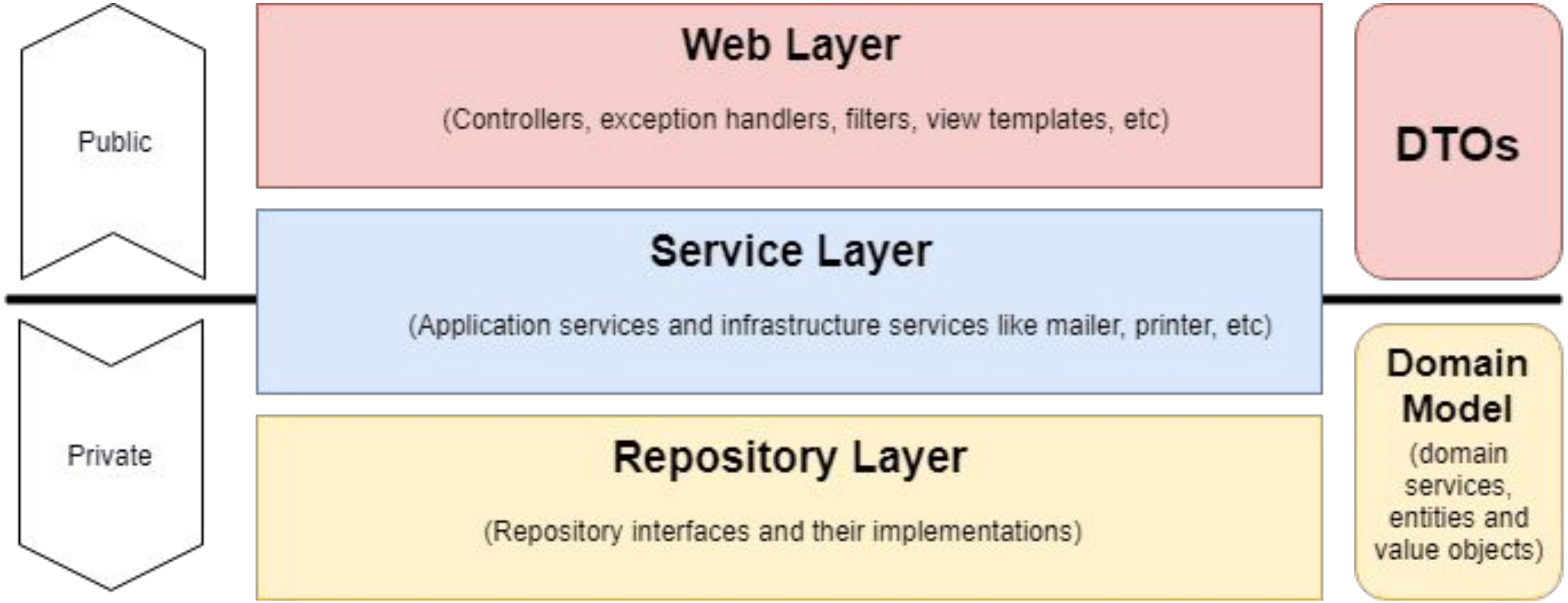
DTO stands for Data Transfer Object. DTOs are used to transfer data between layers and tiers in a software application. They are especially useful when the data being transferred is more than a single primitive or a simple object, or when you want to encapsulate multiple pieces of data into a single object.

2. Create model class



```
9 usages
public class User {

    5 usages
    private String name;

    2 usages
    private int age;


    public String getName() { return name; }

    1 usage
    public void setName(String name) { this.name = name; }

    no usages
    public int getAge() { return age; }

    no usages
    public void setAge(int age) {...}
```

