How to Run and Test the Task Management Application:

This document describes how to execute and test the application of Spring Boot for Task Management: it exposes a RESTful API that is used to handle the task by providing different endpoints for creating, updating, retrieving, and deleting tasks. It also provides very basic authentication for securing the API.

1. Prerequisites:

Install the following to run the application:

- Java Development Kit (JDK) 11 or above
- Maven: dependencies management and project building
- Spring Boot: if you do not use Maven for running directly the app
- IDE: IntelliJ IDEA, Eclipse, and Visual Studio Code
- Postman and/or cURL: optional - test your API endpoints

I PERSONALLY USE ECLIPSE TO IMPORT THE MAVEN PROJECT AND IT BUILD AUTOMATICALLY.

2. Project Configuration in Eclipse:

Before running the Spring Boot application, ensure that your project is configured correctly in Eclipse:

1.Import the Project:

- Open Eclipse IDE.
- Go to File > Import > Existing Maven Projects
- Navigate to your project folder, select it, Then click Finish.

2.Build the Project:

- After you import the project, Maven will automatically build it, and you can check if everything is all right by looking at errors or warnings in the Console view inside Eclipse.

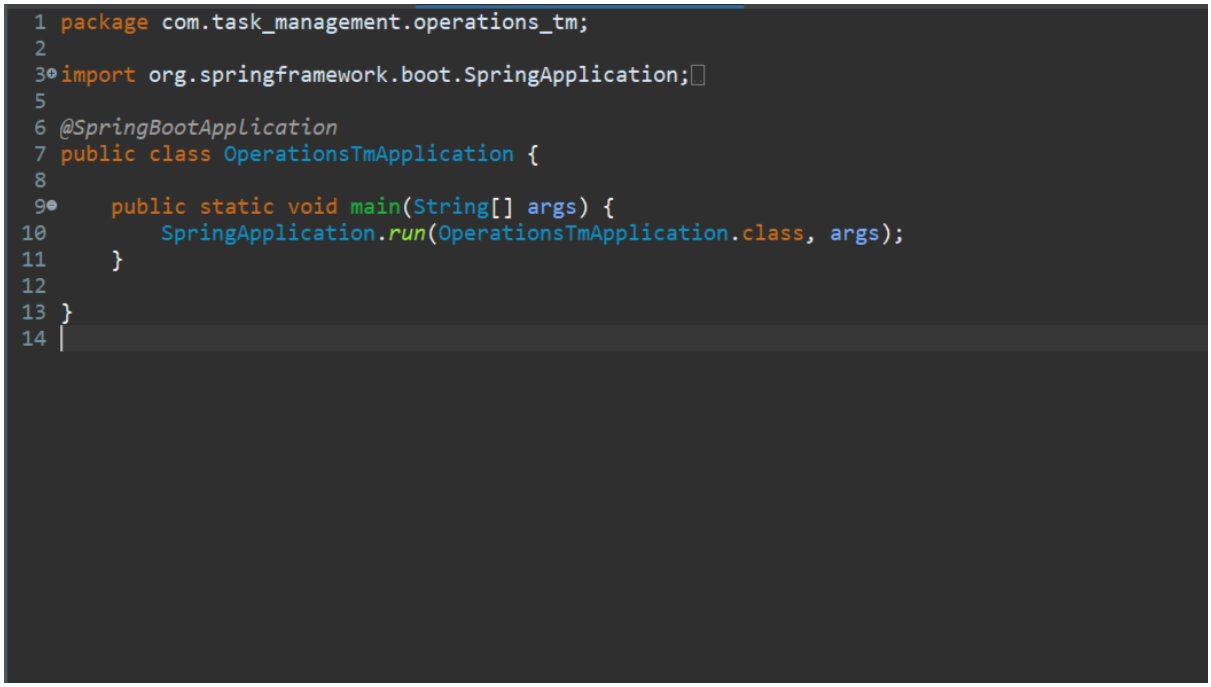
- If it doesn't start building automatically, you can try forcing it: right-click on your project inside the Project Explorer>click on Maven>Update Project.

2. Run the Spring Boot Application from Eclipse:

1. **Run the Application**:

   - Right-click on the main class `OperationsTmApplication.java` (which contains the `main()` method).

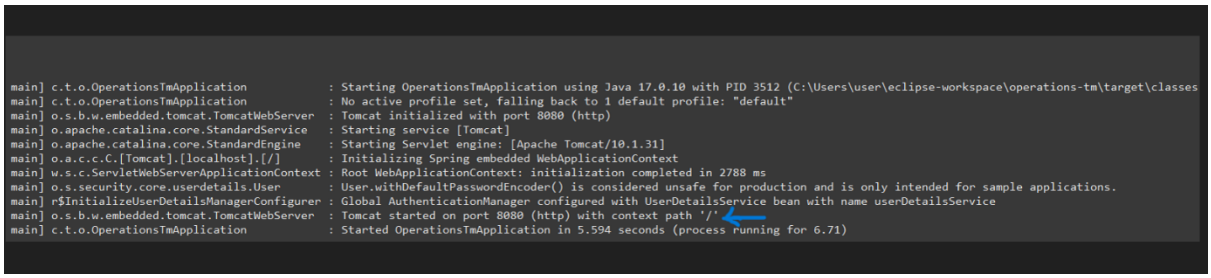
   - Select **Run As > Spring Boot App**. This will start the Spring Boot application in your Eclipse environment.



```java
package com.task_management.operations_tm;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class OperationsTmApplication {

    public static void main(String[] args) {
        SpringApplication.run(OperationsTmApplication.class, args);
    }

}
```

