Spring Boot REST API Development

By Ramesh Fadatare (Java Guides)

Spring Boot Application Architecture

POSTMAN
Controller Layer
Service Layer
Repository Layer
DB

Create and Setup Spring Boot Project in IntelliJ

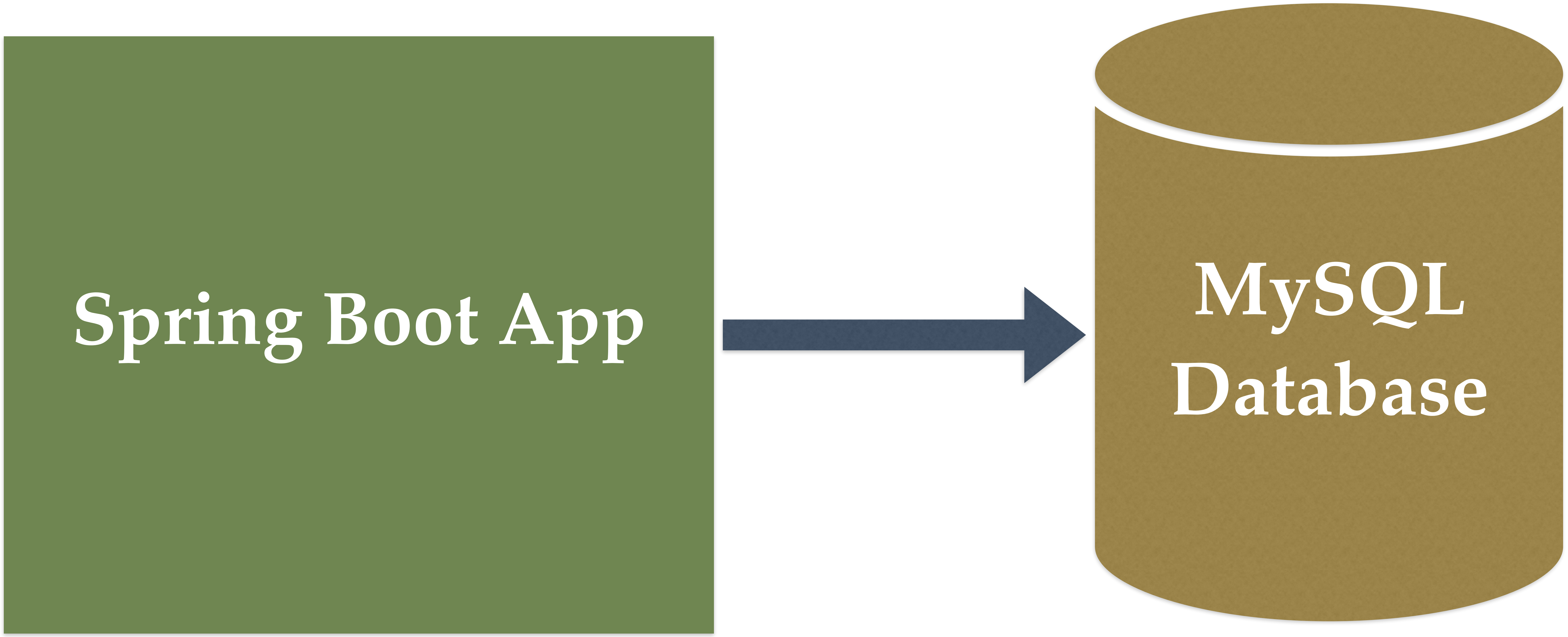
By Ramesh Fadatare (Java Guides)

Understanding spring-boot-starter-web Dependency

By Ramesh Fadatare (Java Guides)

Configure MySQL Database in Spring Boot App

By Ramesh Fadatare (Java Guides)


Spring Boot App
MySQL Database

Create User JPA Entity

By Ramesh Fadatare (Java Guides)

Create UserRepository Interface

By Ramesh Fadatare (Java Guides)

Build Create User REST API

By Ramesh Fadatare (Java Guides)