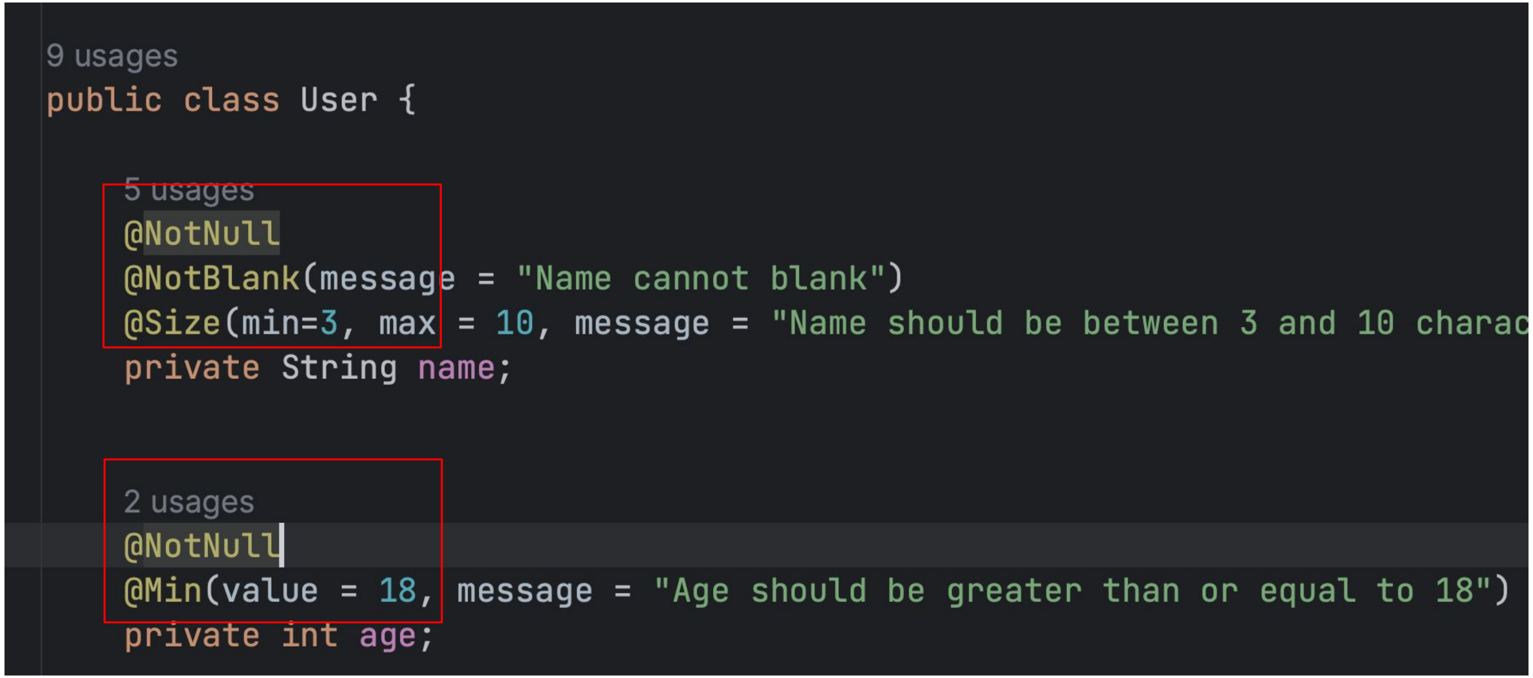
Controller Input Validation


Public
Private
Web Layer
(Controllers, exception handlers, filters, view templates, etc)
Service Layer
(Application services and infrastructure services like mailer, printer, etc)
Repository Layer
(Repository interfaces and their implementations)
DTOs
Domain Model
(domain services, entities and value objects)

Add Maven dependency

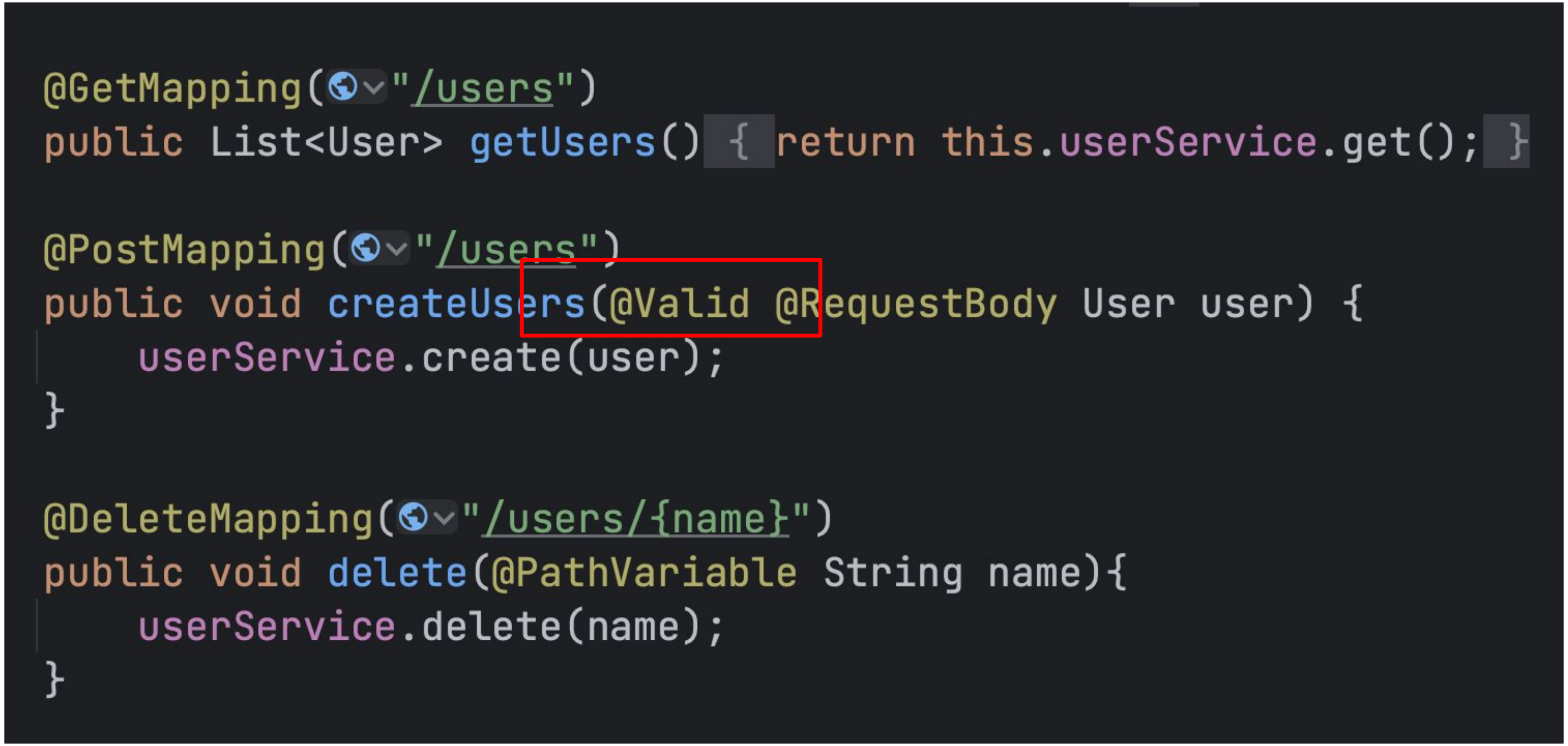
```
<dependency>

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

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

</dependency>
```

