Deliverable-5: Phase 3 Report

Table of Contents

a. Requirements

Phase 3 (Deliverable 5) Requirements

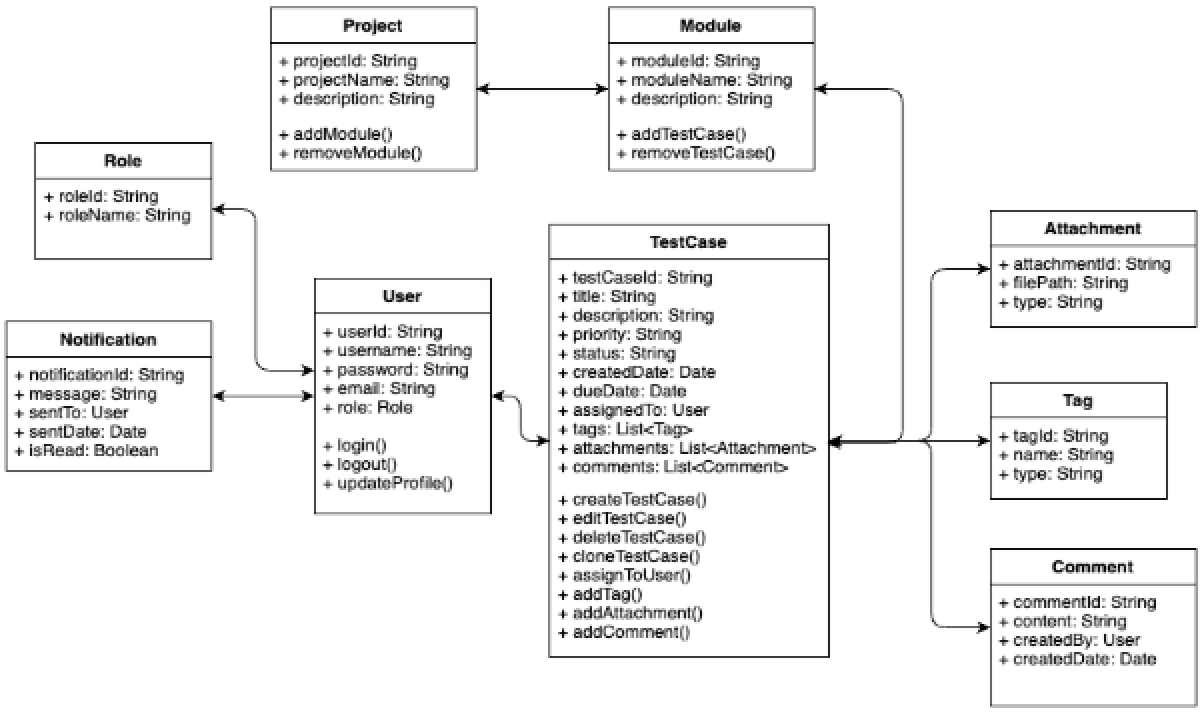
The scope for Phase 3 remains changed from Deliverable 4. Below are the detailed requirements for Phase 3, with expanded descriptions:

1. **Email Notifications**:
    a. Implement a system to send email alerts to users for task deadlines, status changes, or critical issues.
    b. Enable email notifications for task assignments, with a templates including task details (e.g., title, description, due date, assignee).
    c. Ensure notifications are triggered based on predefined events (e.g., new task assigned, updated assignee).
2. **Dashboard Charts**:
    a. Develop interactive charts on the dashboard to visualize project metrics, such as test case completion, status, assigned users, priority deadlines and so on.
    b. Support multiple chart types (e.g., bar, pie, line) using a library like Chart.js.
    c. Allow users to filter test cases by project, status, modules, or team member.
    d. Ensure charts are responsive and accessible across devices.
3. **Report Generation**:
    a. Enable generation of reports in PDF and CSV formats, summarizing project progress, task statuses, and team performance.
    b. Include customizable report templates with options to select specific data fields (e.g., completed tasks, overdue tasks).
    c. Provide export functionality accessible from the dashboard.
    d. Ensure reports are downloadable and shareable via email.
4. **Testing**:
    a. Conduct manual testing to verify application functionality, focusing on new features (alerts, charts, reports).
    b. Perform integration testing to ensure seamless interaction between Phase 3 components and existing features from Phases 1 and 2.
    c. Execute performance testing to assess application scalability under load (e.g., 100 concurrent users).
    d. Conduct User Acceptance Testing (UAT) with stakeholders to validate usability and functionality.
    e. Assign test cases to team members and notify assignees via email.
5. **Release Beta Version**:

a. Package the application as a beta release, including all features implemented in Phases 1–3.
b. Deploy the beta version to a staging environment for user feedback.
c. Provide documentation and instructions for beta testers.

