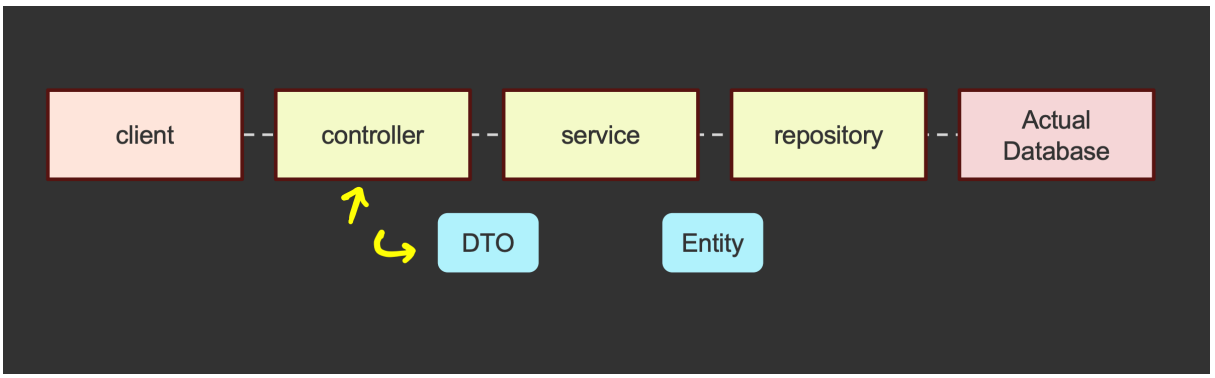
Presentation Layer

Spring Boot Web Project Structure



Annotated Controllers

Spring MVC provides an annotation-based programming model where `@Controller` and `@RestController` components use annotations to express request mappings, request input, exception handling, and more.

The `@RestController` annotation is a shorthand for `@Controller` and `@ResponseBody`, meaning all methods in the controller will return JSON/XML directly to the response body.

Request Mappings

You can use the `@RequestMapping` annotation to map requests to controllers methods. It has various attributes to match by URL, HTTP method, request parameters, headers, and media types.

There are also HTTP method specific shortcut variants of `@RequestMapping`:

- `@GetMapping`
- `@PostMapping`
- `@PutMapping`
- `@DeleteMapping`
- `@PatchMapping`

Dynamic URLs Paths

`@PathVariable`	`@RequestParam`
/employees/123	*/employees?id=123*
Use path variables when the parameter is an essential part of the URL path that identifies a resource.	Use query parameters when the parameter is optional and used for filtering, sorting, or other modifications to the request.

RequestBody

`@RequestBody` is used to bind the HTTP request body to a Java object. When a client sends data in the body of a request (e.g., JSON or XML), `@RequestBody` maps this data to a Java object.

Use Case:

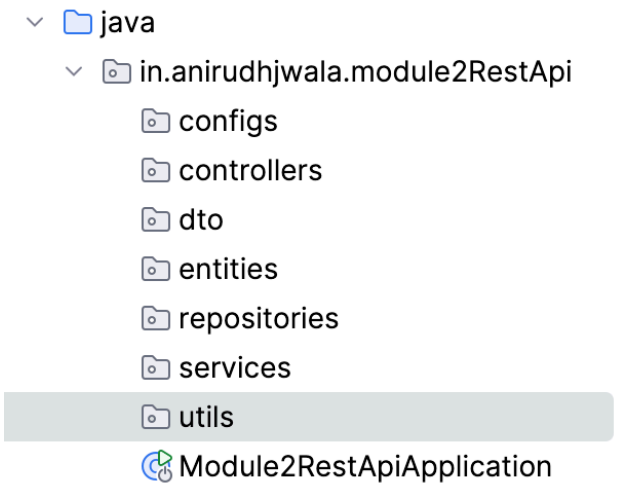
- Typically used in POST, PUT, and PATCH methods where the client sends data that needs to be processed by the server.

- Converts JSON or XML data from the request body into a Java object using a message converter (e.g., Jackson for JSON).