Add validation constraints



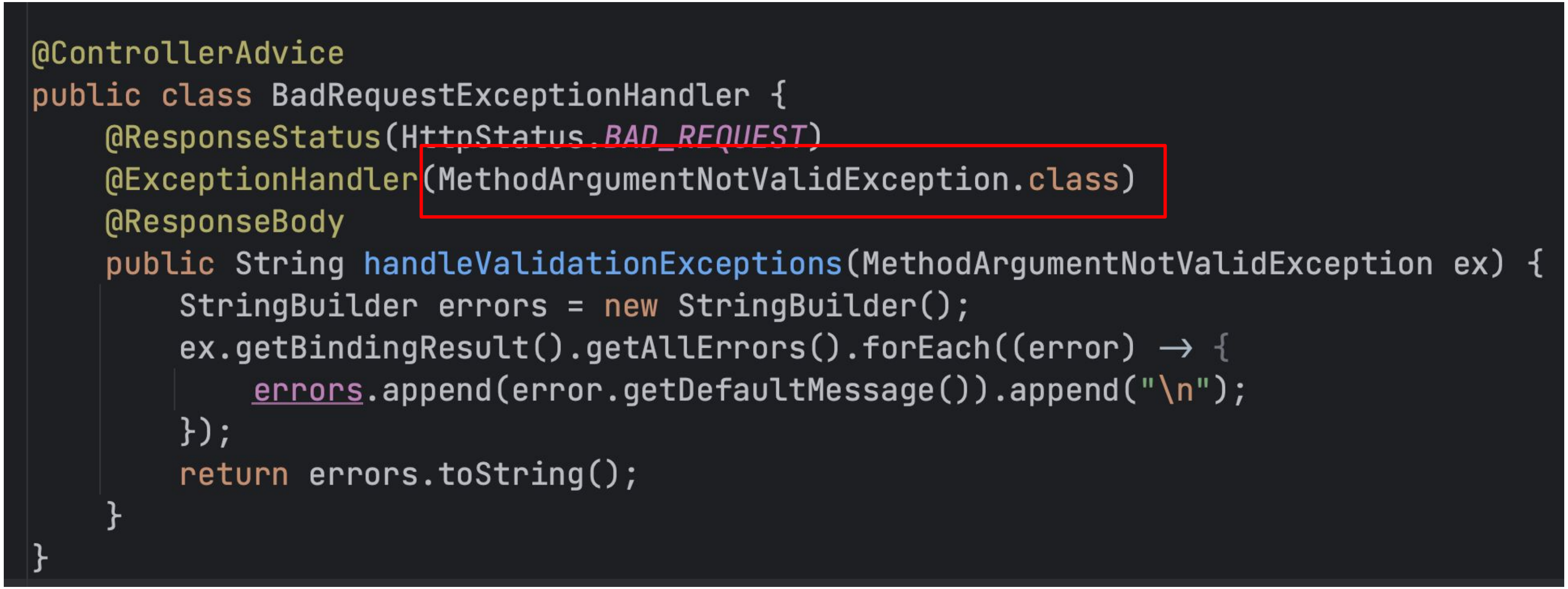
```
9 usages
public class User {

    5 usages
    @NotNull
    @NotBlank(message = "Name cannot blank")
    @Size(min=3, max = 10, message = "Name should be between 3 and 10 charac
    private String name;


    2 usages
    @NotNull
    @Min(value = 18, message = "Age should be greater than or equal to 18")
    private int age;
```

Allow Controller to do validation



```
@GetMapping("/users")
public List<User> getUsers() { return this.userService.get(); }

@PostMapping("/users")
public void createUsers(@Valid @RequestBody User user) {
    userService.create(user);
}

@DeleteMapping("/users/{name}")
public void delete(@PathVariable String name){
    userService.delete(name);
}
```

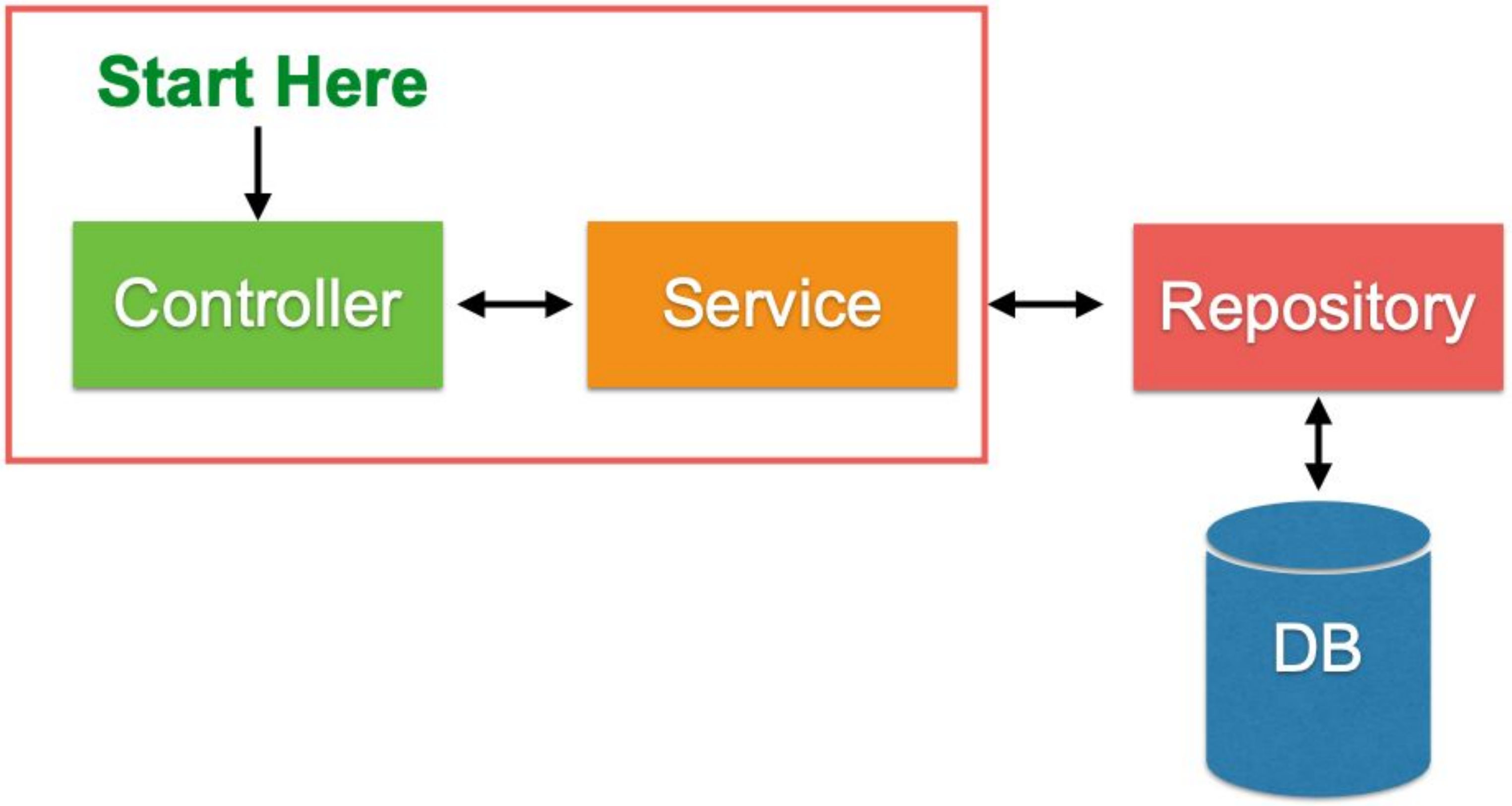
Advice for Error Response



```
@ControllerAdvice
public class BadRequestExceptionHandler {
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    @ResponseBody
    public String handleValidationExceptions(MethodArgumentNotValidException ex) {
        StringBuilder errors = new StringBuilder();
        ex.getBindingResult().getAllErrors().forEach((error) → {
            errors.append(error.getDefaultMessage()).append("\n");
        });
        return errors.toString();
    }
}
```