Build Get User By ID REST API

By Ramesh Fadatare (Java Guides)

Build Get All Users REST API

By Ramesh Fadatare (Java Guides)

Build Update User REST API

By Ramesh Fadatare (Java Guides)

Build Delete User REST API

By Ramesh Fadatare (Java Guides)

Spring Boot Application Architecture

DTO
POSTMAN
Controller Layer
Service Layer
Repository Layer
DB

Development Steps

1. Create UserDto class
2. Refactor Create User REST API to use DTO
3. Create UserMapper class
4. Refactor Get User By Id REST API to use DTO
5. Refactor Get All Users REST API to use DTO
6. Refactor Update User REST API to use DTO
7. Refactor Delete User REST API to use DTO

Using ModelMapper Library

JPA
DTO
DTO
POSTMAN
Controller Layer
Service Layer
Repository Layer
DB

Development Steps

1. Add ModelMapper Maven Dependency
2. Configure ModelMapper class as Spring bean
3. Inject and use ModelMapper Spring bean in Service class

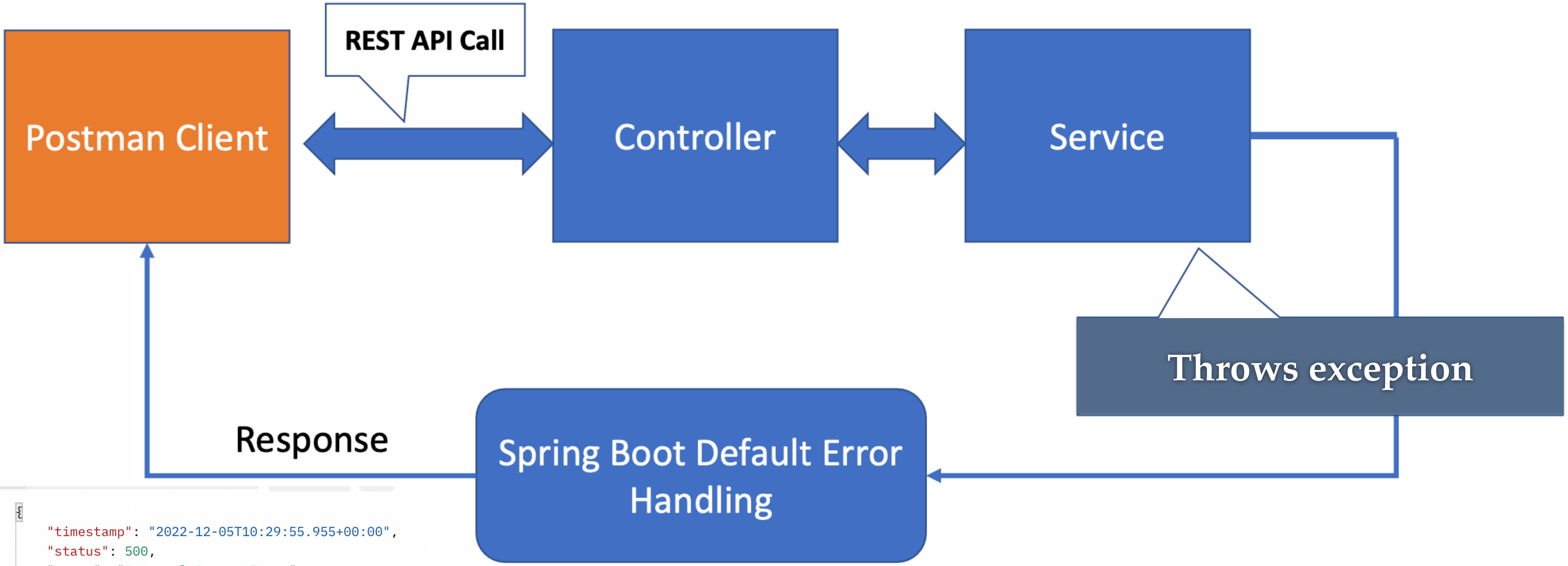
Using MapStruct Library

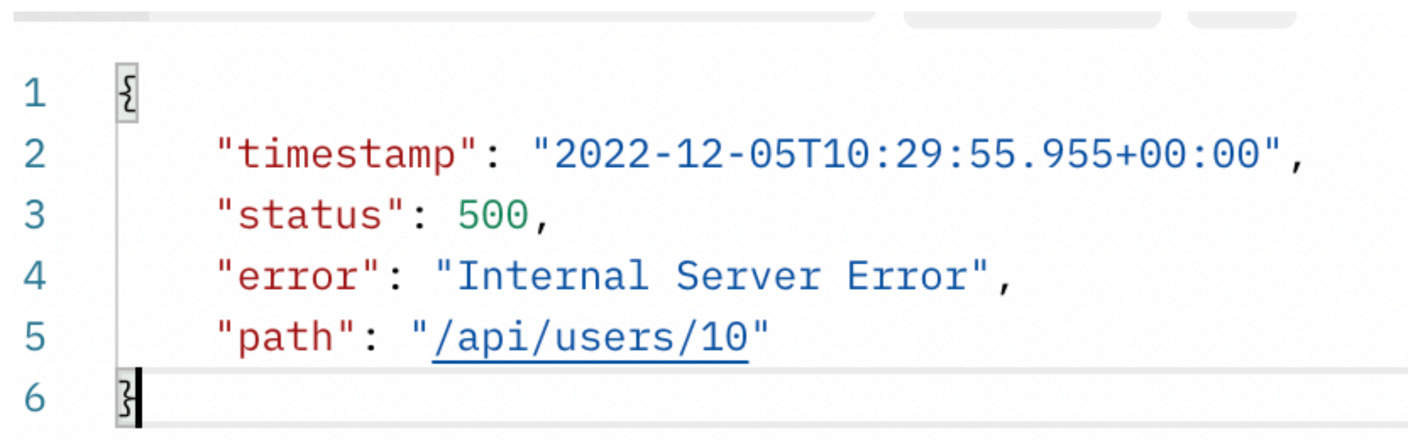
JPA
DTO
DTO
POSTMAN
Controller Layer
Service Layer
Repository Layer
DB

Development Steps

1. Add MapStruct Maven Dependency
2. Create AutoUserMapper using MapStruct
3. Use AutoUserMapper in UserServiceImpl Class

Spring Boot REST API Exception Handling



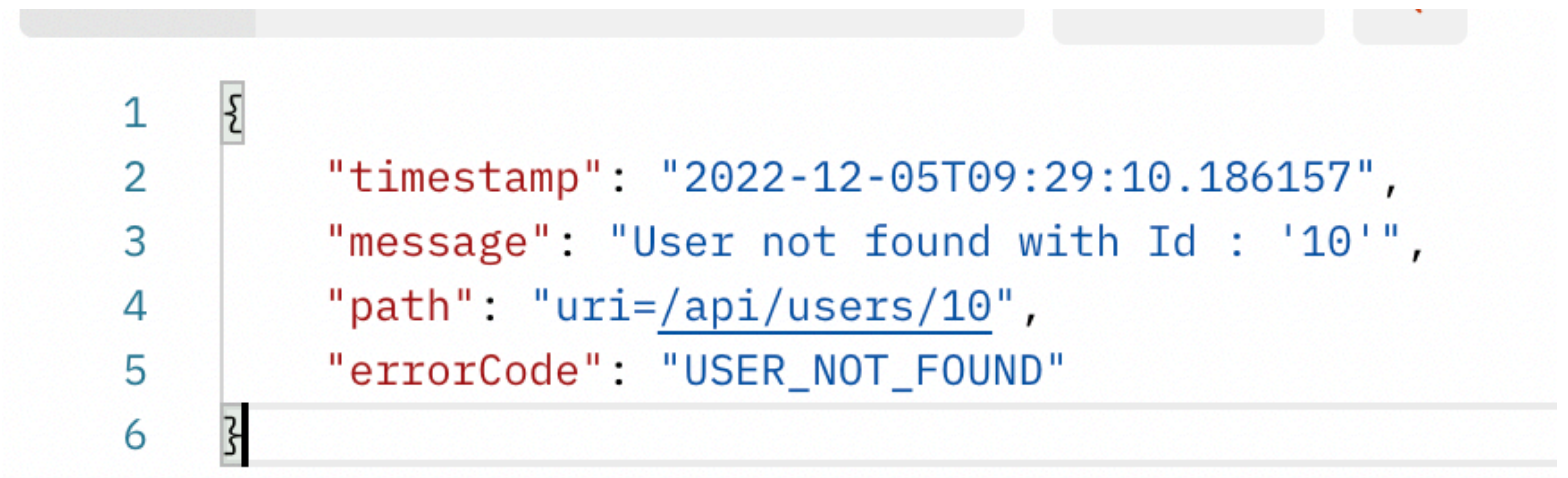


```
{
    "timestamp": "2022-12-05T10:29:55.955+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "path": "/api/users/10"
}
```