Generate a project

Link → https://start.spring.io/#!type=maven-project&language=java&platformVersion=4.0.0&packaging=jar&configurationFileFormat=properties&jvmVersion=25&grc

Use the following folder structure to maintain the code files



Create a class under controllers called `EmployeeController.java`

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmployeeController {

    @GetMapping(path = "/getSecretMessage")
    public String getMySuperSecretMessage() {
        return "Secret Message :: 13AS89$2#239";
    }

}
```

Whoever tries to access they get the message back

`http://localhost:8080/getSecretMessage`

These are simple strings and works fine, now to work with a Java object that can be sent in a JSON format we will declare a DTO of `EmployeeDTO` with following details:

```
package in.anirudhjwala.module2RestApi.dto;

import java.time.LocalDate;

// POJO Class
public class EmployeeDTO {
    private long id;
    private String name;
    private String email;
    private Integer age;
    private LocalDate dateOfJoining;
    private boolean isActive;

    public EmployeeDTO() {}

    public EmployeeDTO(long id, String name, String email, Integer age, LocalDate dateOfJoining, boolean isActive)
{
        this.id = id;
        this.name = name;
        this.email = email;
        this.age = age;
        this.dateOfJoining = dateOfJoining;
        this.isActive = isActive;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Integer getAge() {
        return age;
    }
```

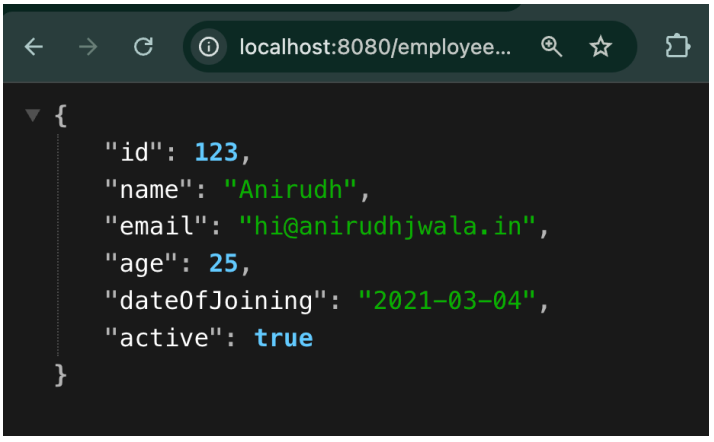
```
    public void setAge(Integer age) {
        this.age = age;
    }

    public LocalDate getDateOfJoining() {
        return dateOfJoining;
    }

    public void setDateOfJoining(LocalDate dateOfJoining) {
        this.dateOfJoining = dateOfJoining;
    }

    public boolean isActive() {
        return isActive;
    }

    public void setActive(boolean active) {
        isActive = active;
    }
}
```

Let's declare and provide a new endpoint of GET to fetch an employee details by their ID. We pick the route in which employee id value is passed in the path of the URL requested and hence we have following snippet

```
@GetMapping(path = "/employees/{employeeId}")
public EmployeeDTO getEmployeeById(@PathVariable Long employeeId) {
    return new EmployeeDTO(employeeId, "Anirudh", "hi@anirudhjwala.in", 25, LocalDate.of(2021, 3, 4),true);
}
```

Jackson does automatic conversion of the details from Java object to a JSON format that works for client.

http://localhost:8080/employees/123

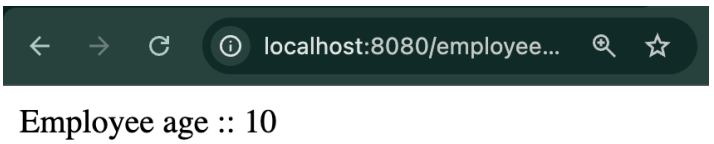


To pass values as **query** parameters we can do via Request parameters that are defined for a route to access say age of an employee. If we don't mention anything then by default these are mandatory to pass,

```
@GetMapping(path = "/employees")
public String getEmployees(@RequestParam Integer age) {
```

```
    return "Employee age :: " + age;
}
```

http://localhost:8080/employees?age= 10

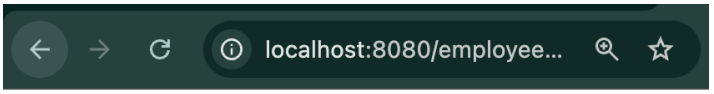


Employee age :: 10

To capture optionally then we can use something like below:

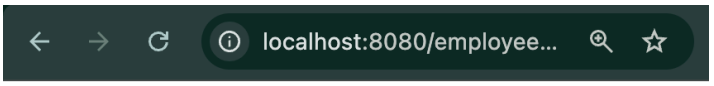
```
@GetMapping(path = "/employees")
public String getEmployees(
    @RequestParam(required = false) Integer age,
  @RequestParam(required = false) String sortBy
) {
    return "Employee age :: " + age + " sort by :: " + sortBy;
}
```

http://localhost:8080/employees?age=25



Employee age :: 25 sort by :: null

http://localhost:8080/employees?age=25&sortBy=name



Employee age :: 25 sort by :: name

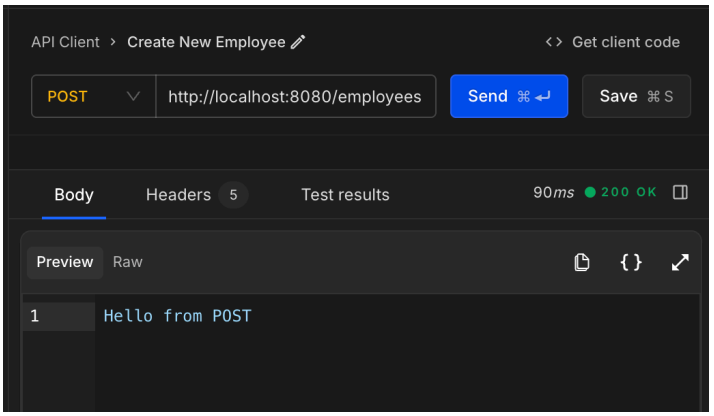
Since all the employees have a shared pre-fix path we can move it to the top-level such like this