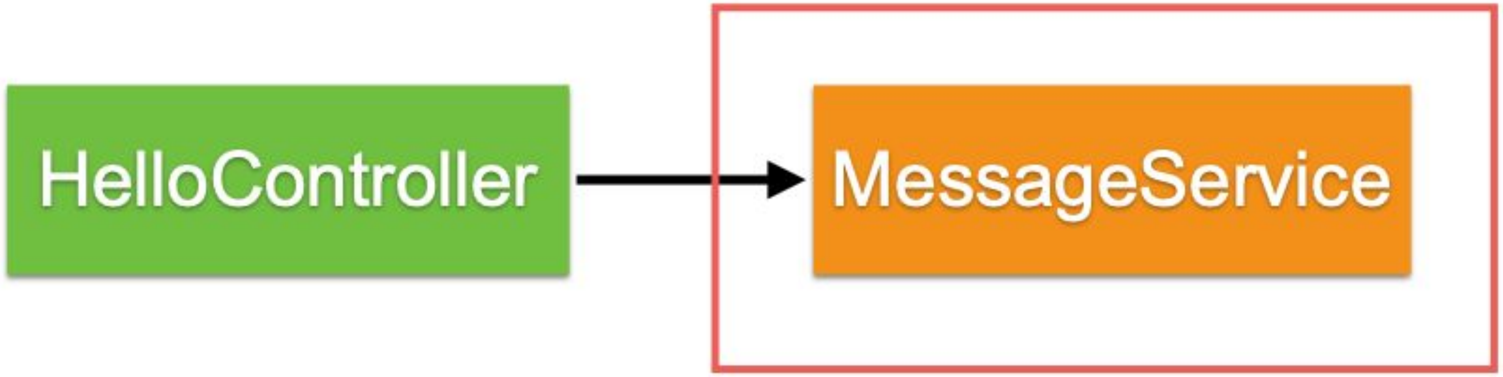
Move business logic to service

Working with service

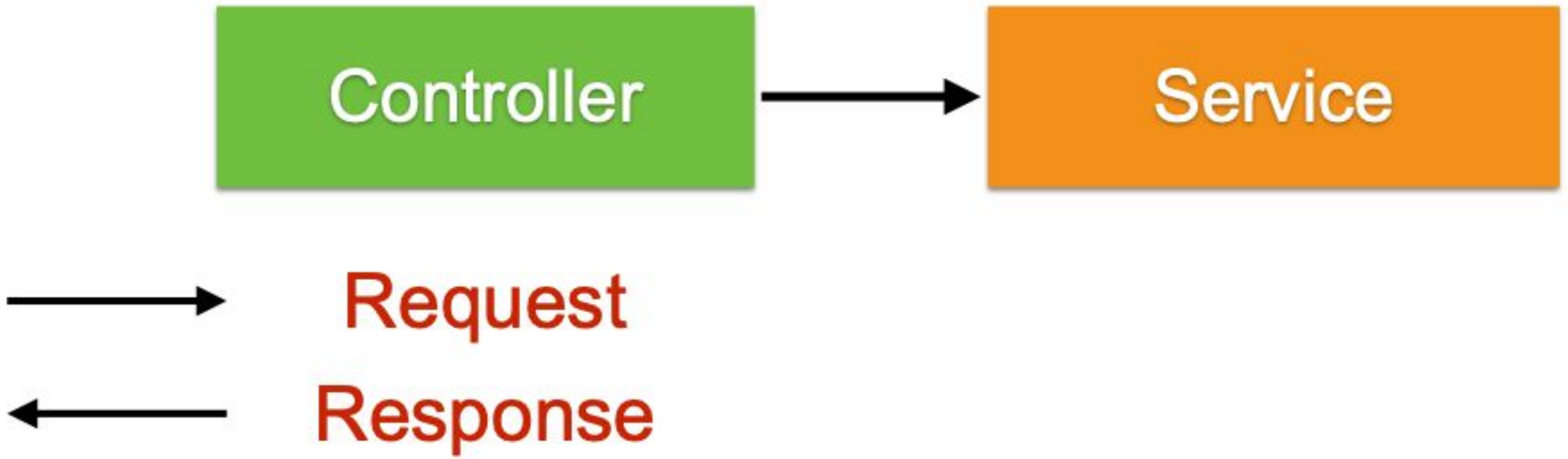


Move business logic to service