b. UML Design for Phase 3

Class Diagram

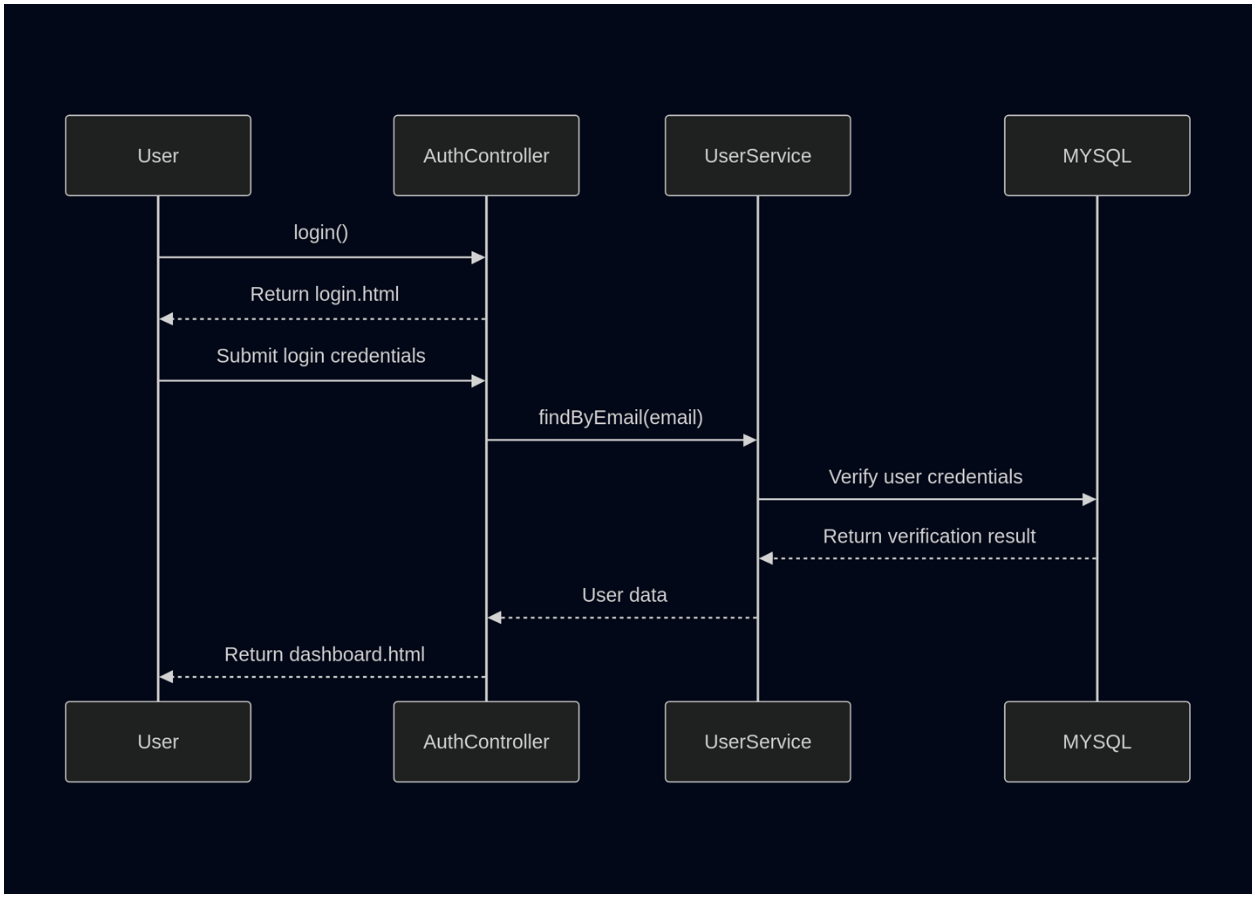


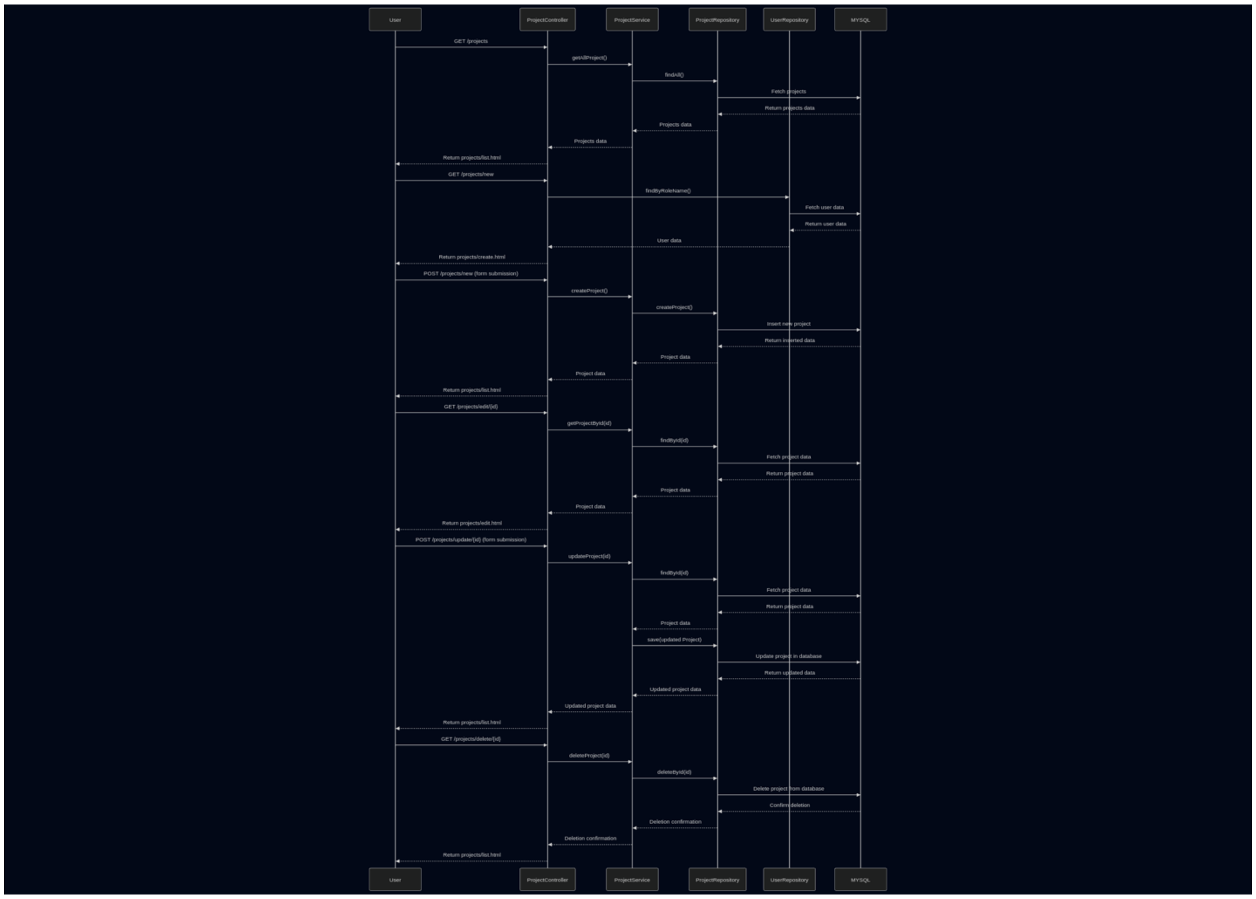
- **Classes**:
  - User: Attributes (id, name, email); Methods (register()).
  - TestCase: Attributes (id, title, description, dueDate, assignee, status); Methods (create(), update(), delete(), find(), notifyAssignee()).
  - Notification: Attributes (id, recipient, message); Methods (send()).