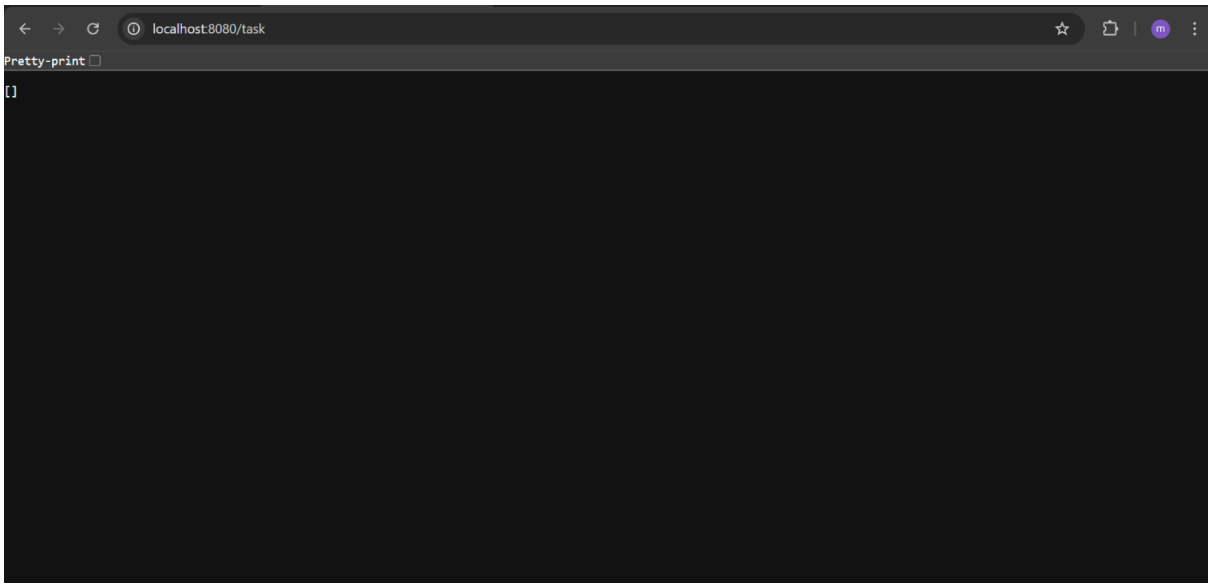
2.Check the Console:

- Eclipse will show the log in the **Console** tab.
- Look for a message like `Tomcat started on port(s): 8080 (http)` indicating the server is running.



3.Verify the Application:

- Open a web browser and navigate to `http://localhost:8080/`.
- If everything is set up correctly, you should see a default page or be able to test the available endpoints.



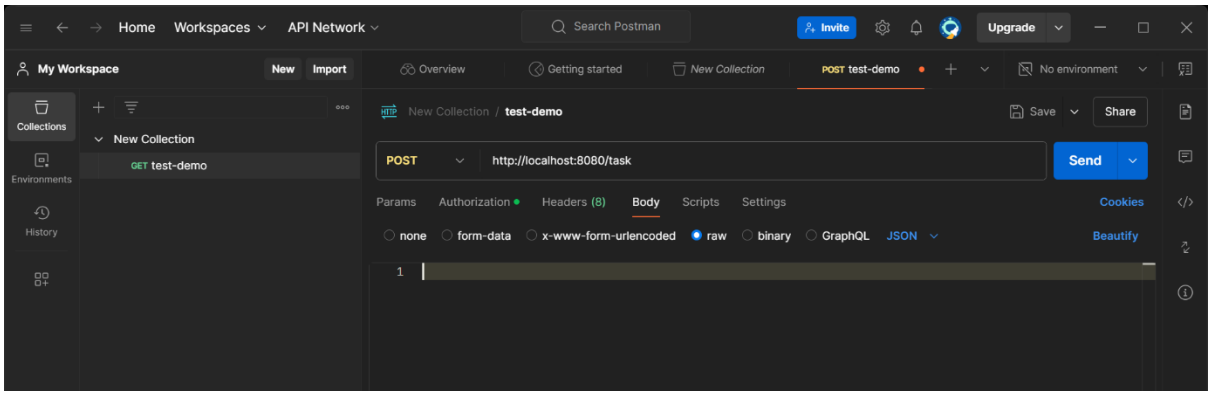
3. Testing with Postman:

- Once your application is running, you can test it using **Postman** to make HTTP requests. Since you've implemented **Basic Authentication**, you'll need to include your authentication credentials with the requests.

Testing POST Request (Create Task):

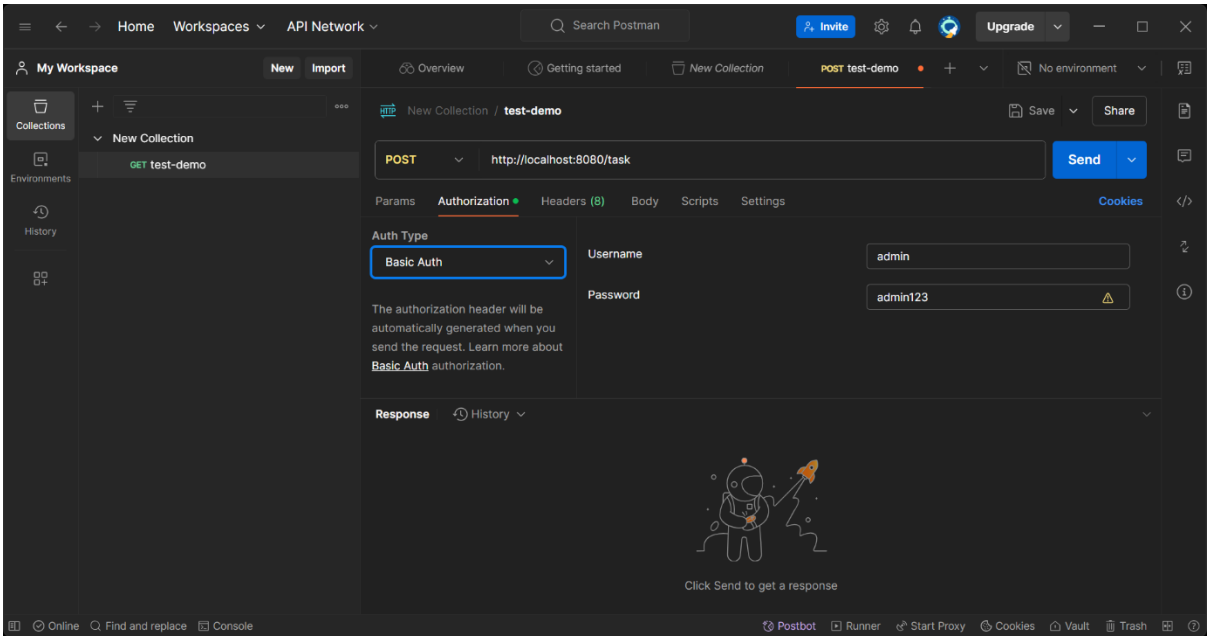
1. **Open Postman**.
2. **Set up the POST Request**:

- URL: `http://localhost:8080/task`
- HTTP Method: `POST`



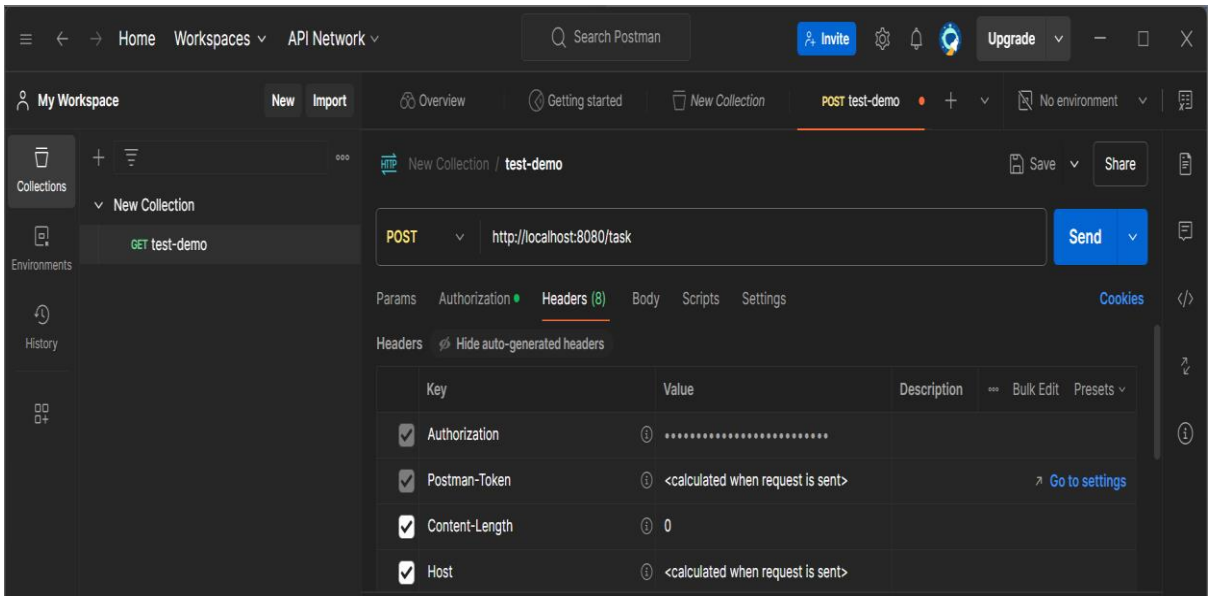
Authorization:

- Go to the **Authorization** tab in Postman.
- Set **Type** to `Basic Auth`.
- Username: `admin`
- Password: `admin123`



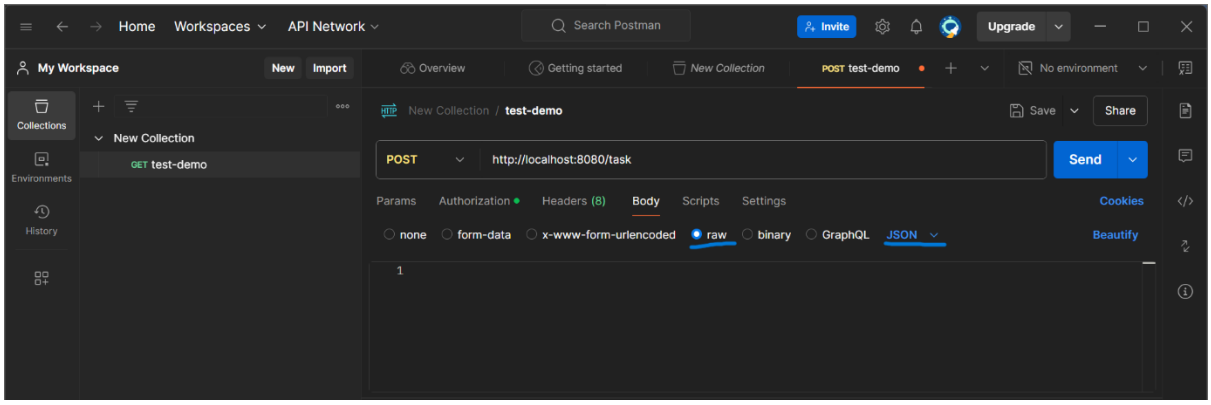
Headers:

- Ensure the header `Content-Type` is set to `application/json`.

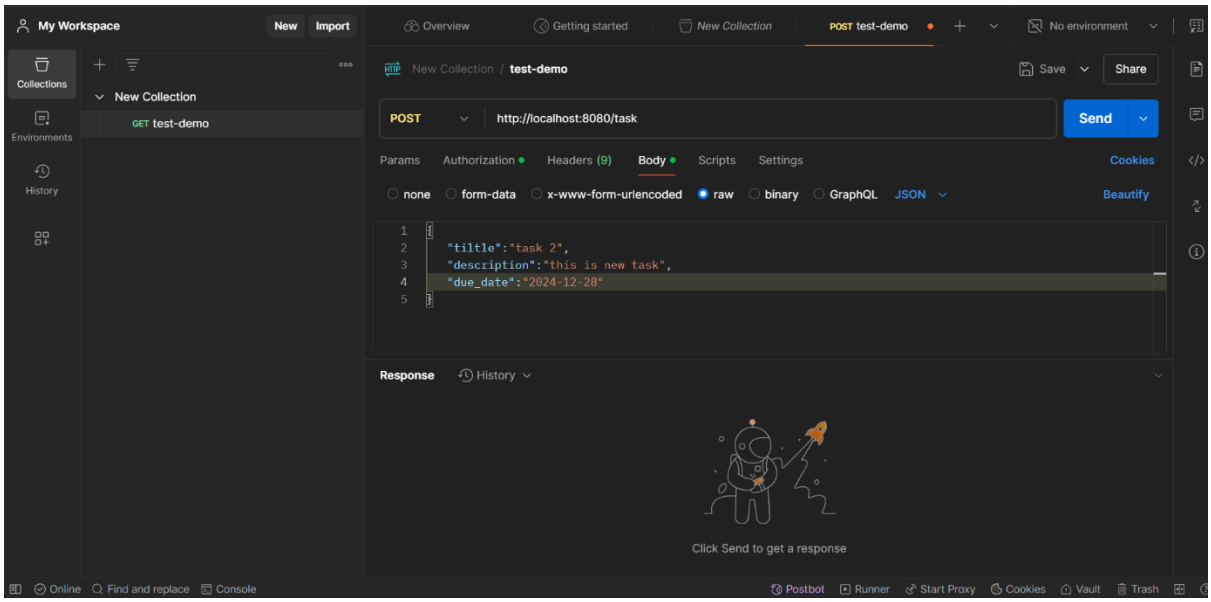


Request Body (JSON):

- Select the **Body** tab and choose **raw**.
- Set the format to **JSON**.
- Add the following task details in the body:

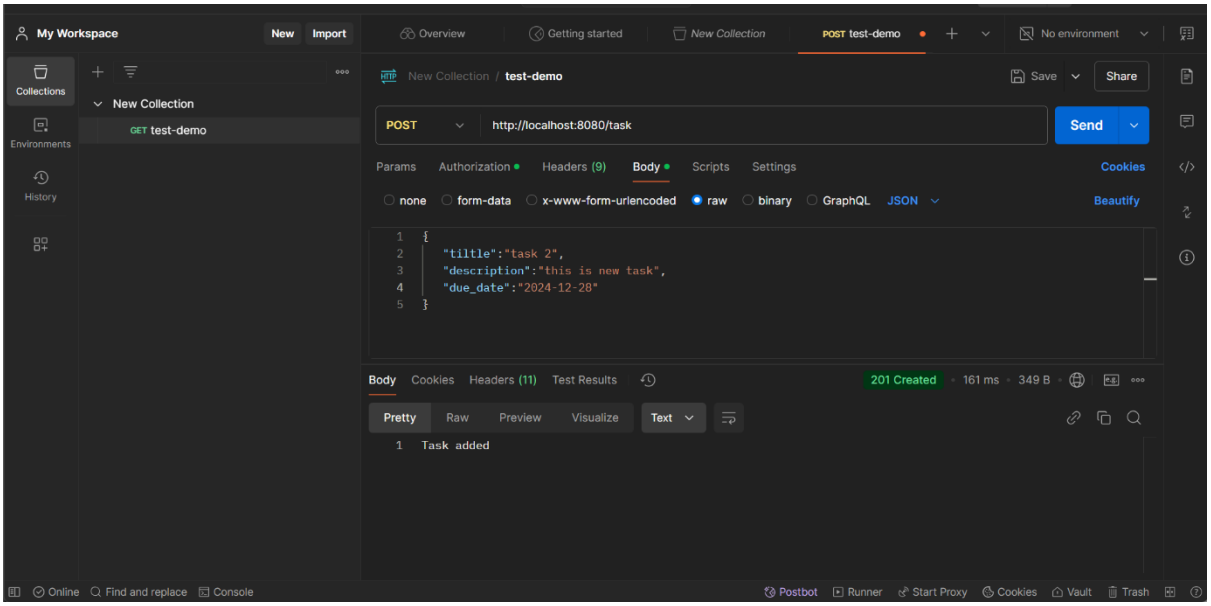


- Add the following task details in the body:



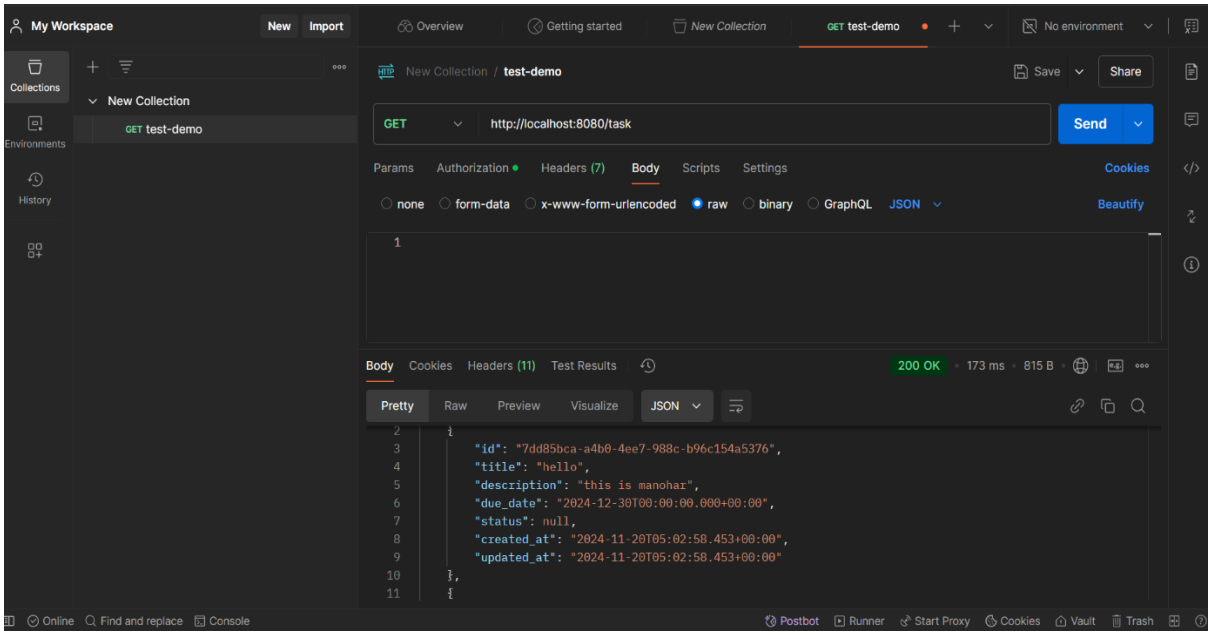
Send the Request:

- Click **Send** to send the POST request.
- You should get a **201 Created** response with the body: "`Task added`".



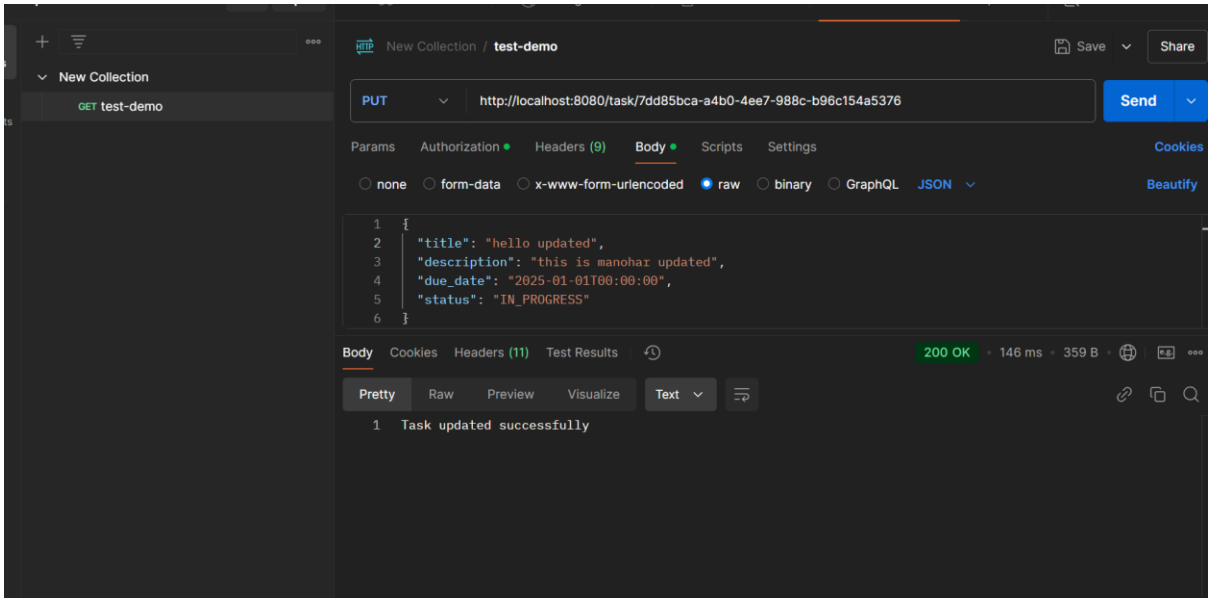
Testing GET Request (Get All Tasks):