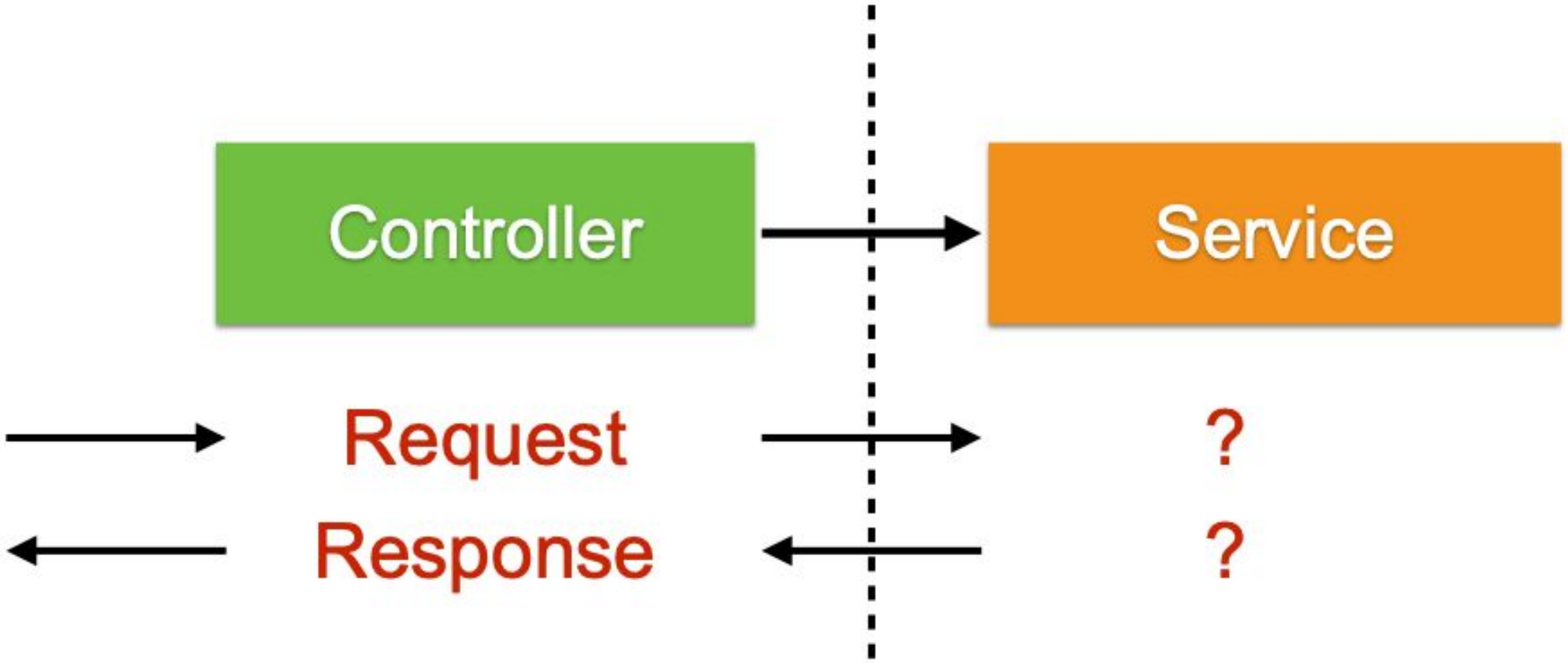
Service class or interface ?



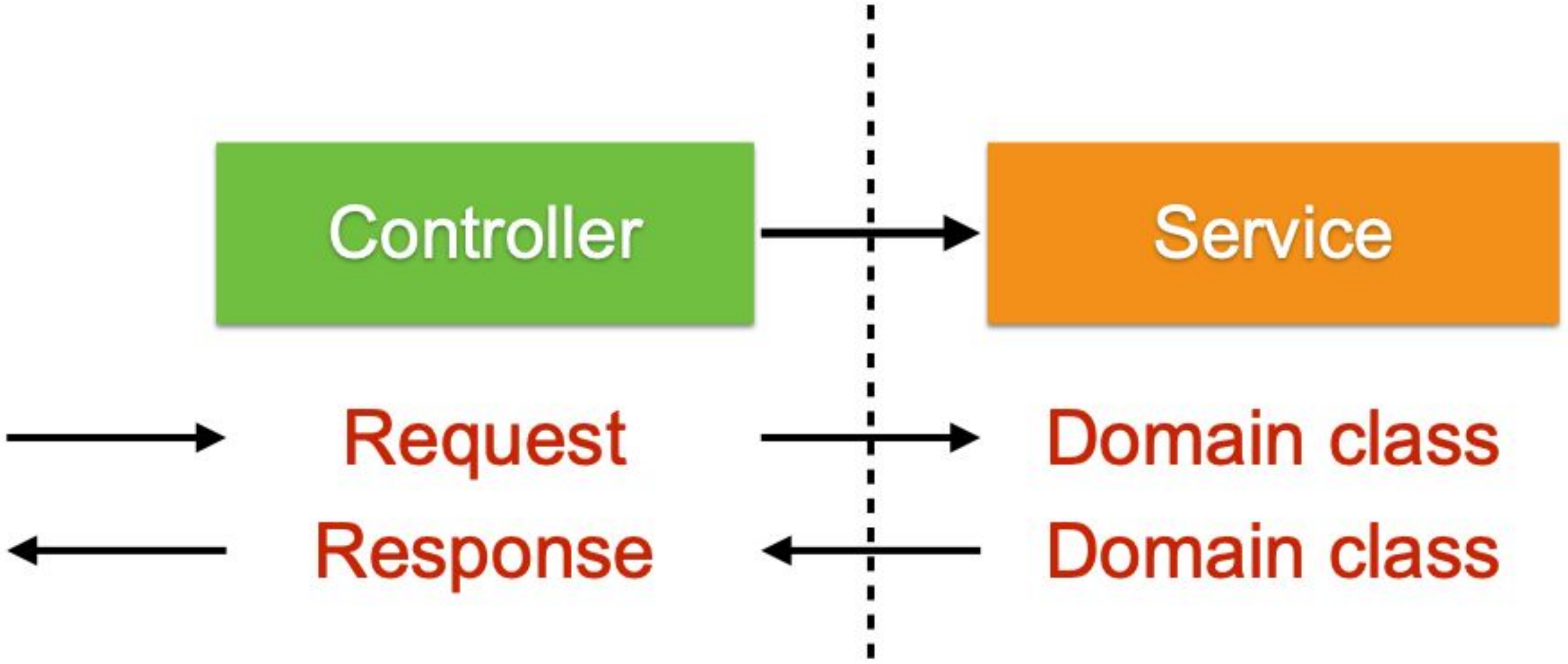
Data Model for service ?