```
@RestController
@RequestMapping("/employees")
public class EmployeeController {

    @GetMapping(path = "/getSecretMessage")
    public String getMySuperSecretMessage() {
        return "Secret Message :: 13AS89$2#239";
    }

    @GetMapping(path = "/{employeeId}")
    public EmployeeDTO getEmployeeById(@PathVariable Long employeeId) {
        return new EmployeeDTO(employeeId, "Anirudh", "hi@anirudhjwala.in", 25, LocalDate.of(2021, 3, 4),true);
```

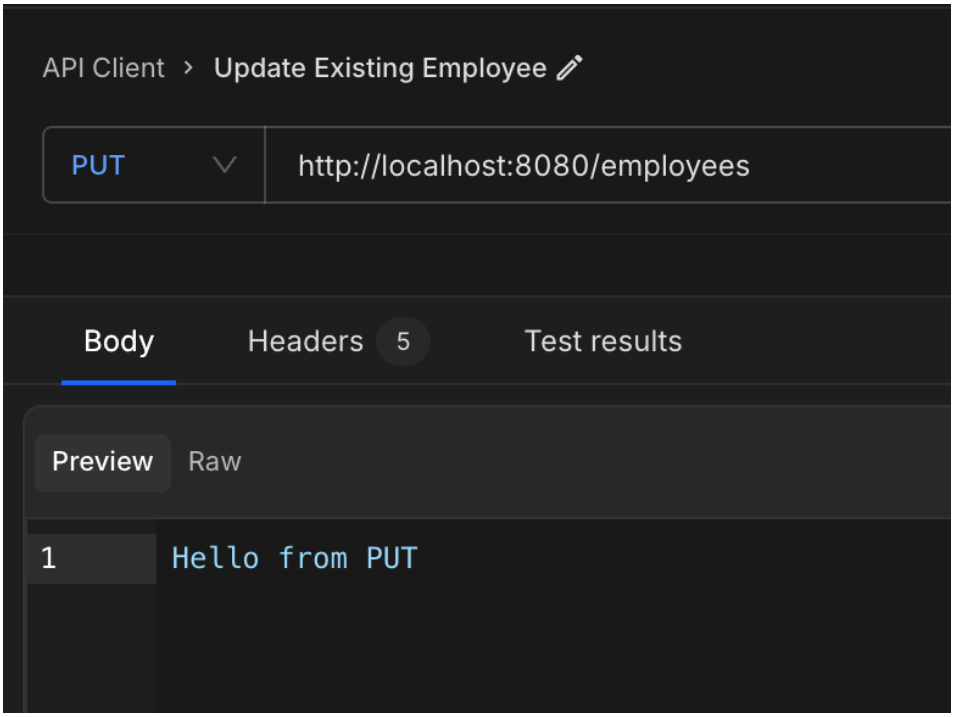
```
    }

    @GetMapping(path = "/")
    public String getEmployees(
            @RequestParam(required = false) Integer age,
            @RequestParam(required = false) String sortBy
    ) {
        return "Employee age :: " + age + " sort by :: " + sortBy;
    }
}
```

Now onto the next kind of mapping which is `@PostMapping` that is used for creation of new resource within our persistence layer.