Create and Use Custom Exception

Requirement:

If User does not exist with given id in the database then we need to throw exception with proper error message and status code.



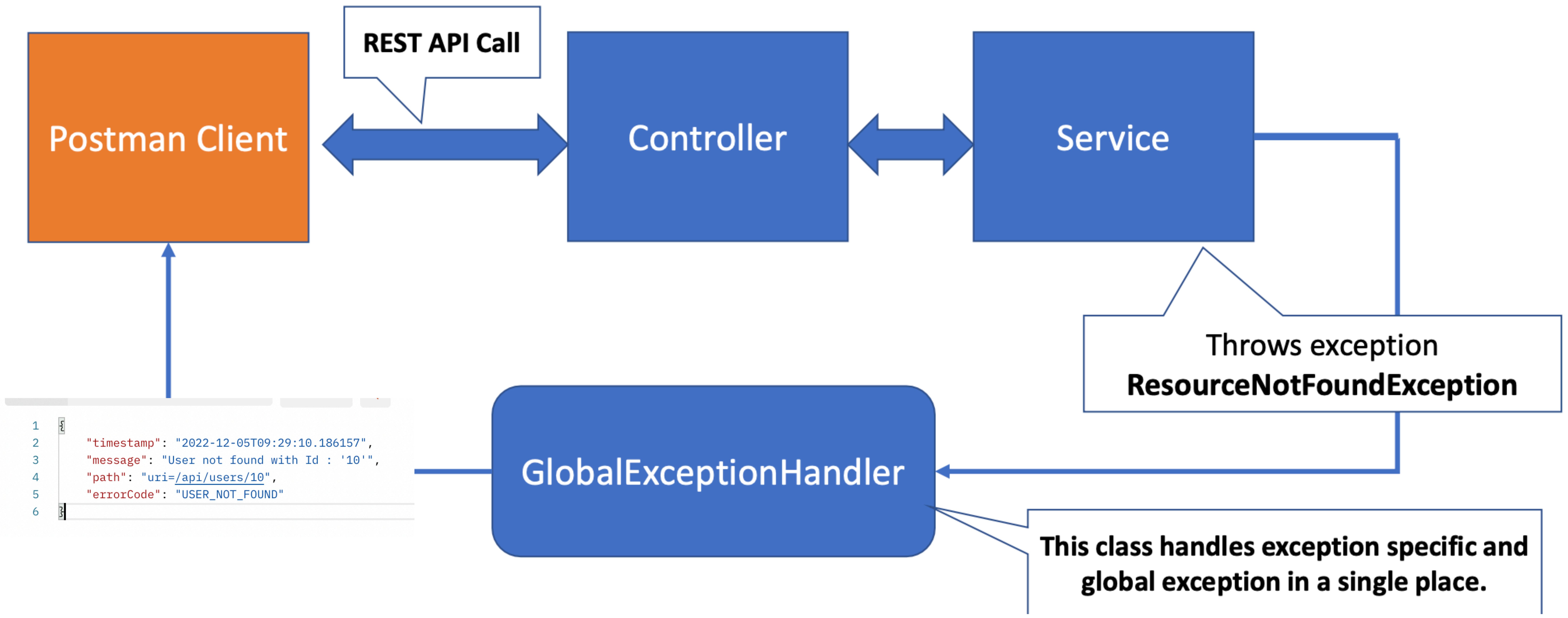
```
{
    "timestamp": "2022-12-05T09:29:10.186157",
    "message": "User not found with Id : '10'",
    "path": "uri=/api/users/10",
    "errorCode": "USER_NOT_FOUND"
}
```

