TEAM PROJECT TO-DO-APP

Course IT 538 SW-T

TO DO APP Test Specification version v2024.3

By: Baran Karakurt
Hakan Karayılmaz

Supervisor: Ömer Karacan

Istanbul, 17.03.2023

Table of Contents

1. Introduction 3

2. Test planning 6

3. Test monitoring and control 8

4. Test analysis 9

5. Test design 10

6. Test implementation 23

7. Test execution 24

8. Test completion 28

1. Introduction

Developed during the SW-T IT538 2024 spring semester by Group Baran KARAKURT & Hakan KARAYILMAZ

Structure based on the final work "SW-T-(IT 538)-To-Do-App-Test-Specification-v2024.3.docx"

Motivation

In today's world, efficiently managing and organizing tasks has become crucial for personal and professional productivity. The To Do App serves as a prior tool in this context, offering such as task creation, deletion, updating, and viewing.

Task organization, including creation, deletion, updating and viewing all tasks are use cases. Assumed that User has prior knowledge about how to use To Do App.

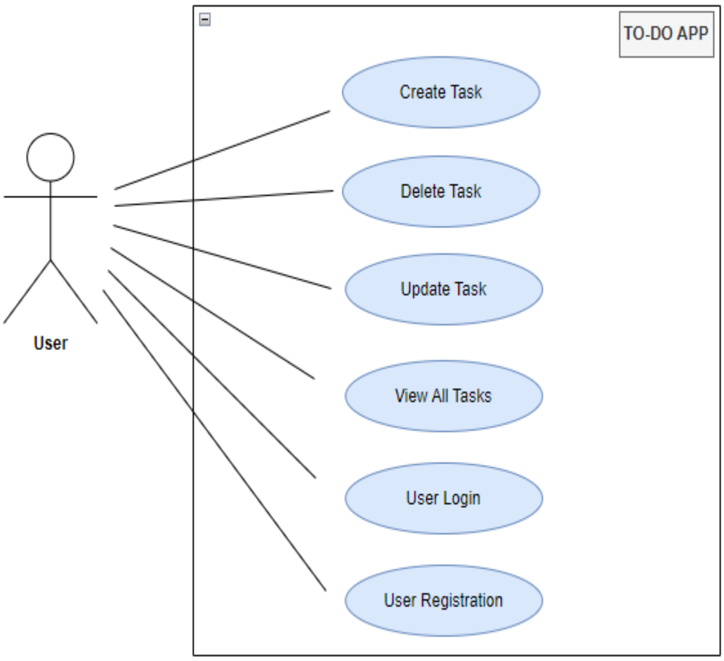


Figure 1.1 To Do App UML Diagram – Use Cases

The "To Do App" serves as an application designed to streamline task management, offering users the ability to create, modify, and delete tasks with associated titles, descriptions. In this project, To Do App will be tested.

The purpose of this document is to describe the test specification for To Do App based on the business requirements. According to these requirements, the following To Do App Activity Diagram is visualized, see Figure 1.2

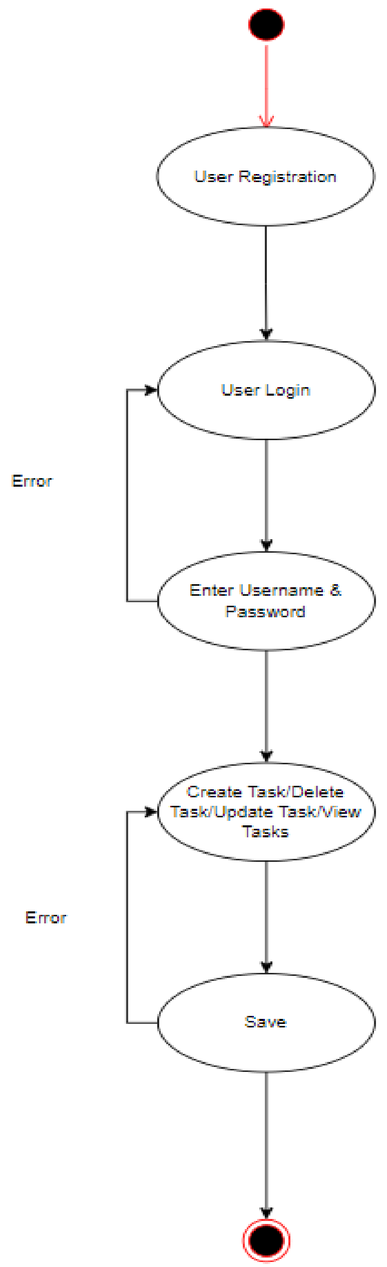

User Registration
User Login
Error
Enter Username & Password
Create Task/Delete Task/Update Task/View Tasks
Error
Save

Figure 1.2 To Do App Activity Diagram

System Under Test

The system under test is the "To Do App" application, covers use cases such as task creation, editing, deletion, viewing all tasks. Additionally, use cases such as user login and user registration will be tested. Also, tests aim to ensure that activity diagram work the same way shown in Figure 1.2. according to the business requirements. This testing approach aims to validate the application's overall use case scenarios.

Data Structures

To use To Do App, user need to register and login To Do App with registered account. User registration success depends specified data types which is given by user, according to business requirements.

After user login into the To Do App, user should be able to create new tasks. Creation of task success depends on specified data types according to the business requirement. Table 1.1, shows data structure of To Do App.

	Attribute Name	Description	Data Type	Verification Method
Data Structure	id	Unique identifier for each user	integer	unique
	fullname	User fullname	string	char < 25 && not null
	email	User email address	string	valid email structure
	password	User password	string	char < 25 && not null
	Id	Unique to do id	integer	unique
	title	Title for to do	string	char < 50 && not null
	content	Content for to do	string	char < 100 && not null
	completed	To do status	boolean	true | false

Table 1.1 Data Structure of The To Do App

2. Test planning

Determining the scope, objectives, and risks of testing

Scope: Aim for testing the use cases of the application; allowing a user to create, update, and delete tasks through the interface and backend processes.
Objectives: Ensure the application is error-free, identify potential defects.
Risks: Malfunctioning of task management features, login, and registration problems.

Making decisions about what to test

- Backend API calls (Rest API Interface).
- User input validation.
- Create, delete, update, view, user login, user registration works properly.

How test activities will be carried out

Test activities are:

- test monitoring and control,
- test analysis,
- test design,
- test implementation,
- test execution, and
- test completion.

Scheduling of test analysis, design, implementation, execution, and evaluation activities on particular dates

Test analysis and design activities completed on 24.02.2021. Test implementation activity started on 04.03.2024 and will be completed on 10.03.2024. Test execution activities will start after the first cases are written and cases will be run every day by using the Jest library. As part of Evaluation activities, problems that arise during execution test cases will be detected and resolved. The process will continue until all test cases are successfully executed. Target date for evaluation activities is 11.03.2022.

Selecting metrics for test monitoring and control

A coverage rate of overall 70% has been decided for evaluation of testing except for user registration which is aimed for 100%.

Determining the level of detail and structure for test documentation, e.g., by providing templates or example documents

Template of the To Do App Test Specification document was provided by our supervisor. Document is detailed by the project team.

<u>Level of Detail:</u>

We will create clear and concise test cases for the features of our application. This includes user registration, login, adding, updating, and deleting tasks. Each test case will be directly associated with the relevant feature, making it clear which functionality the test addresses. We will verify the functions of our application through tests, without engaging in automatic tests.

3. Test monitoring and control

Test monitoring and control are supported by the evaluation of exit criteria, of which tasks may include:

Checking test results and logs against specified coverage criteria!
Coverage criteria for all contents are 70%, except for user registration coverage criteria which is aimed for 100%. After the Coverage tool is run, the ratios will be checked.

Determining if more tests are needed (if yes, …)!
If coverage does not meet the requirement of criteria, then more tests will be designed to test those items that were increase coverage. Coverage ratio will be observed after all tests, if coverage acceptance rate is not aligned with acceptance criteria, process will iterate.