- **Open Postman**.
- **Set up the GET Request**:
- URL: `http://localhost:8080/task`
- HTTP Method: `GET`
- **Authorization**:
- Again, use **Basic Auth** with username `admin` and password `admin123`.
- **Send the Request**:
- Click **Send**.
- The response should return a JSON array of all the tasks. It will be empty initially unless you've created tasks beforehand.



Testing PUT Request (Update Task)

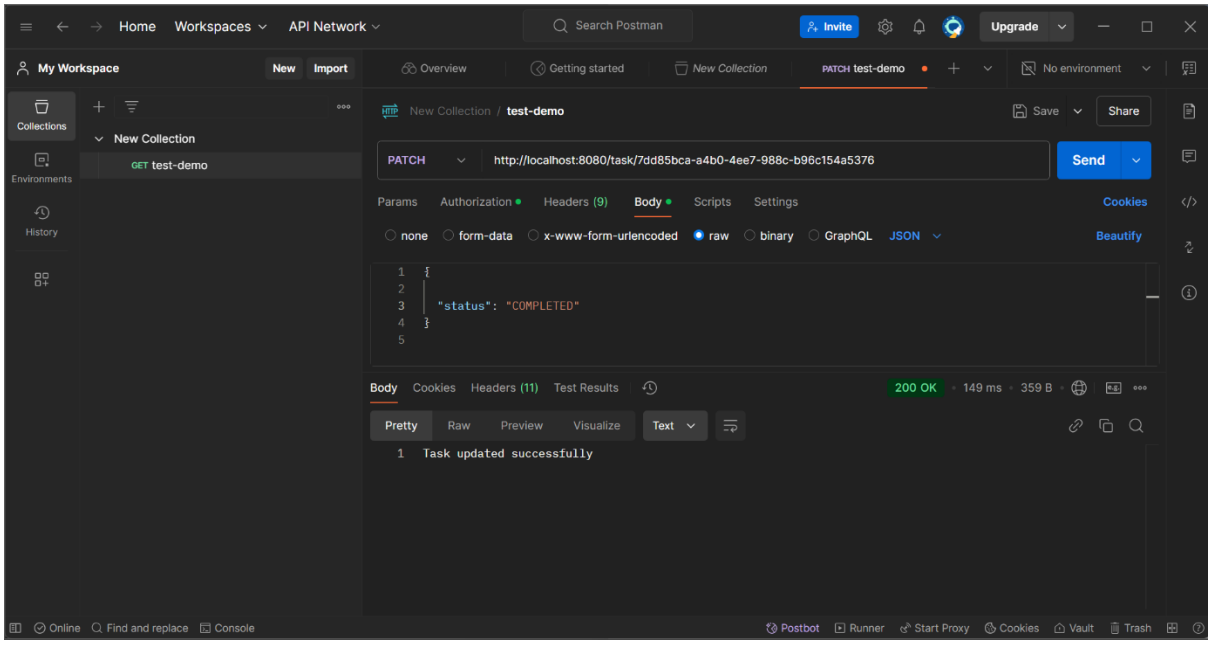
1. **Create a Task First**:
    - Make sure you've already created a task using the POST method.
2. **Set up the PUT Request**:
    - URL: `http://localhost:8080/task/{taskId}` (replace `{taskId}` with an actual task ID created earlier).
    - HTTP Method: `PUT`
3. **Authorization**:
    - Use **Basic Auth** again with username `admin` and password `admin123`.
4. **Request Body** (JSON):
    - Set the body of the request to modify the task:



- **Send the Request**:
- Click **Send**.
- You should receive a **200 OK** response with `"Task updated successfully"`.

Testing PATCH Request (Partially Update Task):

1. **Set up the PATCH Request**:
    - URL: `http://localhost:8080/task/{taskId}` (replace `{taskId}` with an actual task ID).
    - HTTP Method: `PATCH`
2. **Authorization**:
    - Use **Basic Auth** again with username `admin` and password `admin123`.
3. **Request Body** (JSON):
    - Add only the fields you want to modify (e.g., update only the status



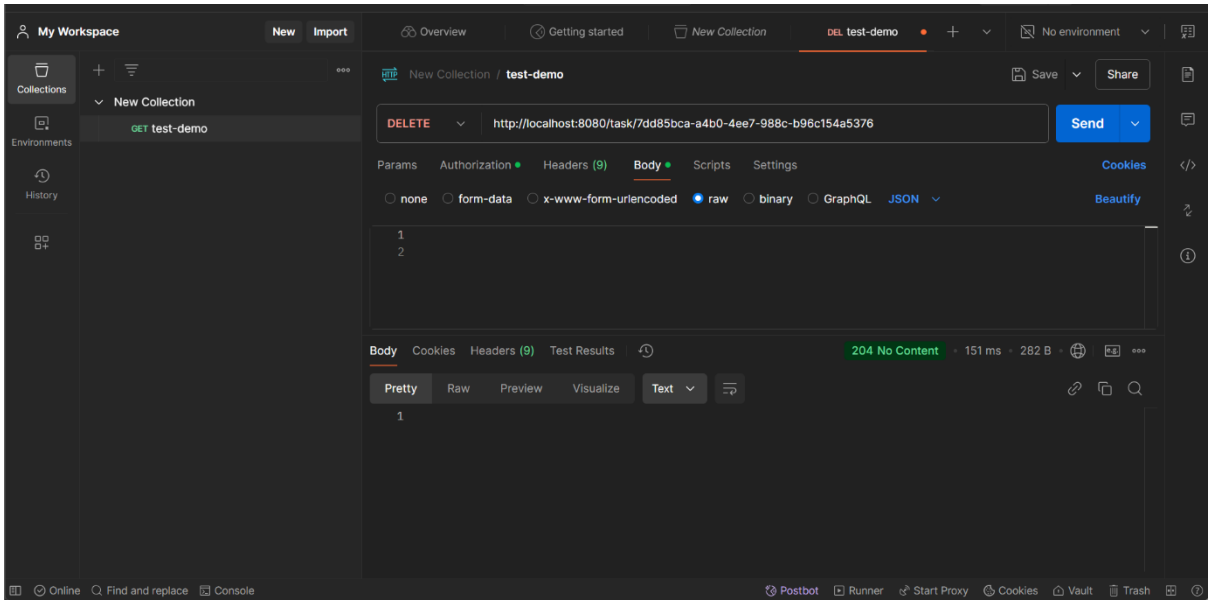
Send the Request:

- Click **Send**.
- You should receive a **200 OK** response with `"Task updated successfully"`.

Testing DELETE Request (Delete Task):

1. **Set up the DELETE Request**:
    - URL: `http://localhost:8080/task/{taskId}` (replace `{taskId}` with an actual task ID).
    - HTTP Method: `DELETE`
2. **Authorization**:
    - Use **Basic Auth** with username `admin` and password `admin123`.
3. **Send the Request**:
    - Click **Send**.
    - You should receive a **204 No Content** response if the task was deleted successfully.

    If the task ID doesn't exist, you will get a **404 Not Found** response.


My Workspace
New
Import
Overview
Getting started
New Collection
DEL test-demo
No environment
Collections
Environments
History
New Collection
GET test-demo
New Collection / test-demo
Save
Share
DELETE
http://localhost:8080/task/7dd85bca-a4b0-4ee7-988c-b96c154a5376
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1
2
Body
Cookies
Headers (9)
Test Results
204 No Content
151 ms
282 B
Pretty
Raw
Preview
Visualize
Text
1
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

Testing from the Browser:

For the **GET** request (like retrieving all tasks or a task by ID), you can also test it directly in your browser.

1. Open your browser.
2. Type the following URL to get all tasks:

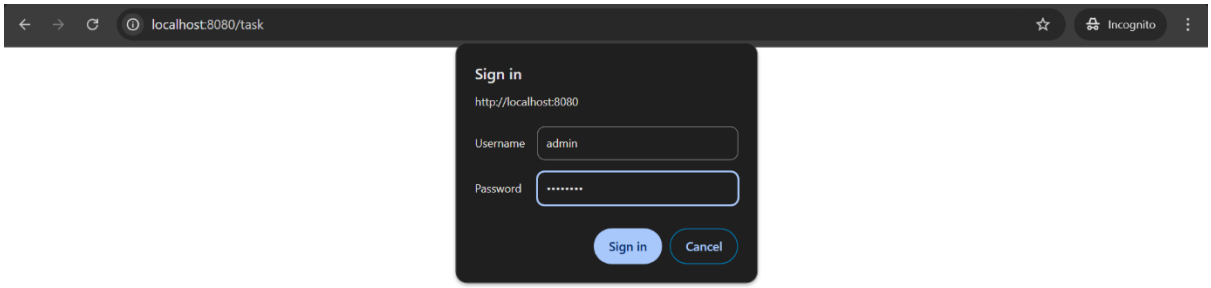
   ```
   Copy code
   http://localhost:8080/task
   ```

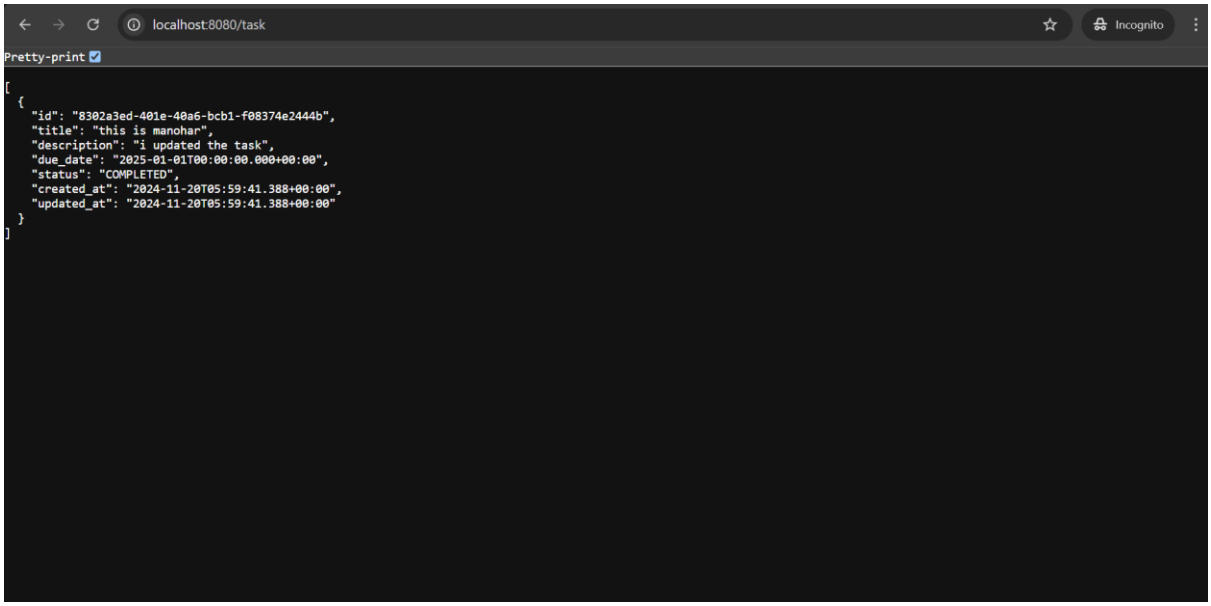
   - You should see a JSON response with all tasks, or an empty array if no tasks exist.

3. To test getting a specific task by ID, use:

   ```
   Copy code
   http://localhost:8080/task/{taskId}
   ```

   - Replace `{taskId}` with an actual task ID.
   - You should get the details of the task if it exists or a **404 Not Found** response if it doesn't.


localhost:8080/task
Incognito
Sign in
http://localhost:8080
Username
admin
Password
Sign in
Cancel


localhost:8080/task
Incognito
Pretty-print
[
{
"id": "8302a3ed-401e-40a6-bcb1-f08374e2444b",
"title": "this is manohar",
"description": "i updated the task",
"due_date": "2025-01-01T00:00:00.000+00:00",
"status": "COMPLETED",
"created_at": "2024-11-20T05:59:41.388+00:00",
"updated_at": "2024-11-20T05:59:41.388+00:00"
}
]

JUnit Testing in Spring Boot Application:

1. Understanding the JUnit Test Class:

In my project, I have a **JUnit test class** `ControllerTest.java` where i testing my `Controller`'s HTTP endpoints.