Data Model ?

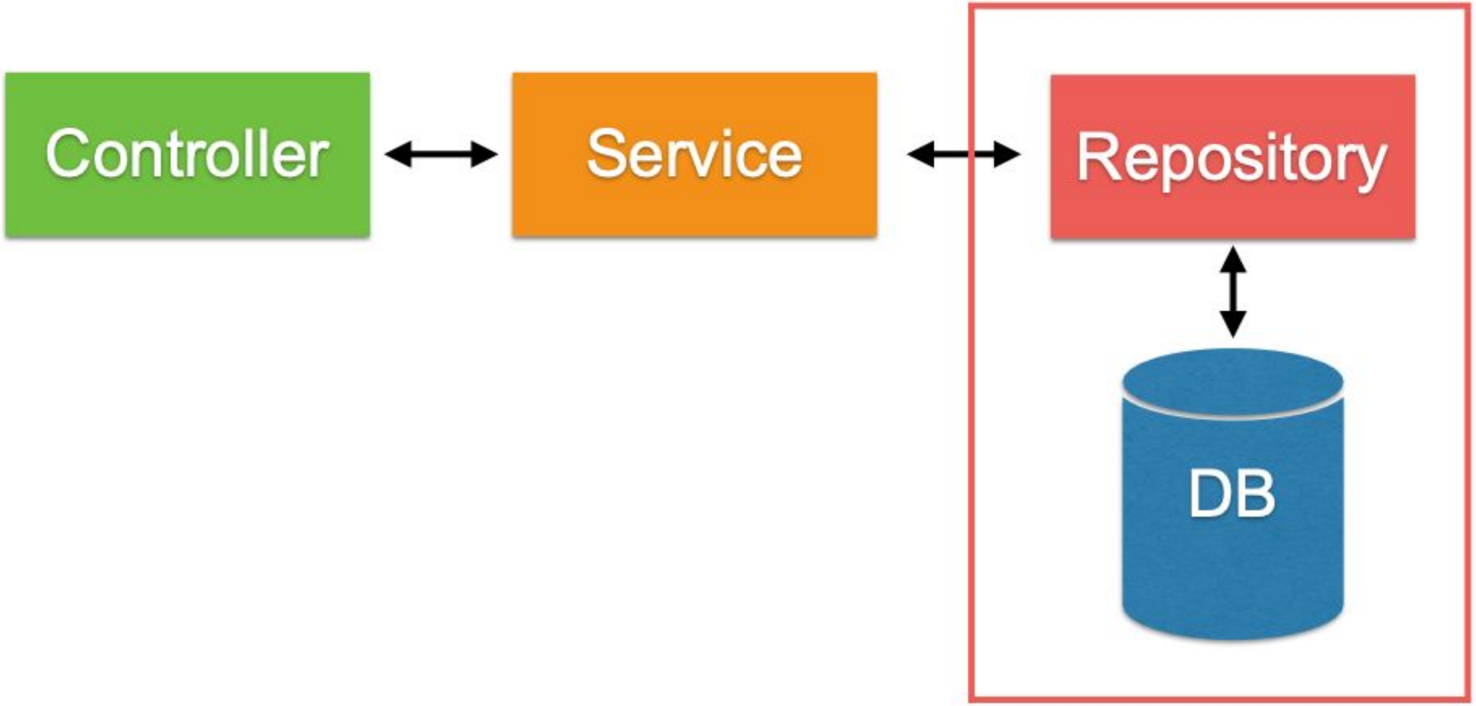


Data Model ?