4. Test analysis

Analyzing the requirement specifications

Test analysis is performed for To Do App activity diagram. As mentioned before, the figure is used for analyzing the business requirement specification, see Figure 1.2.

Design and implementation information

Test cases are designed based on To Do App Activity diagram and business requirements.
Jest library is going to be used unit testing. Additionally, Postman is going to be used for testing Rest APIs.

The implementation of the component or system itself

The app is developed by TypeScript using Express.js (Node.js framework) for the backend to handle requests. The main features include user registration, login, and the ability to add, view, edit, and delete tasks. Our testing will concentrate on ensuring that these features work as expected. We wrote the code with testing in mind, aiming to make each part of the app easy to test individually.

Identifying features and sets of features to be tested.

- User Registration: Test the ability for new users to register.
- User Login: Verify that registered users can log in with valid credentials and receive appropriate error messages for invalid attempts.
- Task Creation: Validate that users can create tasks with description details.
- Task Viewing: Test the functionality for users to view their all tasks.
- Task Updating: Ensure that users can update the details of an existing task.
- Task Deletion: Verify that users can delete their tasks.

5. Test design

User Registration

	Step	Description
Main Success Scenario A: Actor S: System	1	A: Send valid username, email, and password
	2	S: Validates input and creates new user account

Table 2.1 User Registration Use Case Scenario

User Login

	Step	Description
Main Success Scenario A: Actor S: System	1	A: Send registered username and password
	2	S: Validates credentials
Extensions	2a	User enters incorrect username or password S: Send message "Username does not exist or wrong password"

Table 2.2 User Login Use Case Scenario

Create Task

Main Success Scenario A: Actor S: System	Step	Description
	1	A: Send valid title, description and completed info
	2	S: Create new task
Extensions	2a	User enters invalid input S: Send message "Please enter a valid input"

Table 2.3 Create Task Use Case Scenario

Update Task

Main Success Scenario A: Actor S: System	Step	Description
	1	A: Send valid title, description and completed info
	2	S: Update the task
Extensions	2a	User enters invalid input: S: Send message "Please enter a valid input"

Table 2.4 Update Task Use Case Scenario

Delete Task

Main Success Scenario A: Actor S: System	Step	Description
	1	A: Send task ID
	2	S: Delete Task
Extensions	2a	User enters incorrect id S: Send message "The task doesn’t exist"

Table 2.5 Delete Task Use Case Scenario

View All Task

Main Success Scenario A: Actor S: System	Step	Description
	1	A: Send request for all task
	2	S: Send Tasks
Extensions	2a	There are not any tasks. S: Send message "There is not any task”

Table 2.6 View All Task Use Case Scenario

Designing test cases

In this project, we employed a use case approach to analyze the functionality of the To Do App, focusing on essential operations such as creating, deleting, updating, and viewing tasks. We focus on the user's interactions with the system through specific use cases.

Each use case represents a distinct user goal or task that the system should facilitate. For instance, creating a task involves the user adding a new task to their list, while deleting a task entail removing an existing task from the list. Updating a task involves modifying task details, and viewing tasks involves displaying the list of tasks to the user.

Our goal is to ensure that the To Do App accurately fulfills each of these use cases, meeting the requirements. To achieve this, we identify and define each use case, outlining the steps involved and the expected response of the system in response to user actions.

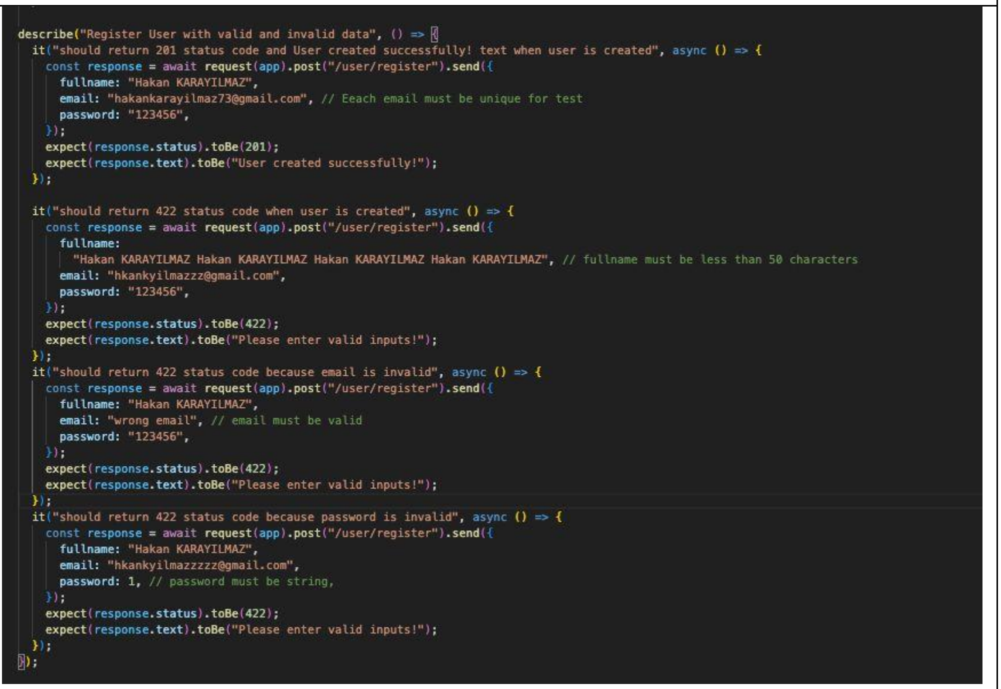
Identifying necessary test data to support test conditions and test cases

Valid and invalid inputs are given as test data. The application will give the information that the operation has occurred or not according to the relevant data.

Identifying any required infrastructure and tools

For CRUD method testing, VS Code and Jest were used. We can conduct the tests by writing a Jest test script, running the test, and viewing the results to see whether they match the expected results or not.