Let's break down the key elements in my test class:

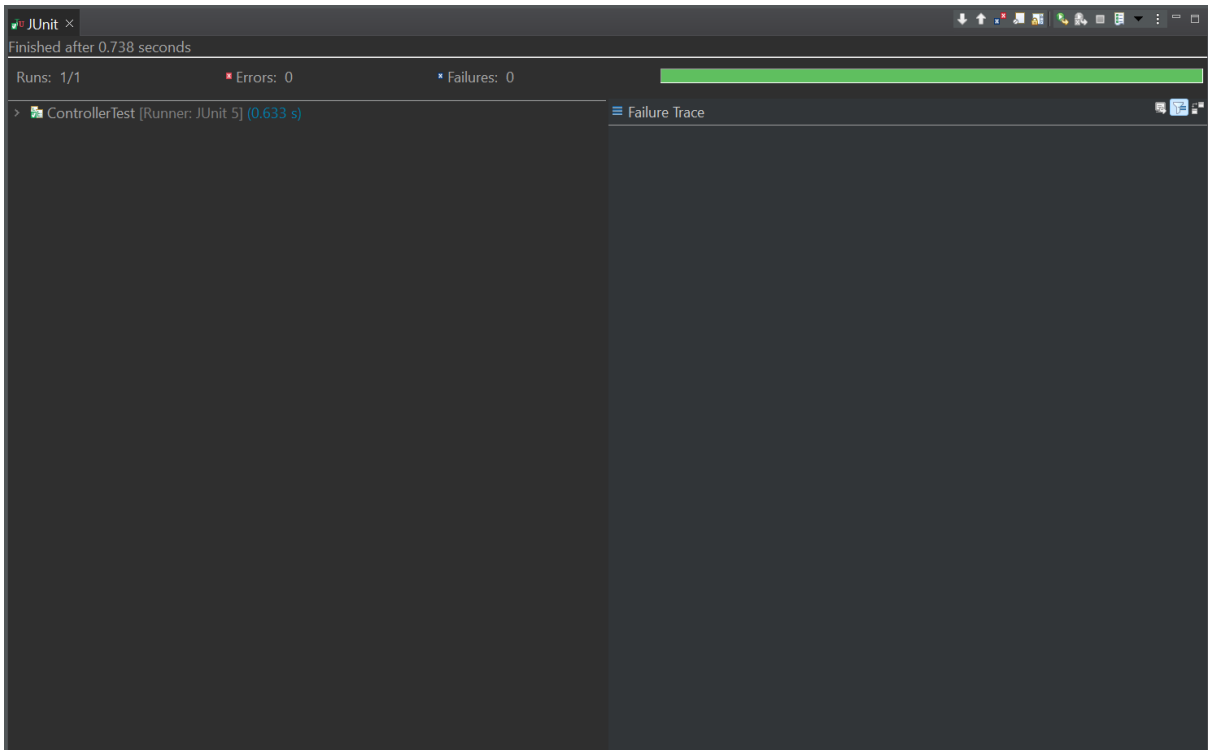
Key Annotations Used in JUnit:

- **`@Test`:** Marks a method as a test method to be run by the JUnit framework.
- **`@BeforeEach`:** Runs before each test case. You are initializing a `RestTemplate` instance before each test.
- **`@SpringBootTest`:** Loads the full application context for integration testing (used in tests where you want to test the whole Spring Boot application context).

Running JUnit Tests:

- **Right-click** on the test class (e.g., `ControllerTest.java`) in the **Project Explorer**.
- Select **Run As** > **JUnit Test**.

- Eclipse will run the test, and the results will be shown in the **JUnit** view. You'll see green (success) or red (failure) indicators for each test case.



Source code

```
package com.task_management.operations_tm;


import org.springframework.boot.SpringApplication;

import
org.springframework.boot.autoconfigure.SpringBootAp
plication;


@SpringBootApplication

public class OperationsTmApplication {


	public static void main(String[] args) {


	SpringApplication.run(OperationsTmApplication.cl
ass, args);

	}
```