Controller
Service
Request
Domain class
Response
Domain class

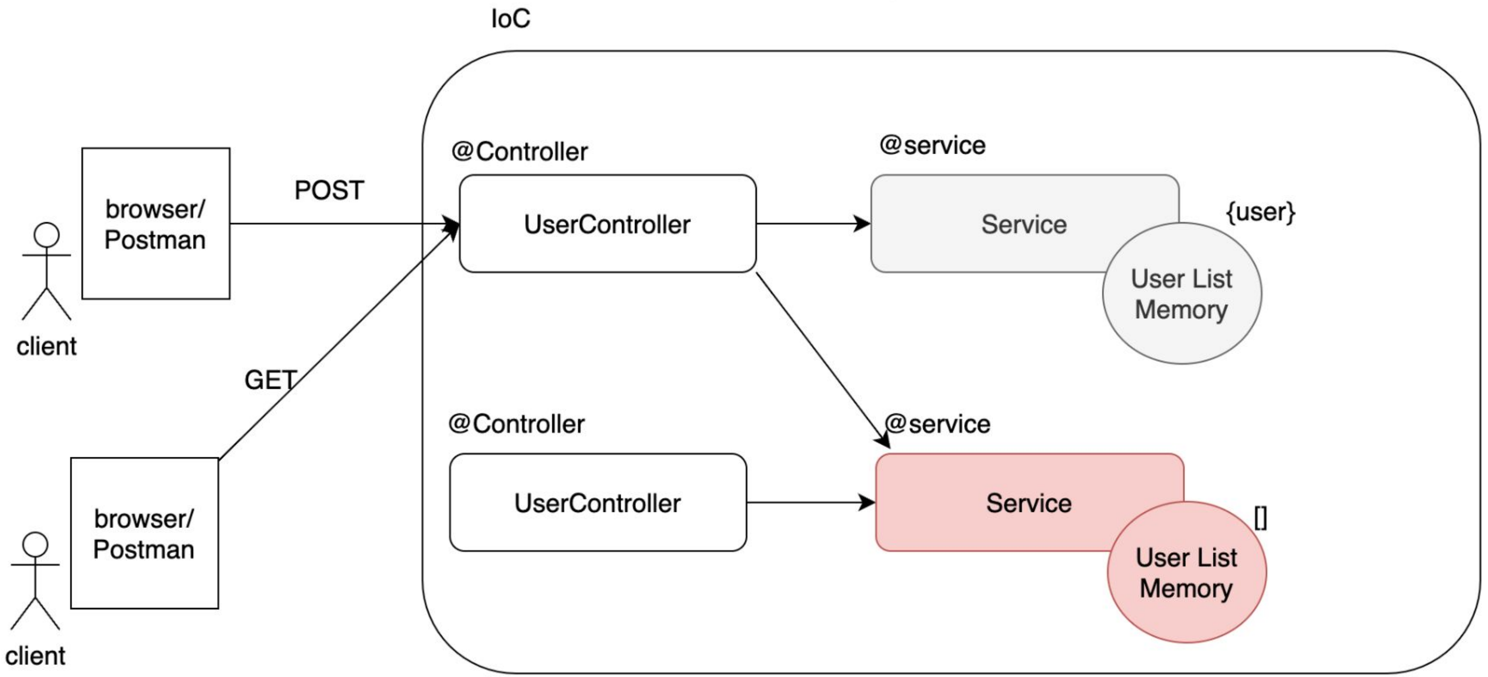
Working with Repository

Working with repository



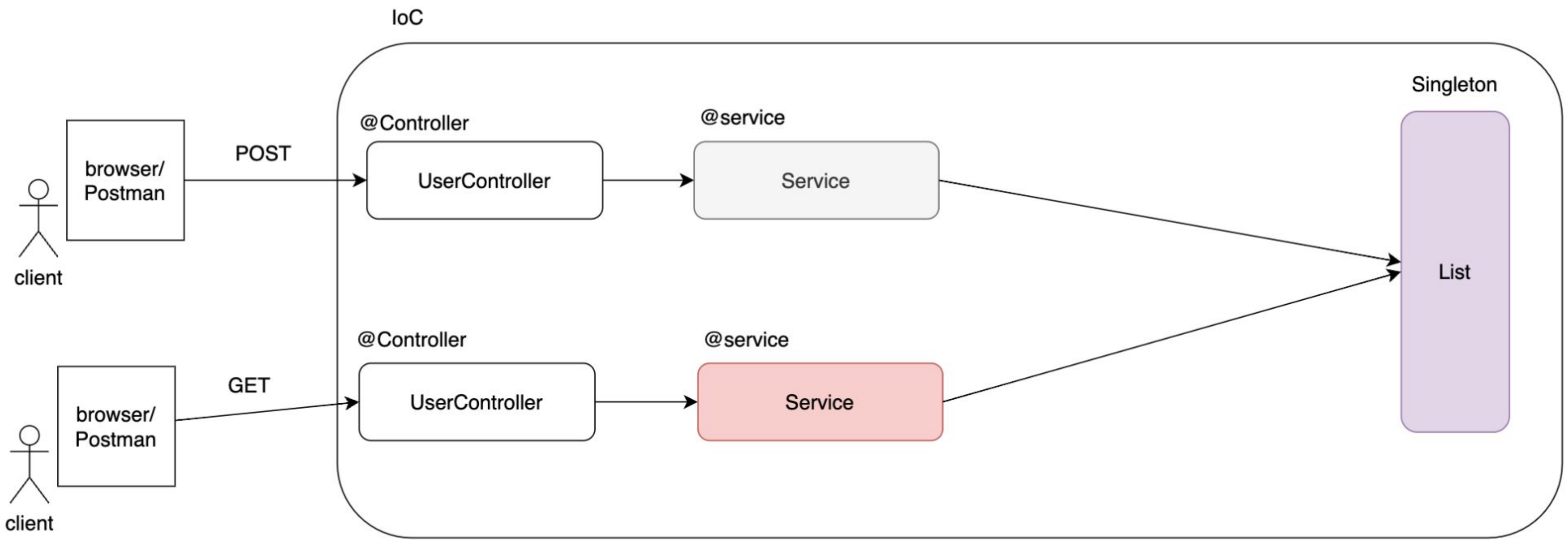
Bean not guarantee same instance

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

}
```



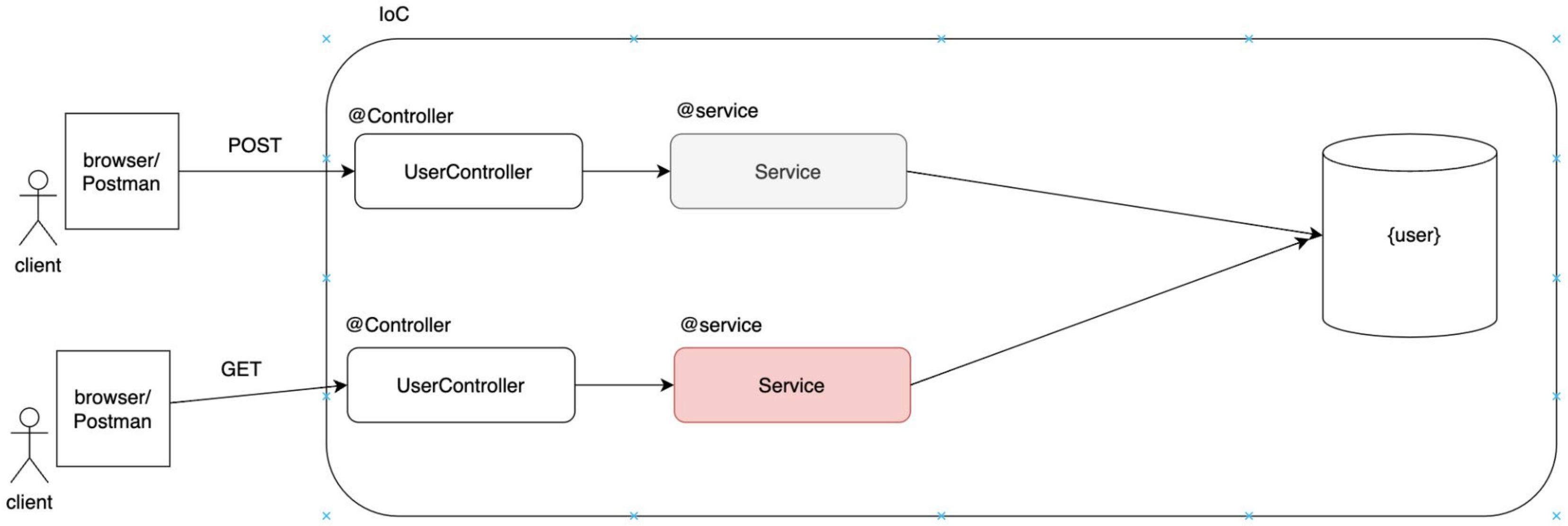
Share memory between service instance

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

}
```