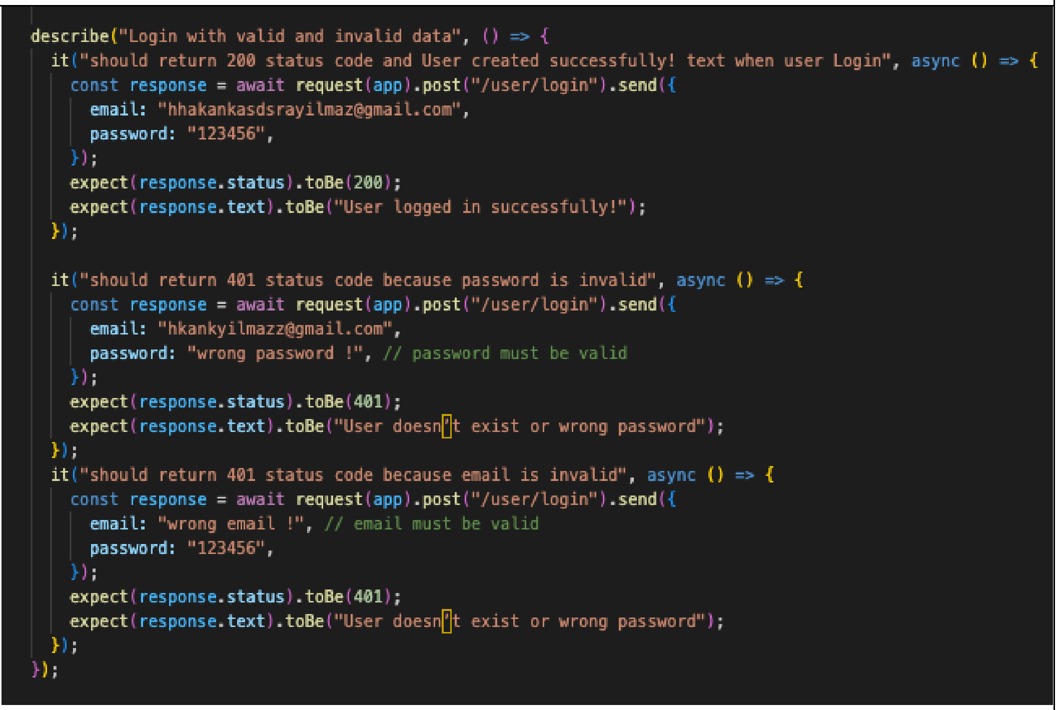
Register User Test Cases

Test Case ID Name	TC1_Register User with valid and invalid data
Preconditions	
Test Steps	1.Send POST http request for register user
Post-Condition	It is assumed that the user has registered
Test Data	1. Inputs → fullname : valid , email : valid, password : valid 2. Inputs → fullname : unvalid , email : valid, password : valid 3. Inputs → fullname : valid , email : unvalid, password : valid 4. Inputs → fullname : valid , email : valid, password : unvalid
Expected Result	1. Status → 201 data → “User created successfully!” 2. Status → 422 data → “Please enter valid input” 3. Status → 422 data → “Please enter valid input” 4. Status → 422 data → “Please enter valid input”
Actual Result	Matches the Expected results
Verdict (Pass/Fail)	PASS
Code Coverage	

```
describe("Register User with valid and invalid data", () => {
  it("should return 201 status code and User created successfully! text when user is created", async () => {
    const response = await request(app).post("/user/register").send({
      fullname: "Hakan KARAYILMAZ",
      email: "hakankarayilmaz73@gmail.com", // Eeach email must be unique for test
      password: "123456",
    });
    expect(response.status).toBe(201);
    expect(response.text).toBe("User created successfully!");
  });

  it("should return 422 status code when user is created", async () => {
    const response = await request(app).post("/user/register").send({
      fullname:
        "Hakan KARAYILMAZ Hakan KARAYILMAZ Hakan KARAYILMAZ Hakan KARAYILMAZ", // fullname must be less than 50 characters
      email: "hkankyilmazzz@gmail.com",
      password: "123456",
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status code because email is invalid", async () => {
    const response = await request(app).post("/user/register").send({
      fullname: "Hakan KARAYILMAZ",
      email: "wrong email", // email must be valid
      password: "123456",
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status code because password is invalid", async () => {
    const response = await request(app).post("/user/register").send({
      fullname: "Hakan KARAYILMAZ",
      email: "hkankyilmazzzzz@gmail.com",
      password: 1, // password must be string,
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
});
```

Table 3.1 To Do App TC 1 Test User Registration

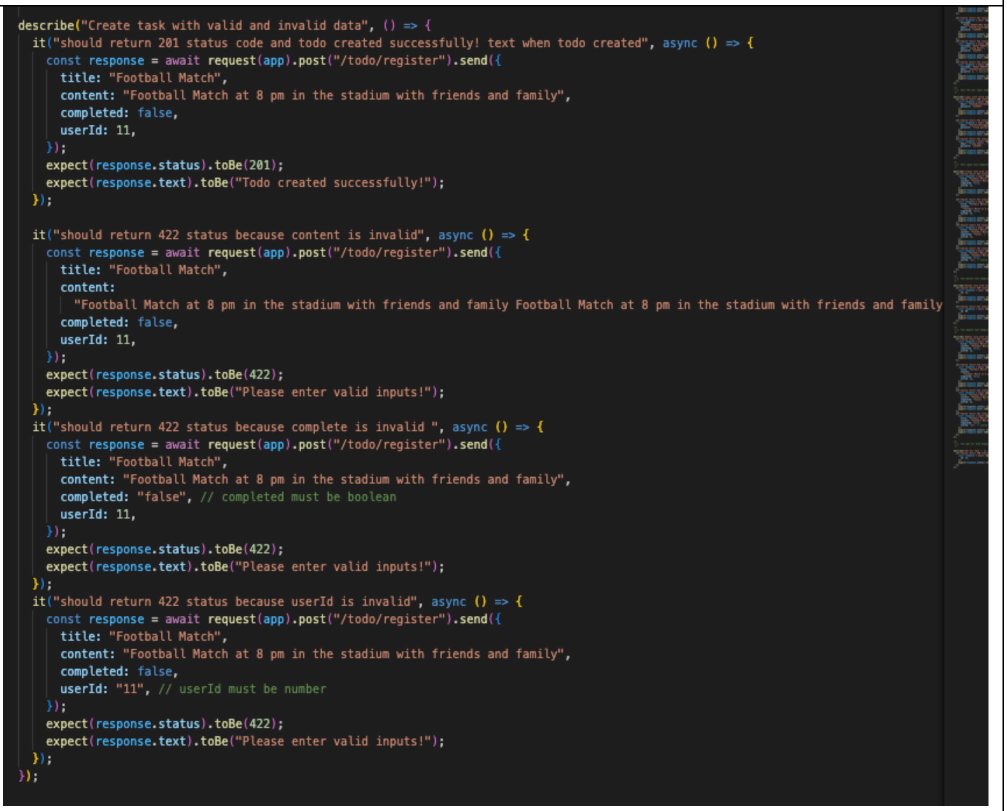
Login Task Test Cases

Test Case ID Name	TC2_Login with valid and invalid data
Preconditions	User is registered and at the login screen.
Test Steps	1.Send POST http request for create a task
Post-Condition	It is assumed that the user has registered and logged in to the to-do application.
Test Data	1. Inputs → email : valid , password : valid 2. Inputs → email : invalid , password : valid 3. Inputs → email : valid , password : invalid
Expected Result	1. Status → 200 data → “User logged in successfully!” 2. Status → 401 data → “User doesn’t exist or wrong password” 3. Status → 401 data → “User doesn’t exist or wrong password”
Actual Result	Matches the Expected results
Verdict (Pass/Fail)	PASS
Code Coverage	

```
describe("Login with valid and invalid data", () => {
  it("should return 200 status code and User created successfully! text when user Login", async () => {
    const response = await request(app).post("/user/login").send({
      email: "hhakankasdsrayilmaz@gmail.com",
      password: "123456",
    });
    expect(response.status).toBe(200);
    expect(response.text).toBe("User logged in successfully!");
  });

  it("should return 401 status code because password is invalid", async () => {
    const response = await request(app).post("/user/login").send({
      email: "hkankyilmazz@gmail.com",
      password: "wrong password !", // password must be valid
    });
    expect(response.status).toBe(401);
    expect(response.text).toBe("User doesn’t exist or wrong password");
  });
  it("should return 401 status code because email is invalid", async () => {
    const response = await request(app).post("/user/login").send({
      email: "wrong email !", // email must be valid
      password: "123456",
    });
    expect(response.status).toBe(401);
    expect(response.text).toBe("User doesn’t exist or wrong password");
  });
});
```