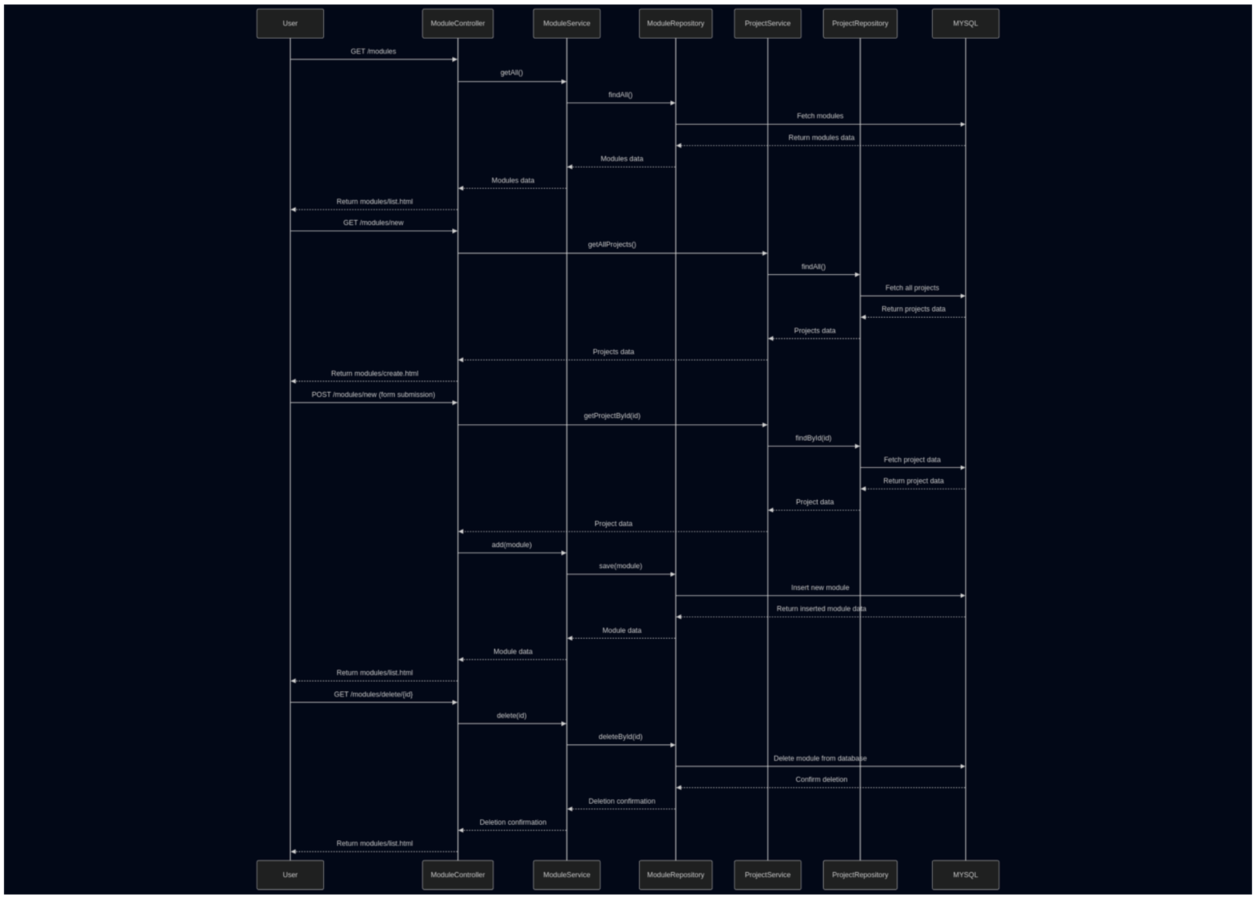
  - Dashboard: Attributes (id, charts); Methods (getChartData(), getPriorityChartData(), getCreationTrend(), getAssignedCounts(), getStatusPriorityDistribution()).
  - Report: Attributes (id, type, data, format); Methods (generate(), export()).
- **Relationships**:
  - User has many TestCases(1:N).
  - TestCases triggers Notification (1:N).
  - Dashboard contains multiple Charts (1:N).
  - Report is generated from TestCase (N:1).