Switch from in memory to persistence

```
@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

Design RESTFul APIs

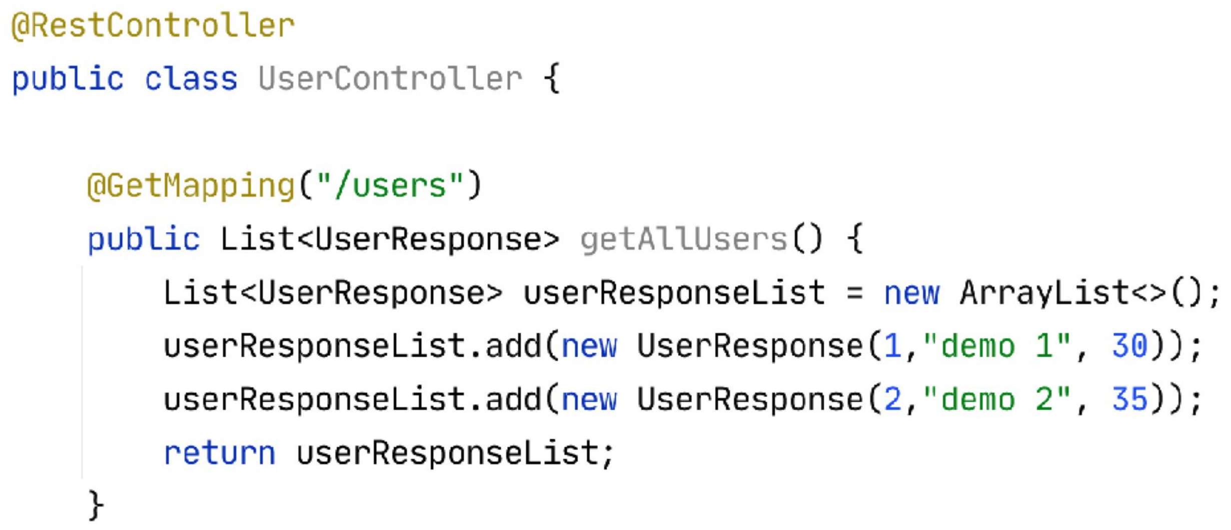
User story

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

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

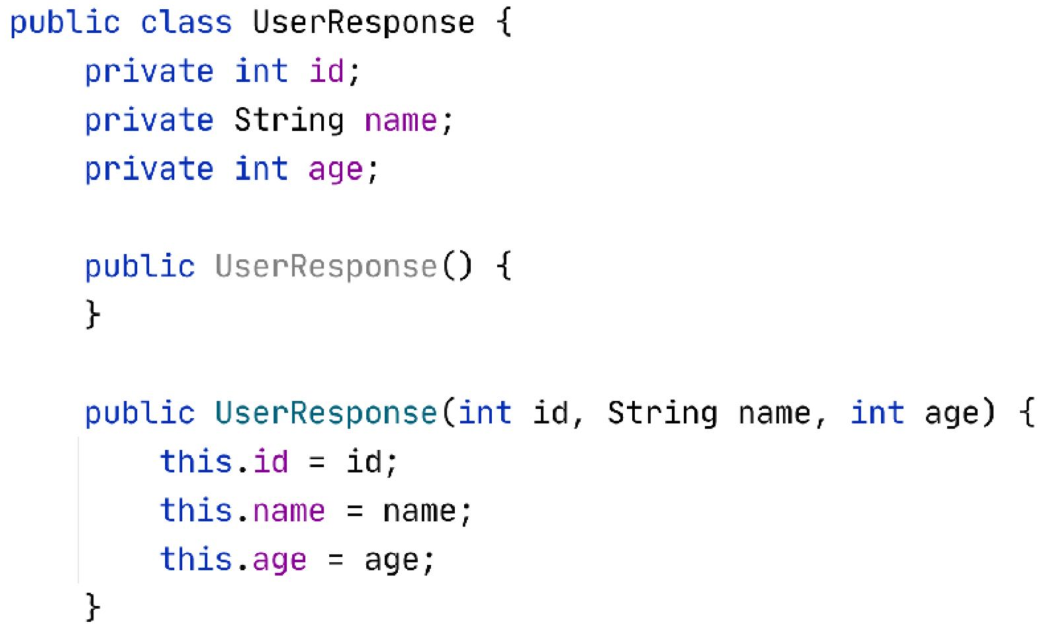
UserController.java



```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

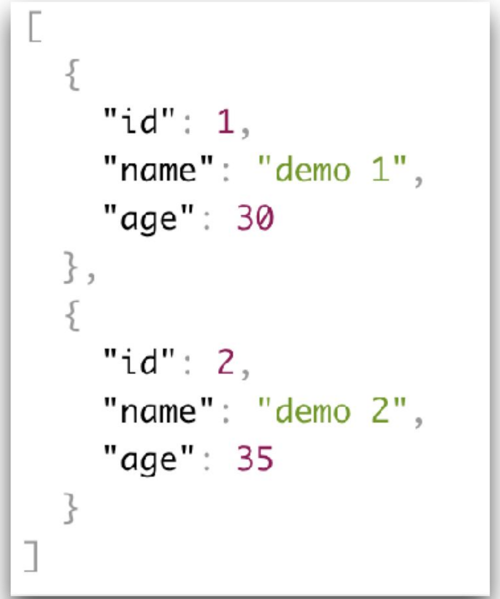
UserResponse.java



```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

```java
@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}

```
@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}

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

Take a break [Lunch]