Table 3.2 To Do App TC 2 Test User Login

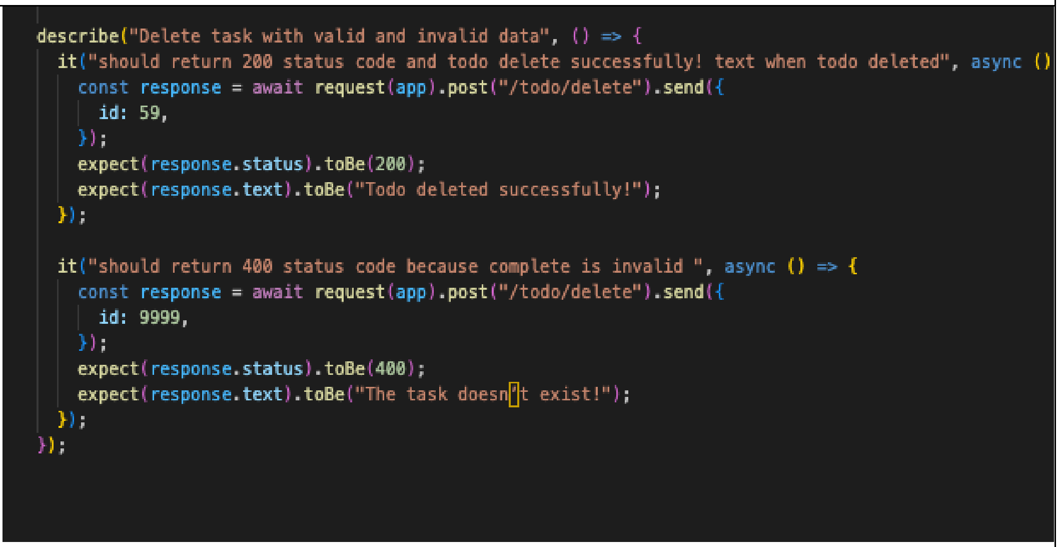
Create Task Test Cases

Test Case ID Name	TC3_Create a task with valid and invalid data
Preconditions	User is registered and at the login screen.
Test Steps	1.Send POST http request for create a task
Post-Condition	It is assumed that the user has registered and logged in to the to-do application.
Test Data	1. Inputs → title : valid, description : valid , completed : valid 2. Inputs → title : invalid, description : valid , completed : valid 3. Inputs → title : valid, description : invalid , completed : valid 4. Inputs → title : valid, description : valid , completed : invalid
Expected Result	1. Status → 200 data → “To do created successfully!” 2. Status → 422 data → “Please enter valid input” 3. Status → 422 data → “Please enter valid input” 4. Status → 422 data → “Please enter valid input”
Actual Result	Matches the Expected results
Verdict (Pass/Fail)	PASS
Code Coverage	

```
describe("Create task with valid and invalid data", () => {
  it("should return 201 status code and todo created successfully! text when todo created", async () => {
    const response = await request(app).post("/todo/register").send({
      title: "Football Match",
      content: "Football Match at 8 pm in the stadium with friends and family",
      completed: false,
      userId: 11,
    });
    expect(response.status).toBe(201);
    expect(response.text).toBe("Todo created successfully!");
  });

  it("should return 422 status because content is invalid", async () => {
    const response = await request(app).post("/todo/register").send({
      title: "Football Match",
      content:
        "Football Match at 8 pm in the stadium with friends and family Football Match at 8 pm in the stadium with friends and family
      completed: false,
      userId: 11,
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status because complete is invalid ", async () => {
    const response = await request(app).post("/todo/register").send({
      title: "Football Match",
      content: "Football Match at 8 pm in the stadium with friends and family",
      completed: "false", // completed must be boolean
      userId: 11,
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status because userId is invalid", async () => {
    const response = await request(app).post("/todo/register").send({
      title: "Football Match",
      content: "Football Match at 8 pm in the stadium with friends and family",
      completed: false,
      userId: "11", // userId must be number
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
});
```

Table 3.3 To Do App TC 3 Test Create Task

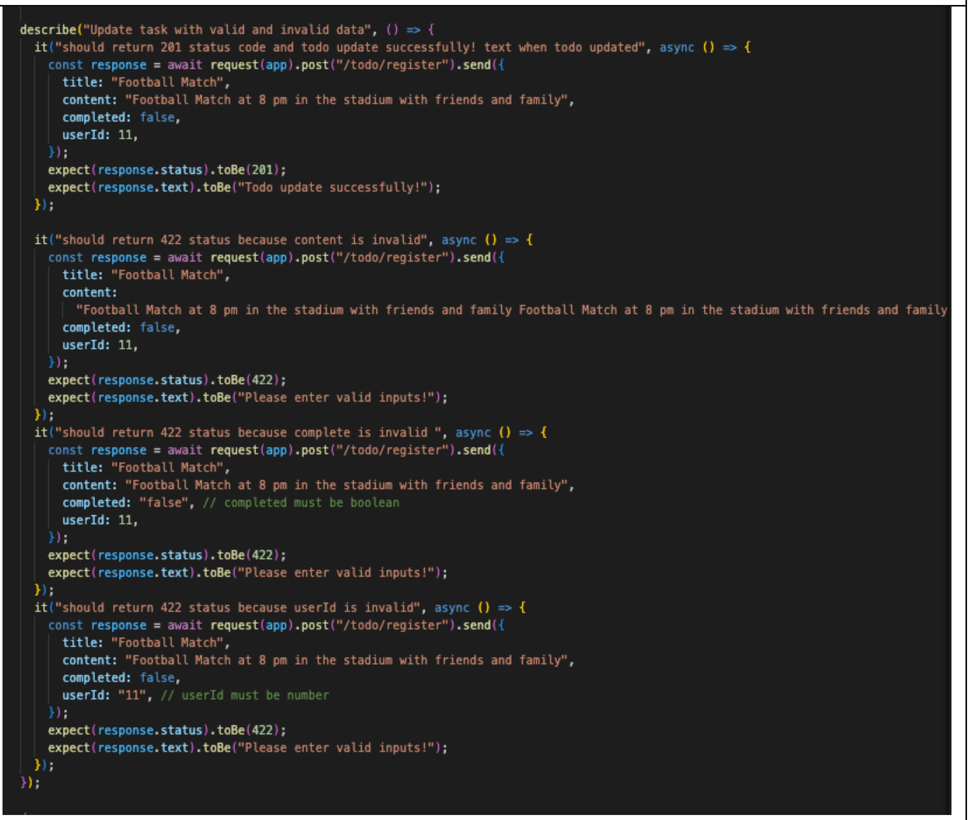
Delete Task Test Case

Test Case ID Name	TC4_Delete a task with a valid and invalid data
Preconditions	User is registered and at the login screen.
Test Steps	1.Send DELETE http request for create a task
Post-Condition	It is assumed that the user has registered and logged in to the to-do application.
Test Data	1. Inputs → id : valid 2. Inputs → id : unvalid
Expected Result	1. Status → 200 data → “To do deleted successfully!” 2. Status → 400 data → “The task doesn’t exist”
Actual Result	Matches the Expected results
Verdict (Pass/Fail)	PASS
Code Coverage	

```
describe("Delete task with valid and invalid data", () => {
  it("should return 200 status code and todo delete successfully! text when todo deleted", async ()
    const response = await request(app).post("/todo/delete").send({
      id: 59,
    });
    expect(response.status).toBe(200);
    expect(response.text).toBe("Todo deleted successfully!");
  });

  it("should return 400 status code because complete is invalid ", async () => {
    const response = await request(app).post("/todo/delete").send({
      id: 9999,
    });
    expect(response.status).toBe(400);
    expect(response.text).toBe("The task doesn't exist!");
  });
});
```

Table 3.4 To Do App TC 4 Test Delete Task

Update Task Test Cases

Test Case ID Name	TC5_Update a task with valid and invalid data
Preconditions	User is registered and at the login screen.
Test Steps	1.Send PUT http request for create a task
Post-Condition	It is assumed that the user has registered and logged in to the to-do application.
Test Data	1. Inputs → title : valid, description : valid , completed : valid 2. Inputs → title : invalid, description : valid , completed : valid 3. Inputs → title : valid, description : invalid , completed : valid 4. Inputs → title : valid, description : valid , completed : invalid
Expected Result	1. Status → 200 data → task(json) 2. Status → 422 data → “Please enter valid input” 3. Status → 422 data → “Please enter valid input” 4. Status → 422 data → “Please enter valid input”
Actual Result	Matches the Expected results
Verdict (Pass/Fail)	PASS
Code Coverage	

```
describe("Update task with valid and invalid data", () => {
  it("should return 201 status code and todo update successfully! text when todo updated", async () => {
    const response = await request(app).post("/todo/register").send({
      title: "Football Match",
      content: "Football Match at 8 pm in the stadium with friends and family",
      completed: false,
      userId: 11,
    });
    expect(response.status).toBe(201);
    expect(response.text).toBe("Todo update successfully!");
  });

  it("should return 422 status because content is invalid", async () => {
    const response = await request(app).post("/todo/register").send({
      title: "Football Match",
      content:
        "Football Match at 8 pm in the stadium with friends and family Football Match at 8 pm in the stadium with friends and family
      completed: false,
      userId: 11,
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status because complete is invalid ", async () => {
    const response = await request(app).post("/todo/register").send({
      title: "Football Match",
      content: "Football Match at 8 pm in the stadium with friends and family",
      completed: "false", // completed must be boolean
      userId: 11,
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status because userId is invalid", async () => {
    const response = await request(app).post("/todo/register").send({
      title: "Football Match",
      content: "Football Match at 8 pm in the stadium with friends and family",
      completed: false,
      userId: "11", // userId must be number
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
});
```

