

# 03 - RESTful APIs and API Testing

## Table of Contents

- [Agenda](#)
  - [What is an API? \(Application Programming Interface\)](#)
  - [What is a RESTful API? \(Representational State Transfer API\)](#)
  - [MVC \(Model, Views, and Controller\)](#)
    - [Difference between `@Controller` and `@RestController`:](#)
  - [Pseudocode](#)

## Agenda

- Client Server Model
- API's
- REST Api's
- MVC (Model, Views and Controllers)
- How to implement REST API's in MVC
- HTTP Methods for GET, POST, DELETE
- How to use Postman for API's Testing

## What is an API? (Application Programming Interface)

- **Imagine:**
  - A waiter in a restaurant.
- **What it does:**
  - Takes your order (request) and brings you your food (response).
- **In tech:**
  - It's a way for different software programs to talk to each other and share information.
- **Example:**
  - When you use an app to check the weather, the app uses an API to get weather data from another service.

## What is a RESTful API? (Representational State Transfer API)

- **Imagine:**
  - A very organised and friendly waiter who follows specific rules.

- **What it does:**
  - Takes your order in a standard way and brings you exactly what you asked for, efficiently and neatly.
- **In tech:**
  - It's a type of API that follows specific rules (REST principles) to make it easy and reliable for programs to communicate.
- **Example:**
  - When you post a photo on social media, the app uses a RESTful API to send your photo to the server in a structured way.

## MVC (Model, Views, and Controller)

Model <code>@Entity</code>	Controller <code>@Services</code>	Views <code>@Controller</code>
<p>What it is: The characters and props in the play.</p> <p>In your project: These are your entity classes, like a <code>User</code>, which represents data in your database.</p> <p>Example: The <code>User</code> entity class might have details like the student's name, age, and grade.</p>	<p>What it is: The director of the play.</p> <p>In your project: These are your service classes, like <code>UserService</code>, which contain the main instructions and logic for how the play runs.</p> <p>Example: The <code>UserService</code> class has the logic for what happens when a student is added, updated, or retrieved from the list.</p>	<p>What it is: The script and performance on stage.</p> <p>In your project: These are your controller classes, like <code>UserController</code>, which handle what the audience (users) sees and interacts with.</p> <p>Example: The <code>UserController</code> class manages user-related actions, like when someone asks for a list of students or adds a new student.</p>

## Difference between `@Controller` and `@RestController`:

Feature	<code>@Controller</code>	<code>@RestController</code>
<b>Purpose</b>	Handles web pages (HTML)	Handles RESTful web services (JSON/XML)
<b>Response Type</b>	Returns views (like JSP, Thymeleaf)	Returns data (JSON/XML) directly
<b>Annotation</b>	<code>@Controller</code>	<code>@RestController</code>
<b>ResponseBody</b>	Needs <code>@ResponseBody</code> for data response	Implicitly includes <code>@ResponseBody</code>

Feature	@Controller	@RestController
Use Case	Traditional web applications	RESTful API services
Example	Online portal with HTML pages	Weather service providing JSON data

## Pseudocode

1. Create entities, services, and controllers directory
2. Create `User.java` entity inside `entities` directory
  1. Add fields `Long id`, `String username`, `String phoneNumber`, `String email`
  2. Add `Constructor` and `Getter/Setter`
3. Create `UserService.java` inside `services` directory
  1. Add methods `addUser()`, `getAllUsers()`, `getUserById()`, `updateUser()`, `deleteUser()`
4. Create `UserController.java` inside `controllers` directory
  1. Add annotations `@RestController` to class
  2. Add annotation `@RequestMapping('/user')` to class
  3. Instantiate with `UserService`

```
@Autowired
private UserService userService;
```

## 5. Add Mappings

```
@PostMapping("/add")
addUser(@RequestBody)

@GetMapping
getAllUsers()

@GetMapping("/{id}")
getUserById(@PathVariable)

@PutMapping("/{id}")
updateUser(@PathVariable, @RequestBody)

@DeleteMapping("/{id}")
deleteUser(@PathVariable)
```