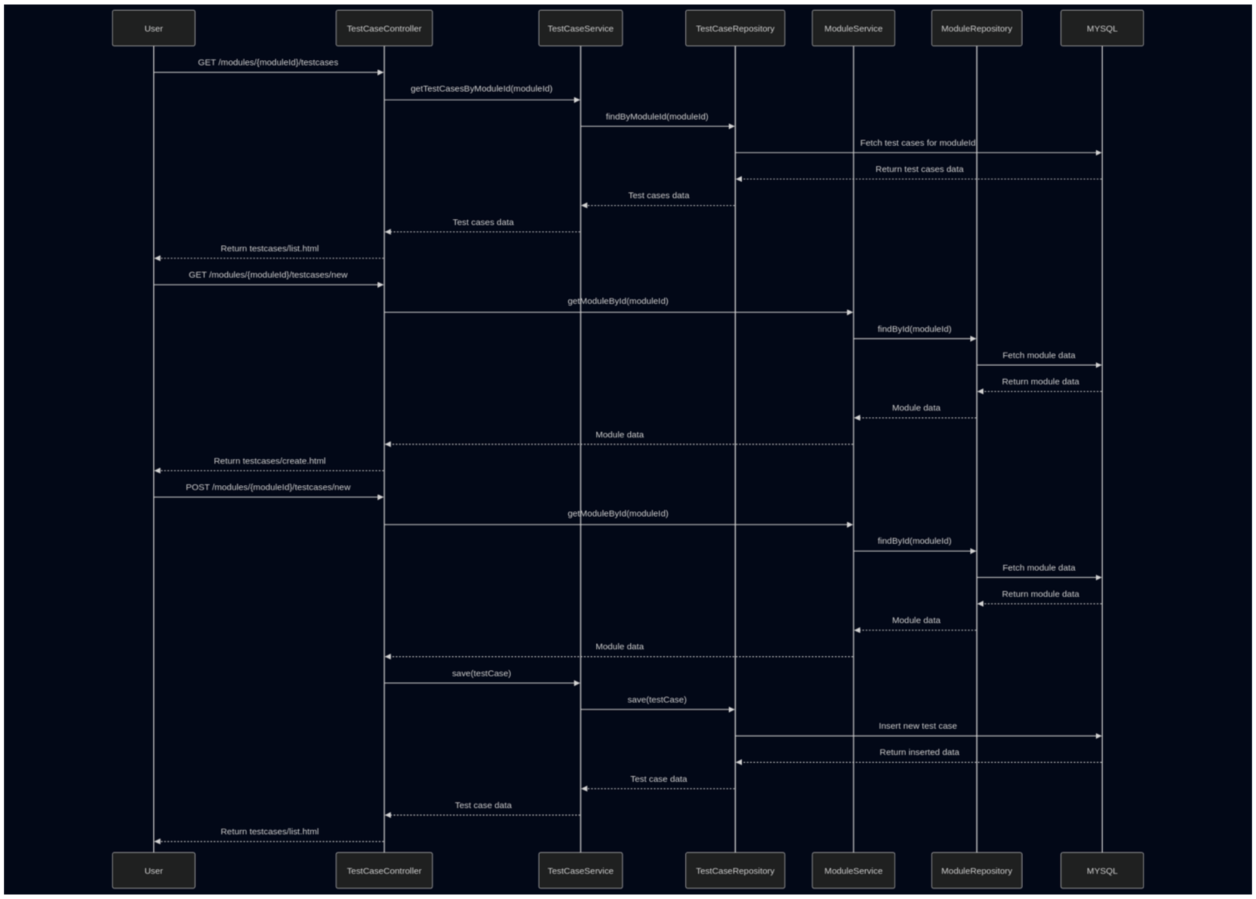
Sequence Diagram

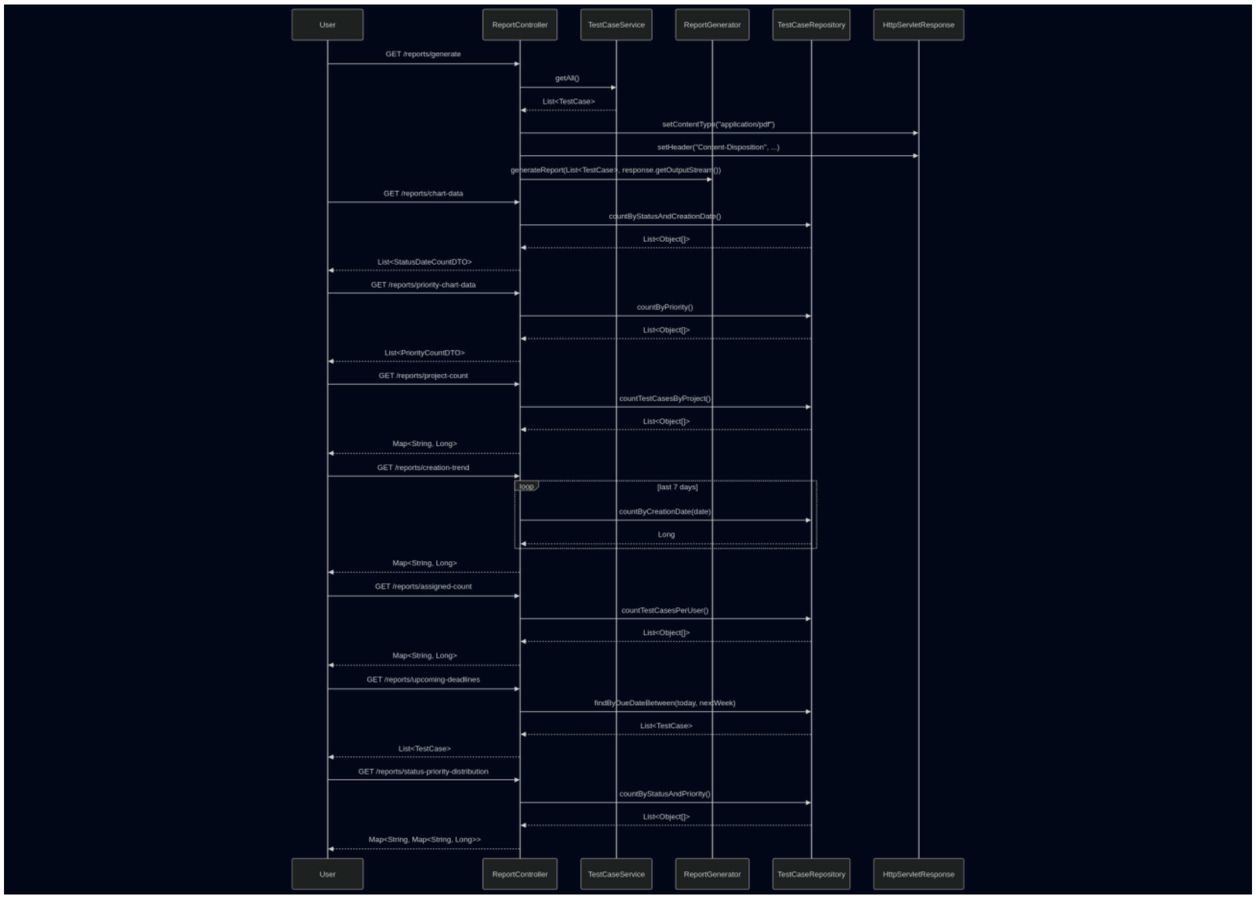
- Illustrates the sequence of interactions for creating a test case:
  - User submits test case form via UI.
  - Controller receives the request and calls the service layer.
  - Service layer validates input and interacts with the repository.
  - Repository saves the test case to the database.
  - Response is returned to the UI, displaying success or error.
- **The sequence diagram is extremely large, so it is recommended to download the image from the document.**


User
AuthController
UserService
MYSQL
login()
Return login.html
Submit login credentials
findByEmail(email)
Verify user credentials
Return verification result
User data
Return dashboard.html
User
AuthController
UserService
MYSQL