Table 55 To Do App TC 5 Test Update Task

Get All Task Test Cases

Test Case ID Name	TC6_Get All Task
Preconditions	User is registered and at the login screen.
Test Steps	1.Send GET http request for create a task
Post-Condition	It is assumed that the user has registered and logged in to the to-do application.
Test Data	1. Inputs → userId : valid
Expected Result	1. Status → 200
Actual Result	Matches the Expected results
Verdict (Pass/Fail)	PASS

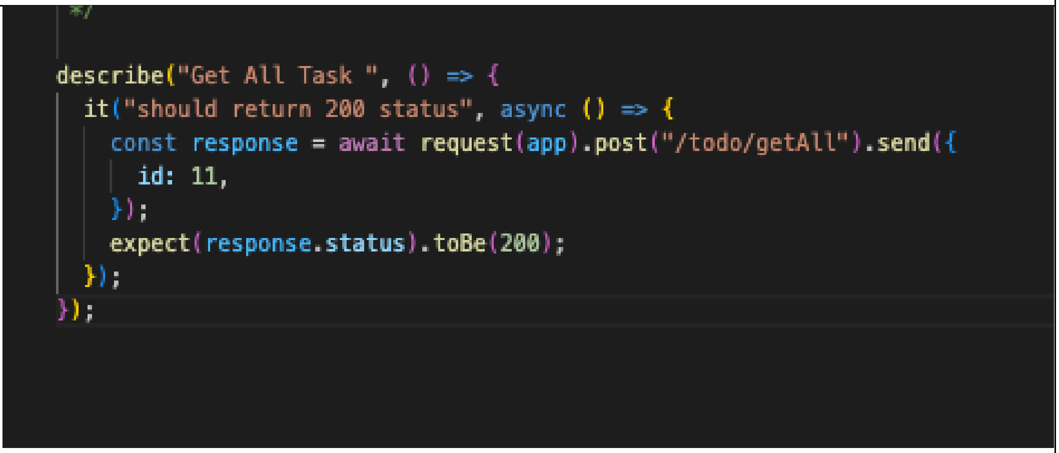
Code Coverage	
	`*/`
	`describe("Get All Task ", () => {`
	`  it("should return 200 status", async () => {`
	`    const response = await request(app).post("/todo/getAll").send({`
	`      id: 11,`
	`    });`
	`    expect(response.status).toBe(200);`
	`  });`
	`});`

Table 3.6 To Do App TC 6 Test View All Tasks

Test Case Description Tables

The test case description tables are below.

Test Scenario	Test Scenario Description	Expected Outcome
1	fullname length is between 1 and 24 characters	Valid
2	fullname length is 0 (empty)	Invalid
3	fullname length is more than 24 characters	Invalid

Table 4.1 Test Case For Fullname

TC1_Register Task with valid and invalid data **Using userRegister method**	
Test Scenario Description	**Expected Outcome**
fullname length = 0	Invalid
fullname length = 1	Valid
fullname length = 24	Valid
fullname length = 25	Invalid

Table 4.2 Test Case For Fullname

Test Scenario	Test Scenario Description	Expected Outcome
1	Password length is between 1 and 24 characters	Valid
2	Password length is 0 (empty)	Invalid
3	Password length is more than 24 characters	Invalid

Table 4.3 Test Case For Password

TC2_Login with valid and invalid data **Using userLogin method**	
Test Scenario Description	**Expected Outcome**
Password length = 0	Invalid
Password length = 1	Valid
Password length = 24	Valid
Password length = 25	Invalid

Table 4.4 Test Case For Password

Test Scenario	Test Scenario Description	Expected Outcome
1	Title length is between 1 and 49 characters	Valid
2	Title length is 0 (empty)	Invalid
3	Title length is more than 49 characters	Invalid

Table 4.5 Test Case For Title

<table>
<tr><th colspan="2">TC3_Create a task with valid and invalid data
<b>Using toDoCreate method</b></th></tr>
<tr><th>Test Scenario Description</th><th>Expected Outcome</th></tr>
<tr><td>Title length = 0</td><td>Invalid</td></tr>
<tr><td>Title length = 1</td><td>Valid</td></tr>
<tr><td>Title length = 49</td><td>Valid</td></tr>
<tr><td>Title length = 50</td><td>Invalid</td></tr>
</table>

Table 4.6 Test Case For Title

Test Scenario	Test Scenario Description	Expected Outcome
1	Content length is between 1 and 99 characters	Valid
2	Content length is 0 (empty)	Invalid
3	Content length is more than 99 characters	Invalid

Table 4.7 Test Case For Content

<table>
<tr><th colspan="2">TC5_Update a task with valid and invalid data
<b>Using toDoUpdate method</b></th></tr>
<tr><th>Test Scenario Description</th><th>Expected Outcome</th></tr>
<tr><td>Content length = 0</td><td>Invalid</td></tr>
<tr><td>Content length = 1</td><td>Valid</td></tr>
<tr><td>Content length = 99</td><td>Valid</td></tr>
<tr><td>Content length = 100</td><td>Invalid</td></tr>
</table>

Table 4.8 Test Case For Content

6. Test implementation

Creating automated test scripts
Not appropriate.

Arranging the test cases within a test execution schedule in a way that results in efficient test execution.
Test cases will be defined as methods.

Building the test environment (including, potentially, test harnesses, service virtualization, simulators, and other infrastructure items) and verifying that everything needed has been set up correctly.
Building test environment is provided by Jest. Service virtualization and simulation are not appropriate. Test Harness is a collection of stubs, drivers and other supporting tools required to automate test execution. In this project we don’t use automation.

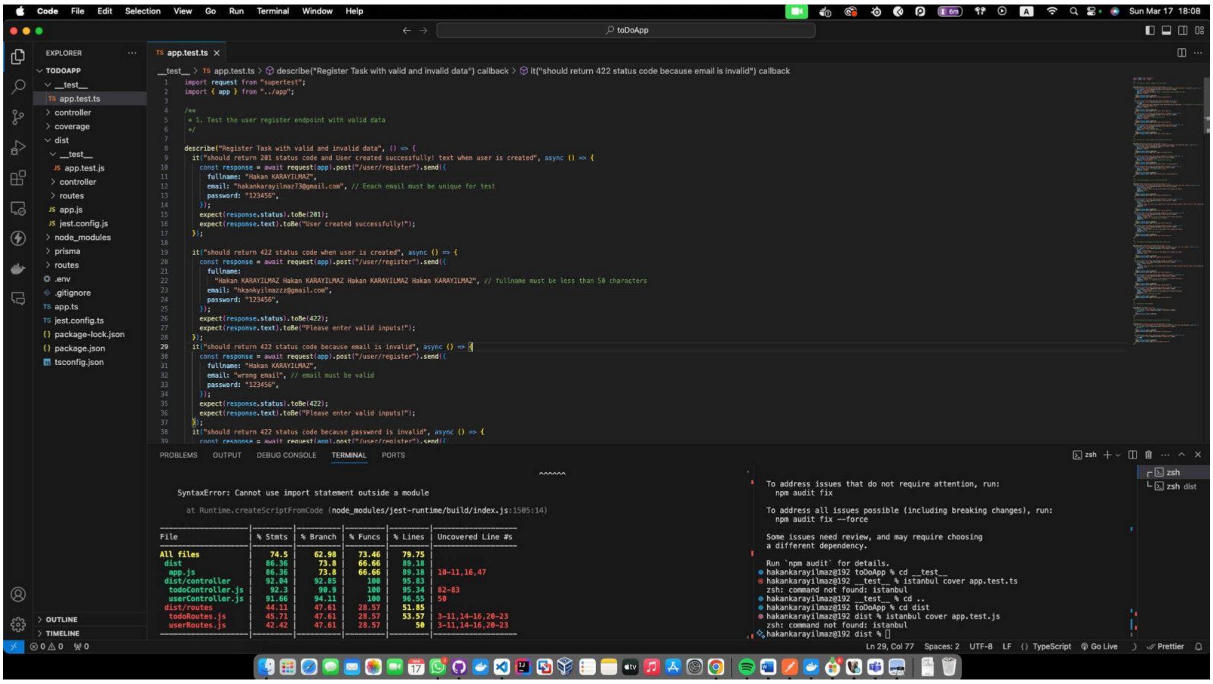
Preparing test data and ensuring it is properly loaded in the test environment.
The main environment and execution tools were VS Code and Jest respectively. VS Code is a space in which software is being tested for a batch of experimental uses. Test data was recorded in database. Test data was prepared by using Postman and DBeaver. For proper loading, source data and target data -i.e., tables, constraints, columns, data types- must be well known.