}

```
package com.task_management.operations_tm.model;
import java.sql.Timestamp;
import java.util.Date;


public class RestUser {

	private String id;
	private String title;
	private String description;
	private Date due_date;
	private Status status;
	private Timestamp created_at;
	private Timestamp updated_at;
```

```
    public enum Status {
    PENDING, IN_PROGRESS, COMPLETED
}

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getDescription() {
```

```
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public Date getDue_date() {
        return due_date;
    }
    public void setDue_date(Date due_date) {
        this.due_date = due_date;
    }
    public Status getStatus() {
        return status;
    }
    public void setStatus(Status status) {
        this.status = status;
    }
    public Timestamp getCreated_at() {
```

```
        return created_at;
    }
    public void setCreated_at(Timestamp created_at) {
        this.created_at = created_at;
    }
    public Timestamp getUpdated_at() {
        return updated_at;
    }
    public void setUpdated_at(Timestamp
updated_at) {
        this.updated_at = updated_at;
    }
}
```

```
package com.task_management.operations_tm.controller;


import java.sql.Timestamp;

import java.util.Collection;

import java.util.HashMap;

import java.util.Map;

import java.util.UUID;


import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

import
org.springframework.web.bind.annotation.DeleteMapping;

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

import
org.springframework.web.bind.annotation.PatchMapping;

import
org.springframework.web.bind.annotation.PathVariable;

import
org.springframework.web.bind.annotation.PostMapping;
```

```
import
org.springframework.web.bind.annotation.PutMapping;

import
org.springframework.web.bind.annotation.RequestBody;

import
org.springframework.web.bind.annotation.RequestMapping;

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


import
com.task_management.operations_tm.model.RestUser;


@RestController

@RequestMapping("/task")

@Validated

public class Controller {


    Map<String,RestUser> alltasks = new HashMap<>();


    @GetMapping
```

```
    public Collection<RestUser> getMethod() {
        return alltasks.values();
    }
     @GetMapping("/{taskId}")
       public ResponseEntity<RestUser>
getTaskById(@PathVariable String taskId) {
          RestUser task = alltasks.get(taskId);
          if (task == null) {
             return new
ResponseEntity<>(HttpStatus.NOT_FOUND);
          }
          return new ResponseEntity<>(task, HttpStatus.OK);
       }


    @PostMapping
    public ResponseEntity<String> postMethod(@Validated
@RequestBody RestUser taskdetails) {
```

```
String taskId = UUID.randomUUID().toString();
RestUser addtask = new RestUser();

addtask.setId(taskId);
addtask.setTitle(taskdetails.getTitle());
addtask.setDescription(taskdetails.getDescription());
addtask.setDue_date(taskdetails.getDue_date());
addtask.setStatus(taskdetails.getStatus());

Timestamp currentTimestamp = new
Timestamp(System.currentTimeMillis());
addtask.setCreated_at(currentTimestamp);
addtask.setUpdated_at(currentTimestamp);

alltasks.put(taskId,addtask);
```

```
            return new ResponseEntity<>("Task
added",HttpStatus.CREATED);

        }

        @PutMapping(path="/{taskId}")

        public ResponseEntity<String>
putMethod(@PathVariable String taskId,@RequestBody
RestUser taskdetails) {

            if(!alltasks.containsKey(taskId)) {

                return new ResponseEntity<>("Task ID not
found",HttpStatus.BAD_REQUEST);

            }

                RestUser existT = alltasks.get(taskId);

                existT.setTitle(taskdetails.getTitle());


        existT.setDescription(taskdetails.getDescription());


        existT.setDue_date(taskdetails.getDue_date());

                existT.setStatus(taskdetails.getStatus());
```

```
                existT.setUpdated_at(new
Timestamp(System.currentTimeMillis()));


        alltasks.put(taskId, existT);

        return new ResponseEntity<>("Task updated
successfully", HttpStatus.OK);

            }



    @PatchMapping(path="/{taskId}")

    public ResponseEntity<String>
patchMethod(@PathVariable String taskId, @RequestBody
RestUser taskdetails) {

            if(!alltasks.containsKey(taskId)) {

                return new ResponseEntity<>("user ID not
found",HttpStatus.NOT_FOUND);


            }

                RestUser existT = alltasks.get(taskId);
```

```
                if(taskdetails.getTitle() != null) {
                        existT.setTitle(taskdetails.getTitle());
                }
                if (taskdetails.getDescription() != null) {
            existT.setDescription(taskdetails.getDescription());
        }
        if (taskdetails.getDue_date() != null) {
            existT.setDue_date(taskdetails.getDue_date());
        }
        if (taskdetails.getStatus() != null) {
            existT.setStatus(taskdetails.getStatus());
        }
        existT.setUpdated_at(new
Timestamp(System.currentTimeMillis()));


        alltasks.put(taskId, existT);


            return new ResponseEntity<>("Task updated
successfully",HttpStatus.OK);
```

```
	}

	@DeleteMapping(path="/{taskId}")

	public ResponseEntity<String>
deleteMethod(@PathVariable String taskId) {



		if(!alltasks.containsKey(taskId)) {

			return new ResponseEntity<>("task ID not
found",HttpStatus.NOT_FOUND);

		}

		else {

			alltasks.remove(taskId);

			return new
ResponseEntity<>(HttpStatus.NO_CONTENT);

		}

	}

}
```

```
package com.task_management.operations_tm.controller;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.http.*;
import org.springframework.web.client.RestTemplate;

import static org.junit.jupiter.api.Assertions.*;

public class ControllerTest {

    private static final String url = "http://localhost:8080/task";
    private RestTemplate restTemplate;

    @BeforeEach
    public void setUp() {
        restTemplate = new RestTemplate();
    }
```

```
    @Test
    public void testPostMethodWithBasicAuth() {
        // Create the task object to send
         StringBuilder taskJBuilder = new StringBuilder();
         taskJBuilder.append("{");
         taskJBuilder.append("\"title\": \"New Task\", ");
         taskJBuilder.append("\"description\": \"Task
description\", ");
         taskJBuilder.append("\"due_date\": \"2024-12-
31T00:00:00\", ");
         taskJBuilder.append("\"status\": \"PENDING\"");
         taskJBuilder.append("}");


         String taskJ = taskJBuilder.toString();
        // Set up Basic Authentication
        String username = "admin";
        String password = "admin123";
        String auth = "Basic " +
java.util.Base64.getEncoder().encodeToString((username +
":" + password).getBytes());
```

```
        HttpHeaders headers = new HttpHeaders();

        headers.set("Authorization", auth); // Set the Basic Auth
header


headers.setContentType(MediaType.APPLICATION_JSON); //
Set the content type to JSON



        HttpEntity<String> entity = new HttpEntity<>(taskJ,
headers);



        // Send POST request

        ResponseEntity<String> response =
restTemplate.exchange(url, HttpMethod.POST, entity,
String.class);



        // Assert response status

        assertEquals(HttpStatus.CREATED,
response.getStatusCode(), "Expected HTTP Status 201");

        assertEquals("Task added", response.getBody(),
"Expected response body to be 'Task added'");
```

```
    }
}
```

```
package com.task_management.operations_tm.security;


import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders
.HttpSecurity;
import
org.springframework.security.config.annotation.web.configu
ration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.provisioning.InMemoryUserDe
tailsManager;
import
org.springframework.security.web.SecurityFilterChain;
```

```
@Configuration

@EnableWebSecurity

public class BasicSecurityApi {

	// Configure in-memory user details with basic authentication

	@Bean

	public InMemoryUserDetailsManager userDetailsService() {

	    UserDetails user = User.withDefaultPasswordEncoder() // Password encoder for demo purposes

	        .username("admin") // Username

	        .password("admin123") // Password

	        .roles("USER") // User role

	        .build();


	    return new InMemoryUserDetailsManager(user);

	}


	// Configure security filter chain
```

```
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf().disable() // Disable CSRF for simplicity (for APIs, typically disabled)
            .authorizeRequests()
                .requestMatchers("/task/**").authenticated() // Secure `/task` endpoints
                .anyRequest().permitAll()  // Allow all other requests
            .and()
            .httpBasic();  // Enable HTTP Basic Authentication

        return http.build();
    }
}
```

Conclusion:

- Overall, this project demonstrates how to quickly build a RESTful API for managing tasks using **Spring Boot**. The application is secure, testable, and provides a solid foundation for further enhancements, such as integrating a database, improving security features, and providing a more detailed user interface. The project successfully meets its aim of building a simple but efficient task management system, and it serves as a great starting point for more complex systems.