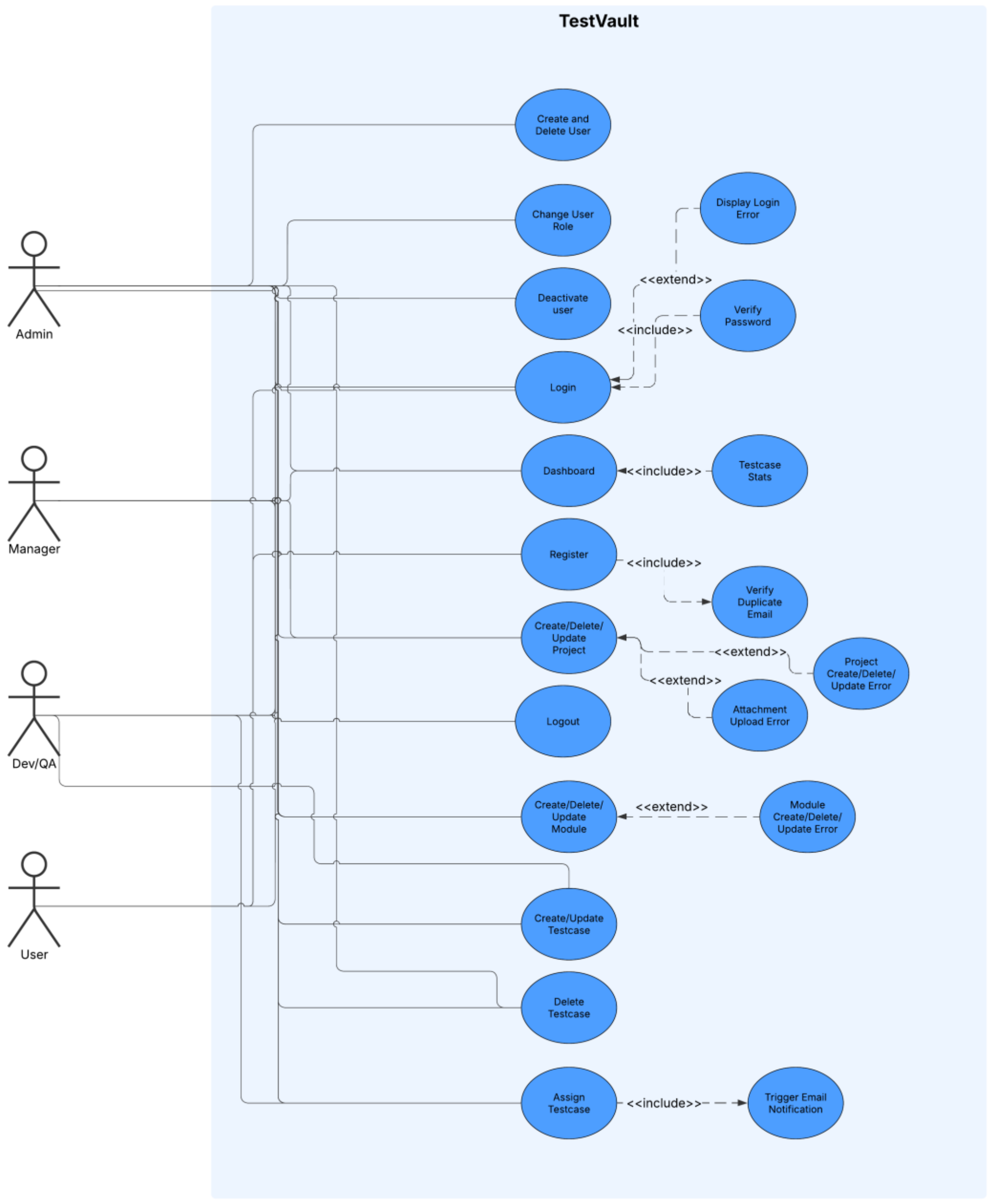

User
ProjectController
ProjectService
ProjectRepository
UserRepository
MYSQL
GET /projects
getAllProject()
findAll()
Fetch projects
Return projects data
Projects data
Projects data
Return projects/list.html
GET /projects/new
findByRoleName()
Fetch user data
Return user data
User data
Return projects/create.html
POST /projects/new (form submission)
createProject()
createProject()
Insert new project
Return inserted data
Project data
Project data
Return projects/list.html
GET /projects/edit/{id}
getProjectById(id)
findById(id)
Fetch project data
Return project data
Project data
Project data
Return projects/edit.html
POST /projects/update/{id} (form submission)
updateProject(id)
findById(id)
Fetch project data
Return project data
Project data
save(updated Project)
Update project in database
Return updated data
Updated project data
Updated project data
Return projects/list.html
GET /projects/delete/{id}
deleteProject(id)
deleteById(id)
Delete project from database
Confirm deletion
Deletion confirmation
Deletion confirmation
Return projects/list.html
User
ProjectController
ProjectService
ProjectRepository
UserRepository
MYSQL


User
ModuleController
ModuleService
ModuleRepository
ProjectService
ProjectRepository
MYSQL
GET /modules
getAll()
findAll()
Fetch modules
Return modules data
Modules data
Modules data
Return modules/list.html
GET /modules/new
getAllProjects()
findAll()
Fetch all projects
Return projects data
Projects data
Projects data
Return modules/create.html
POST /modules/new (form submission)
getProjectById(id)
findById(id)
Fetch project data
Return project data
Project data
Project data
add(module)
save(module)
Insert new module
Return inserted module data
Module data
Module data
Return modules/list.html
GET /modules/delete/{id}
delete(id)
deleteById(id)
Delete module from database
Confirm deletion
Deletion confirmation
Deletion confirmation
Return modules/list.html
User
ModuleController
ModuleService
ModuleRepository
ProjectService
ProjectRepository
MYSQL


User
TestCaseController
TestCaseService
TestCaseRepository
ModuleService
ModuleRepository
MYSQL
GET /modules/{moduleId}/testcases
getTestCasesByModuleId(moduleId)
findByModuleId(moduleId)
Fetch test cases for moduleId
Return test cases data
Test cases data
Test cases data
Return testcases/list.html
GET /modules/{moduleId}/testcases/new
getModuleById(moduleId)
findById(moduleId)
Fetch module data
Return module data
Module data
Module data
Return testcases/create.html
POST /modules/{moduleId}/testcases/new
getModuleById(moduleId)
findById(moduleId)
Fetch module data
Return module data
Module data
Module data
save(testCase)
save(testCase)
Insert new test case
Return inserted data
Test case data
Test case data
Return testcases/list.html
User
TestCaseController
TestCaseService
TestCaseRepository
ModuleService
ModuleRepository
MYSQL



Use Case Diagram



- **Actors**

  - A registered user who accesses system functions that relate to test cases and project units.
  - The admin user possesses special permissions to control role assignments and manage system configuration.