7. Test execution

Executing tests either manually or by using test execution tools
All tests run with the testing library Jest and execution tool Postman.

Comparing actual results with expected results
Comparing actual results with expected results the reason why To Do App have less than 73.46% coverage is that we focused our attention on testing the crucial requirements. We determined that register, login, and CRUD (Create, Read, Update, Delete) use cases are the most important things to test and due to the lack of time, we only implemented tests for these. The rest of these was outside our scope. We don't have user interface, so we don't have user interface test at the same time we don’t have non-funtional requirements test.

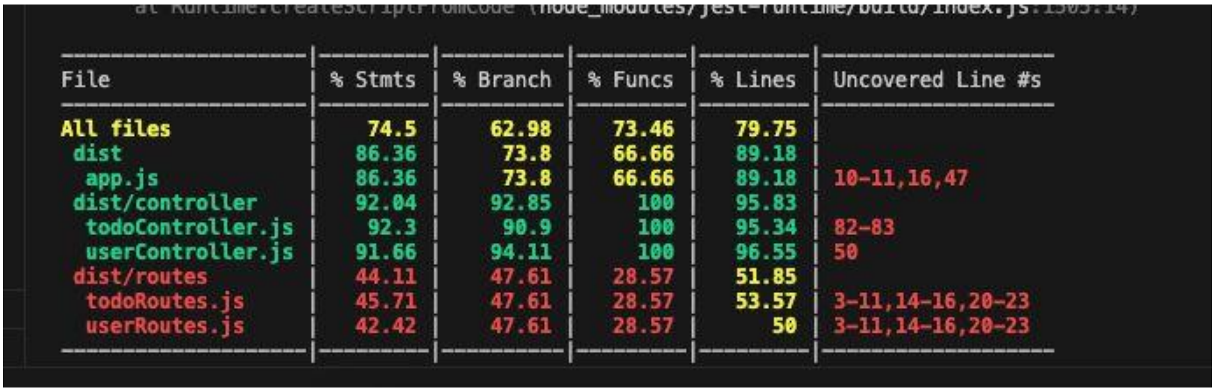
Logging the outcome of test executions based on the failures observed, e.g., pass, fail, blocked.



```
import request from "supertest";
import { app } from "../app";

/**
 * 1. Test the user register endpoint with valid data
 */

describe("Register Task with valid and invalid data", () => {
  it("should return 201 status code and User created successfully! text when user is created", async () => {
    const response = await request(app).post("/user/register").send({
      fullname: "Hakan KARAYILMAZ",
      email: "hakankarayilmaz73@gmail.com", // Eeach email must be unique for test
      password: "123456",
    });
    expect(response.status).toBe(201);
    expect(response.text).toBe("User created successfully!");
  });

  it("should return 422 status code when user is created", async () => {
    const response = await request(app).post("/user/register").send({
      fullname:
        "Hakan KARAYILMAZ Hakan KARAYILMAZ Hakan KARAYILMAZ Hakan KARAYILMAZ", // fullname must be less than 50 characters
      email: "hkankyilmazzz@gmail.com",
      password: "123456",
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status code because email is invalid", async () => {
    const response = await request(app).post("/user/register").send({
      fullname: "Hakan KARAYILMAZ",
      email: "wrong email", // email must be valid
      password: "123456",
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status code because password is invalid", async () => {
    const response = await request(app).post("/user/register").send({
```

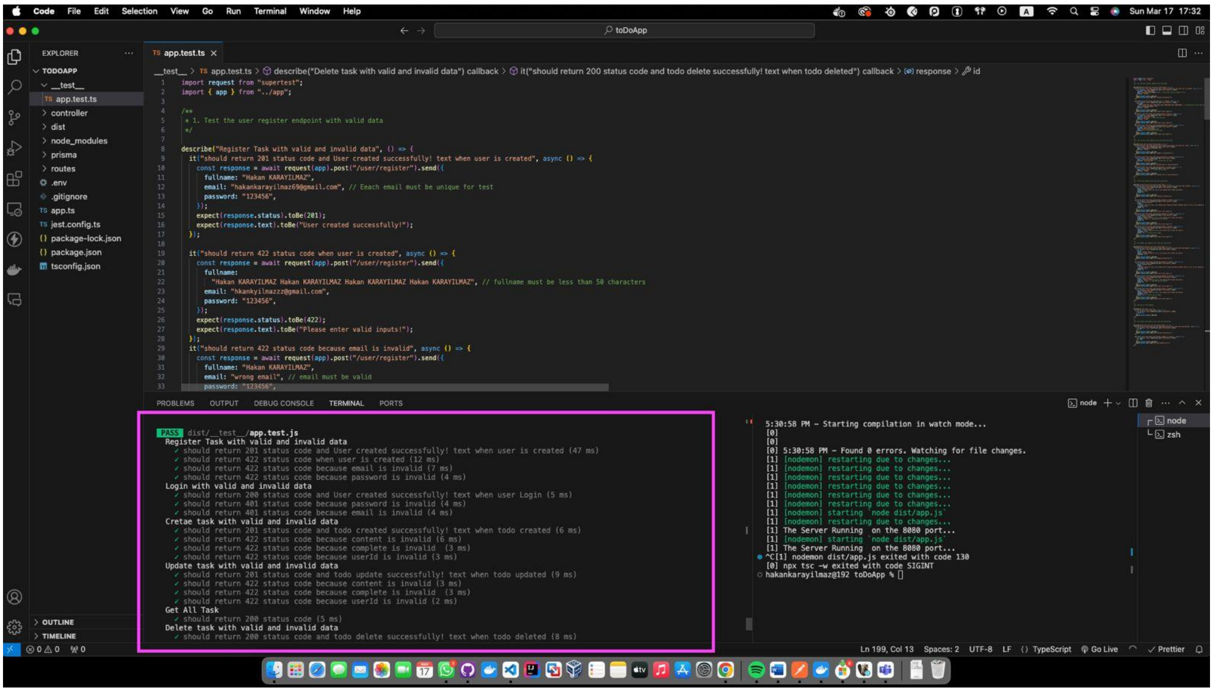
```
                                                  ^^^^^^

    SyntaxError: Cannot use import statement outside a module

      at Runtime.createScriptFromCode (node_modules/jest-runtime/build/index.js:1505:14)

--------------------|---------|----------|---------|---------|-------------------
File                | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------|---------|----------|---------|---------|-------------------
All files           |    74.5 |    62.98 |   73.46 |   79.75 |
 dist               |   86.36 |     73.8 |   66.66 |   89.18 |
  app.js            |   86.36 |     73.8 |   66.66 |   89.18 | 10-11,16,47
 dist/controller    |   92.04 |    92.85 |     100 |   95.83 |
  todoController.js |    92.3 |     90.9 |     100 |   95.34 | 82-83
  userController.js |   91.66 |    94.11 |     100 |   96.55 | 50
 dist/routes        |   44.11 |    47.61 |   28.57 |   51.85 |
  todoRoutes.js     |   45.71 |    47.61 |   28.57 |   53.57 | 3-11,14-16,20-23
  userRoutes.js     |   42.42 |    47.61 |   28.57 |      50 | 3-11,14-16,20-23
--------------------|---------|----------|---------|---------|-------------------
```

```
To address issues that do not require attention, run:
  npm audit fix

To address all issues possible (including breaking changes), run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
hakankarayilmaz@192 toDoApp % cd __test__
hakankarayilmaz@192 __test__ % istanbul cover app.test.ts
zsh: command not found: istanbul
hakankarayilmaz@192 __test__ % cd ..
hakankarayilmaz@192 toDoApp % cd dist
hakankarayilmaz@192 dist % istanbul cover app.test.js
zsh: command not found: istanbul
hakankarayilmaz@192 dist % 
```



