03 - RESTful APIs and API Testing

Table of Contents

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)

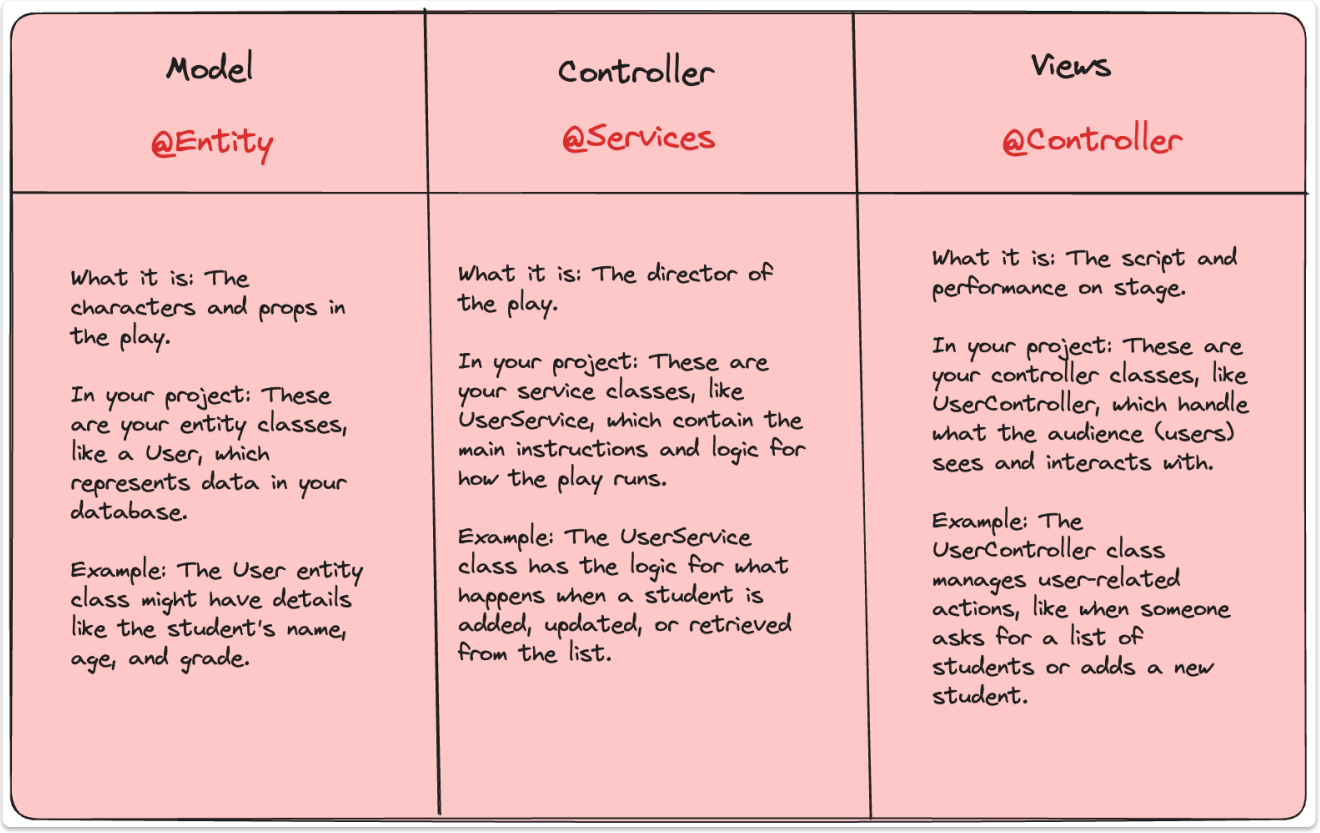
- Imagine:
  - A waiter in a restaurant.
- What it does:
  - Takes your order (request) and brings you your food (response).
- In tech:
  - It's a way for different software programs to talk to each other and share information.
- Example:
  - When you use an app to check the weather, the app uses an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



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

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

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 `@RestContoller` 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)
```