```
@PostMapping
public String createNewEmployee() {
    return "Hello from POST";
}

@PutMapping
public String updateEmployeeById() {
    return "Hello from PUT";
}
```

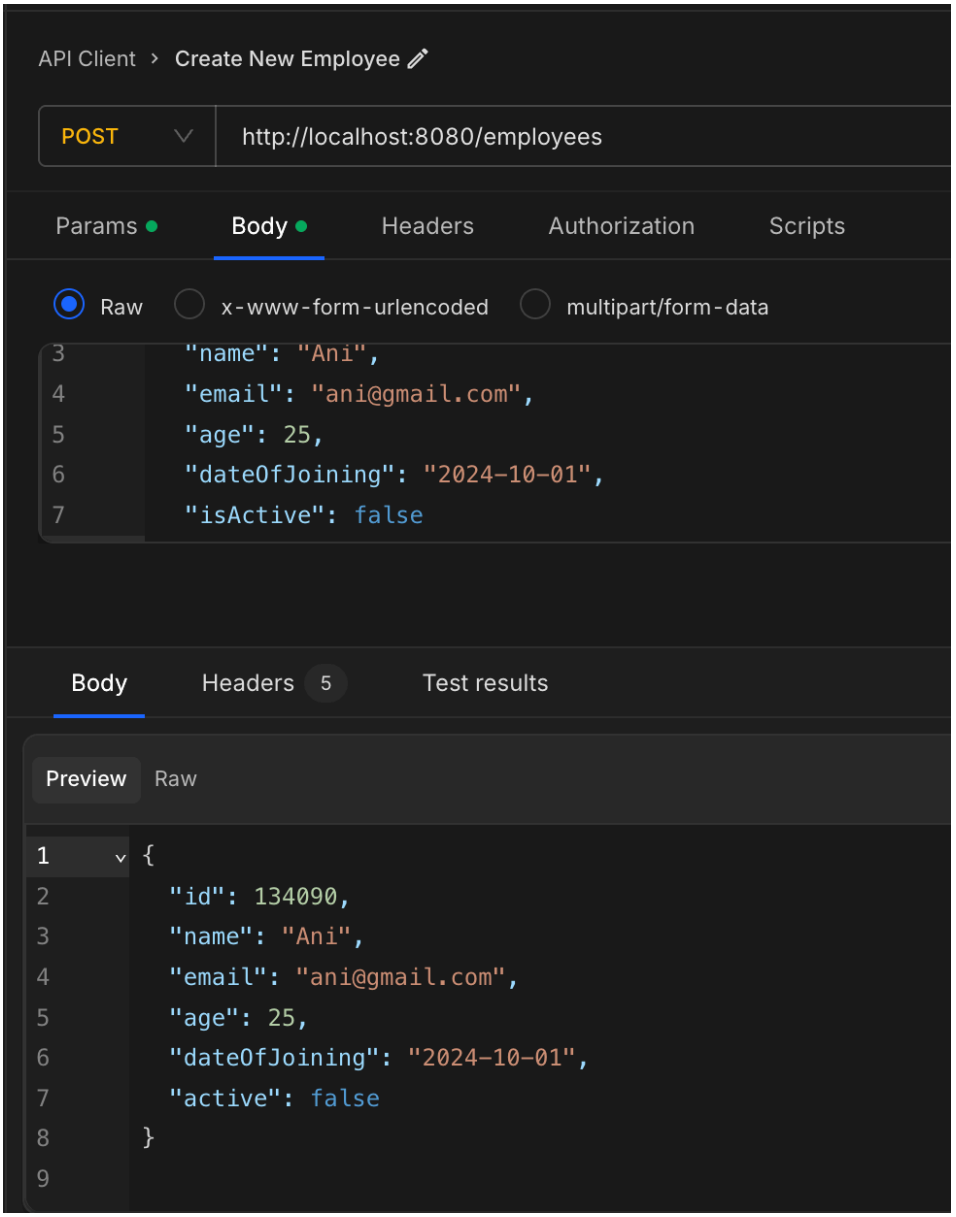


We have another kind of HTTP type for updating an exisitng record



Now lets pass a value in the request body and send the details to make a new Employee object

```
@PostMapping
public EmployeeDTO createNewEmployee(@RequestBody EmployeeDTO inputEmployee) {
    inputEmployee.setId(134090);
  return inputEmployee;
}
```


API Client
Create New Employee
POST
http://localhost:8080/employees
Params
Body
Headers
Authorization
Scripts
Raw
x-www-form-urlencoded
multipart/form-data
"name": "Ani",
"email": "ani@gmail.com",
"age": 25,
"dateOfJoining": "2024-10-01",
"isActive": false
Body
Headers 5
Test results
Preview
Raw
{
"id": 134090,
"name": "Ani",
"email": "ani@gmail.com",
"age": 25,
"dateOfJoining": "2024-10-01",
"active": false
}