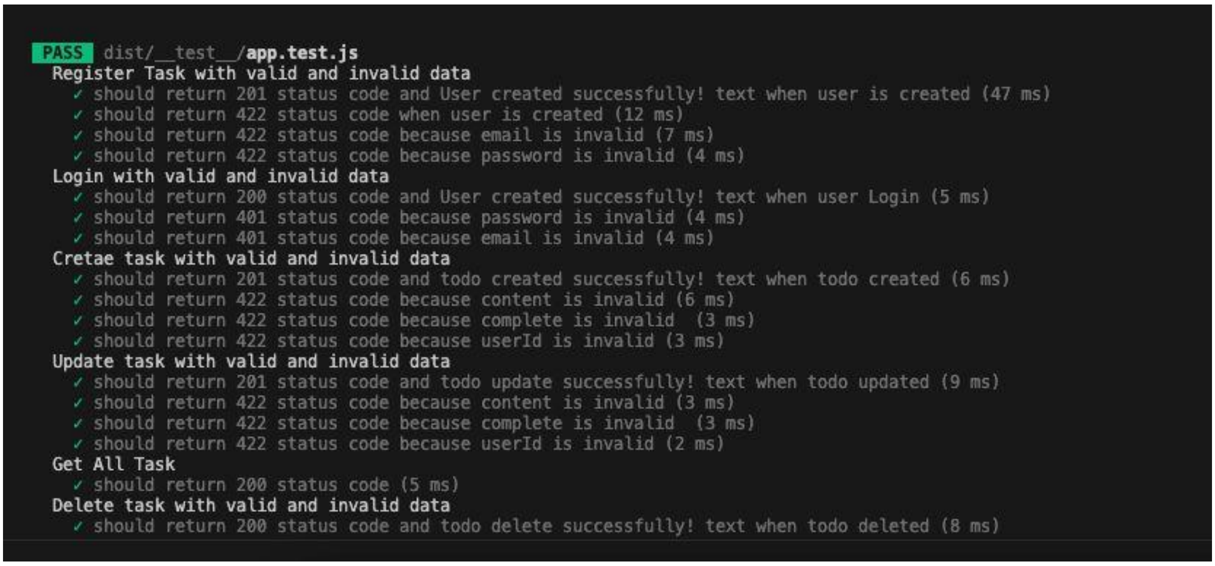
```
  at Runtime.createScriptFromCode (node_modules/jest-runtime/build/index.js:1505:14)

--------------------|---------|----------|---------|---------|-------------------
File                | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------|---------|----------|---------|---------|-------------------
All files           |    74.5 |    62.98 |   73.46 |   79.75 |
 dist               |   86.36 |     73.8 |   66.66 |   89.18 |
  app.js            |   86.36 |     73.8 |   66.66 |   89.18 | 10-11,16,47
 dist/controller    |   92.04 |    92.85 |     100 |   95.83 |
  todoController.js |    92.3 |     90.9 |     100 |   95.34 | 82-83
  userController.js |   91.66 |    94.11 |     100 |   96.55 | 50
 dist/routes        |   44.11 |    47.61 |   28.57 |   51.85 |
  todoRoutes.js     |   45.71 |    47.61 |   28.57 |   53.57 | 3-11,14-16,20-23
  userRoutes.js     |   42.42 |    47.61 |   28.57 |      50 | 3-11,14-16,20-23
--------------------|---------|----------|---------|---------|-------------------
```

Figure 5.1 Coverage Rates of To Do App



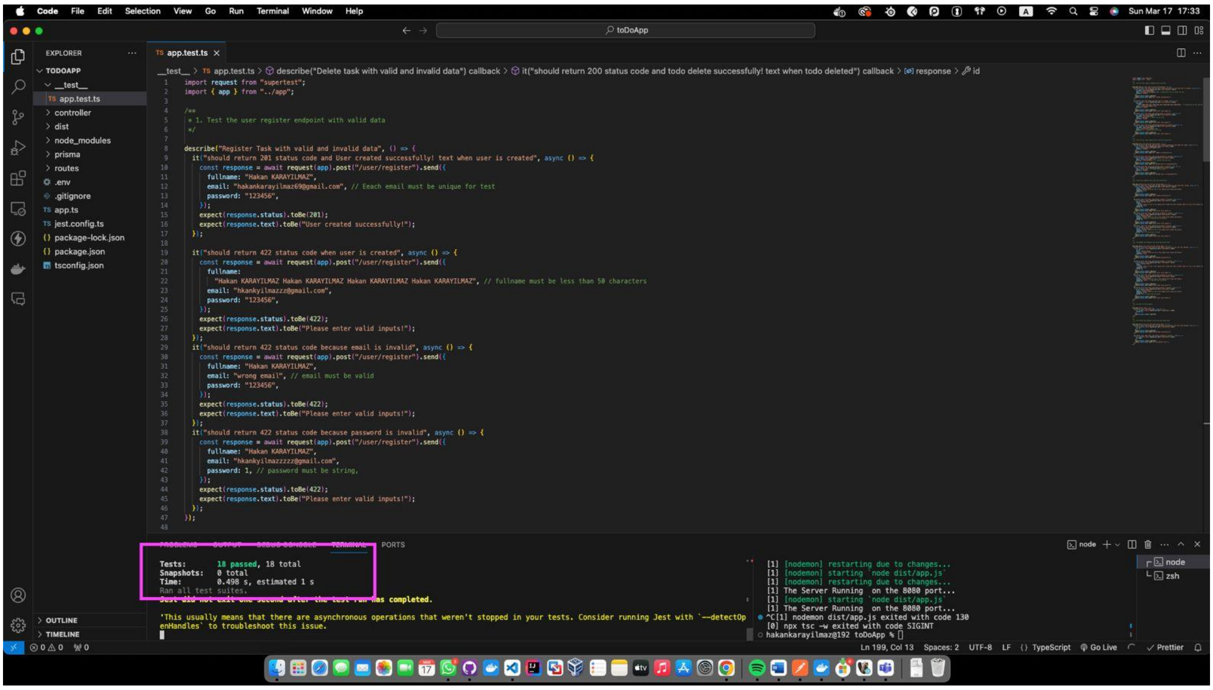
```
import request from "supertest";
import { app } from "../app";

/**
 * 1. Test the user register endpoint with valid data
 */

describe("Register Task with valid and invalid data", () => {
  it("should return 201 status code and User created successfully! text when user is created", async () => {
    const response = await request(app).post("/user/register").send({
      fullname: "Hakan KARAYILMAZ",
      email: "hakankarayilmaz69@gmail.com", // Eeach email must be unique for test
      password: "123456",
    });
    expect(response.status).toBe(201);
    expect(response.text).toBe("User created successfully!");
  });

  it("should return 422 status code when user is created", async () => {
    const response = await request(app).post("/user/register").send({
      fullname:
        "Hakan KARAYILMAZ Hakan KARAYILMAZ Hakan KARAYILMAZ Hakan KARAYILMAZ", // fullname must be less than 50 characters
      email: "hkankyilmazzz@gmail.com",
      password: "123456",
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status code because email is invalid", async () => {
    const response = await request(app).post("/user/register").send({
      fullname: "Hakan KARAYILMAZ",
      email: "wrong email", // email must be valid
      password: "123456",
```

```
5:30:58 PM - Starting compilation in watch mode...
[0]
[0]
[0] 5:30:58 PM - Found 0 errors. Watching for file changes.
[1] [nodemon] restarting due to changes...
[1] [nodemon] restarting due to changes...
[1] [nodemon] restarting due to changes...
[1] [nodemon] restarting due to changes...
[1] [nodemon] restarting due to changes...
[1] [nodemon] restarting due to changes...
[1] [nodemon] starting `node dist/app.js`
[1] [nodemon] restarting due to changes...
[1] The Server Running  on the 8080 port...
[1] [nodemon] starting `node dist/app.js`
[1] The Server Running  on the 8080 port...
^C[1] nodemon dist/app.js exited with code 130
[0] npx tsc -w exited with code SIGINT
hakankarayilmaz@192 toDoApp % 
```



```
PASS dist/__test__/app.test.js
  Register Task with valid and invalid data
    ✓ should return 201 status code and User created successfully! text when user is created (47 ms)
    ✓ should return 422 status code when user is created (12 ms)
    ✓ should return 422 status code because email is invalid (7 ms)
    ✓ should return 422 status code because password is invalid (4 ms)
  Login with valid and invalid data
    ✓ should return 200 status code and User created successfully! text when user Login (5 ms)
    ✓ should return 401 status code because password is invalid (4 ms)
    ✓ should return 401 status code because email is invalid (4 ms)
  Cretae task with valid and invalid data
    ✓ should return 201 status code and todo created successfully! text when todo created (6 ms)
    ✓ should return 422 status code because content is invalid (6 ms)
    ✓ should return 422 status code because complete is invalid  (3 ms)
    ✓ should return 422 status code because userId is invalid (3 ms)
  Update task with valid and invalid data
    ✓ should return 201 status code and todo update successfully! text when todo updated (9 ms)
    ✓ should return 422 status code because content is invalid (3 ms)
    ✓ should return 422 status code because complete is invalid  (3 ms)
    ✓ should return 422 status code because userId is invalid (2 ms)
  Get All Task
    ✓ should return 200 status code (5 ms)
  Delete task with valid and invalid data
    ✓ should return 200 status code and todo delete successfully! text when todo deleted (8 ms)
```