Requirement

If User email already exists in the database then we need to throw exception with proper error message and status code.

EmailAlreadyExistsException

Spring Boot REST API Exception Handling



REST API Call
Postman Client
Controller
Service
Throws exception
ResourceNotFoundException
GlobalExceptionHandler
This class handles exception specific and global exception in a single place.

```
{
    "timestamp": "2022-12-05T09:29:10.186157",
    "message": "User not found with Id : '10'",
    "path": "uri=/api/users/10",
    "errorCode": "USER_NOT_FOUND"
}
```

Development Steps

1. Create and Use **ResourceNotFoundException** Custom Exception
2. Create ErrorDetails class to hold of the custom error response
3. Create GlobalExceptionHandler class to handle specific and global exceptions

Spring Boot REST API Request Validation

By Ramesh Fadatare (Java Guides)

Java Bean Validation API Overview

1. In Java, the **Java Bean Validation API** has become the de-facto standard for handling validations in Java projects.
2. **Hibernate Validator** is the reference implementation of the validation API.

Important Bean Validation Annotations

1. **@NotNull** validates that the annotated property value is not null.
2. @**Size** validates that the annotated property value has a size between the attributes min and max; can be applied to String, Collection, Map, and array properties.
3. **@Min** validates that the annotated property has a value not smaller than the value attribute.
4. **@Max** validates that the annotated property has a value no larger than the value attribute.
5. **@Email** validates that the annotated property is a valid email address.
6. **@NotEmpty** validates that the property is not null or empty; can be applied to String, Collection, Map, or Array values.
7. **@NotBlank** can be applied only to text values and validates that the property is not null or whitespace.

Validation in Spring Boot App

1. Spring Boot provides good integration with Hibernate validator for validating REST API requests
2. We will use Hibernate validator which is reference implementation of Java bean validation API
3. Adding validation is Spring boot app is super easy. We just have to add **spring-boot-starter-validation** dependency.

Validate Create and Update User REST API Requests

By Ramesh Fadatare (Java Guides)

Development Steps

1. Add Validation Dependency
2. Add Validation Annotations to UserDto
3. Enable Validation using @Valid Annotation on Create and Update REST APIs
4. Customize Validation Error Response and Send back to Client

Customizing Validation Error Response

By Ramesh Fadatare (Java Guides)

Spring Boot REST API Documentation Using SpringDoc Open API

By Ramesh Fadatare (Java Guides)

SpringDoc

springdoc-openapi java library helps to automate the generation of API documentation using spring boot projects.

springdoc-openapi java library provides integration between spring-boot and swagger-ui.

Automatically generates documentation in JSON/YAML and HTML format APIs.

This library supports:

- OpenAPI 3
- Spring-boot v3 (Java 17 & Jakarta EE 9)
- JSR-303, specifically for @NotNull, @Min, @Max, and @Size
- Swagger-ui
- OAuth 2

This is a community-based project, not maintained by the Spring Framework Contributors

Development Steps

1. Adding **springdoc-openapi** Maven dependency
2. Defining General API Information (Using Annotations)
3. Customizing Swagger API Documentation with Annotations
4. Customizing Swagger Models Documentation with Annotations