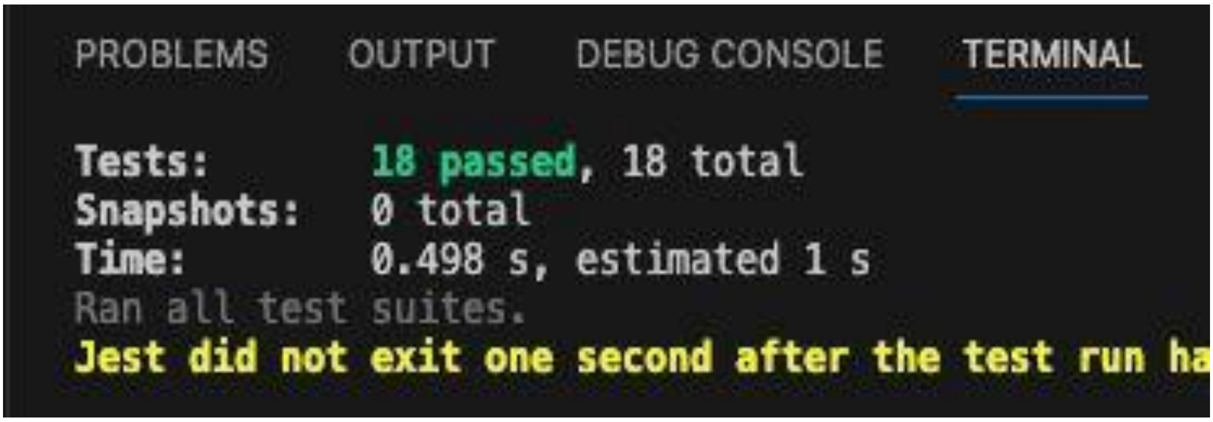
Figure 5.2 Detail of All Tests of To Do App



```typescript
import request from "supertest";
import { app } from "../app";

/**
 * 1. Test the user register endpoint with valid data
 */

describe("Register Task with valid and invalid data", () => {
  it("should return 201 status code and User created successfully! text when user is created", async () => {
    const response = await request(app).post("/user/register").send({
      fullname: "Hakan KARAYILMAZ",
      email: "hakankarayilmaz69@gmail.com", // Eeach email must be unique for test
      password: "123456",
    });
    expect(response.status).toBe(201);
    expect(response.text).toBe("User created successfully!");
  });

  it("should return 422 status code when user is created", async () => {
    const response = await request(app).post("/user/register").send({
      fullname:
        "Hakan KARAYILMAZ Hakan KARAYILMAZ Hakan KARAYILMAZ Hakan KARAYILMAZ", // fullname must be less than 50 characters
      email: "hkankyilmazzz@gmail.com",
      password: "123456",
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status code because email is invalid", async () => {
    const response = await request(app).post("/user/register").send({
      fullname: "Hakan KARAYILMAZ",
      email: "wrong email", // email must be valid
      password: "123456",
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
  it("should return 422 status code because password is invalid", async () => {
    const response = await request(app).post("/user/register").send({
      fullname: "Hakan KARAYILMAZ",
      email: "hkankyilmazzzzz@gmail.com",
      password: 1, // password must be string,
    });
    expect(response.status).toBe(422);
    expect(response.text).toBe("Please enter valid inputs!");
  });
});
```

```
Tests:       18 passed, 18 total
Snapshots:   0 total
Time:        0.498 s, estimated 1 s
Ran all test suites.
Jest did not exit one second after the test run has completed.

'This usually means that there are asynchronous operations that weren't stopped in your tests. Consider running Jest with `--detectOp
enHandles` to troubleshoot this issue.
```

```
[1] [nodemon] restarting due to changes...
[1] [nodemon] starting `node dist/app.js`
[1] [nodemon] restarting due to changes...
[1] The Server Running  on the 8080 port...
[1] [nodemon] starting `node dist/app.js`
[1] The Server Running  on the 8080 port...
^C[1] nodemon dist/app.js exited with code 130
[0] npx tsc -w exited with code SIGINT
hakankarayilmaz@192 toDoApp %
```



```
Tests:       18 passed, 18 total
Snapshots:   0 total
Time:        0.498 s, estimated 1 s
Ran all test suites.
Jest did not exit one second after the test run ha
```

Figure 5.3 All Test Results of To Do App

8. Test completion

Creating a test summary report to be communicated to stakeholders,
All tests were successful and ran without problem.

Finalizing and archiving the test environment, the test data, the test infrastructure, and other testware [1]for later reuse
ToDoApp team project document is submitted as SW-T-(IT-538) -To-Do-App-Test_Specification-v2024.3.docx file and ToDoApp VS Code project is archived as CoffeeMachinev2024.3.zip file. For submission they will be archived with the rest of the guiding templates under SWT-ToDoApp-Project-v2024.3.zip file.

Handing over the testware to the stakeholders
After creating the ToDoAppv2024.3.zip file, SW-T-(IT-538)-To-Do-App-Test_Specification-v2024.3.docx and ToDoAppv2024.3.zip files are going to be archived with the guiding documents in SWT-ToDoApp-Project-v2024.3.zip file and will be uploaded to SUCOURSE.

Analyzing lessons learned from the completed test activities to determine changes needed for future iterations, releases, and projects.
In this project, we tested the To Do App. If we could test all the modules, we would be able to understand the whole system and see the big picture. In addition, we would be able to test the relationship between modules with the integration test.

We learned testing principles, test processes, test techniques like equivalence partitioning and boundary value analysis, General V-Model, testing with agile methodology. We also had the chance to practice using the Spring Boot by coding the test cases we wrote.

Checking whether all work products are finalized.
All products that are this document and To Do App VS Code project, are finalized as SW-T-(IT-538)-To-Do-App-Test_Specification-v2024.3.docx and ToDoAppv2024.3.zip respectively and are going to be added to SUCOURSE along with the guiding documents provided to us.

[1] **Testware** : Artifacts produced during the test process required to plan, design, and execute tests, such as documentation, scripts, inputs, expected results, set-up and clear-up procedures, files, databases, environment, and any additional software or utilities used in testing.