- **Main** Use Cases for All Users.

  - Login – Access the system with credentials.

- Make a new user profile through the register page.
- Logout – Exit the application securely.
- The main dashboard shows the important controls for both navigation and user tools.

- **Project Management**
  Define new projects and modify existing ones to maintain all project information.

- **Module Management**
  Run tasks to handle project components including adding or deleting them.

- **Test Case Management**
  - Our system enables users to create update and remove all their test cases.
  - To help testing you may submit image or PDF documents.
  - Click tags to track and use filter-based test case views.
  - Check all available test cases organized by modules in the system.

- **Admin**-Only Use Cases
  - Admins should distribute test cases among team members for effective work assignments.
  - Design who can handle administrative or user tasks.
  - Kickstart the seed system by loading predefined users and test examples into it.
  - Check system performance through test runs to see how it works.
  - Accept suggestions from inspections by adding them to your code.

c. Test Cases for Phase 3

Unit Tests

Below is a list of unit test cases for Phase 3 Deliverable 5, including descriptions, functionality tested, and inputs/outputs.

Test Case No	**Class**	**Method Type**	**Description**	**Input**	**Expected Result**
TC-001	ReportController	generateReport (GET)	Generate and download a PDF report of test cases	HTTP Request to /reports/generate	Response is a PDF file named test-cases-report.pdf
TC-002	ReportController	getChartData (GET)	Get test case count grouped by status and creation date	HTTP Request to /reports/chart-data	Returns a list of StatusDateCountDTO JSON objects
TC-003	ReportController	getPriorityChartData (GET)	Get test case count grouped by priority	HTTP Request to /reports/priority-chart-data	Returns a list of PriorityCountDTO JSON objects
TC-004	ReportController	getProjectCounts (GET)	Get the count of test cases per project	HTTP Request to /reports/project-count	Returns a map of project names and their respective counts
TC-005	ReportController	getCreationTrend (GET)	Get number of test cases created each day over the last 7 days	HTTP Request to /reports/creation-trend	Returns a map of date strings to counts
TC-006	ReportController	getAssignedCounts (GET)	Get the number of test cases assigned to each user	HTTP Request to /reports/assigned-count	Returns a map of usernames and their assigned test case counts
TC-007	ReportController	getUpcomingDea	Get test cases with due dates	HTTP Request to /reports/upc	Returns a list of TestCase JSON

		dlines (GET)	within the next 7 days	oming-deadlines	objects within next 7 days
TC-008	ReportController	getStatusPriorityDistribution (GET)	Get test case distribution by status and priority	HTTP Request to /reports/status-priority-distribution	Returns a nested map with status as key and priority-count mappings as values
TC-009	EmailServiceImpl	sendAssignmentNotification	Send assignment notification email successfully	User, TestCase (with Date), HttpServletRequest	Email is sent via JavaMailSender
TC-010	EmailServiceImpl	sendAssignmentNotification	Handle exception during email sending	User, TestCase (with Date), HttpServletRequest (and simulate error)	Exception is logged, no crash occurs
TC-011	AttachmentController	POST	Upload valid attachment	Valid file (MultipartFile)	Returns status 200 with success message and UUID
TC-012	AttachmentController	POST	Upload empty file	Empty file	Returns status 400 with “No file selected” message
TC-013	AttachmentController	GET	View attachment by UUID	Valid UUID	Returns status 200 with file data and proper Content-Disposition
TC-014	AttachmentController	GET	View attachment with non-	Non-existing UUID	Returns status 404 “Not Found”

			existing UUID		
TC-015	AttachmentController	GET	Get attachments for an existing test case	Valid TestCase ID	Returns status 200 with a list of attachments
TC-016	AttachmentController	GET	Get attachments for a non-existing test case	Non-existing TestCase ID	Returns status 404 “Not Found”

The test case classes, and the methods can be found at the end of the inspection-code-fantastic-six.pdf provided in the repository of the project.
To, view them, follow the link:
https://github.com/my-unt-projects/TestVault/blob/main/inspection-code-fantastic-six.pdf

System (functional) Testing

Login/Registration

Test Case	**Test Scenario**	**Expected Outcome**	**Test Description**
Test 1: Show Login Page (GET /login)	When accessing the login page URL.	The system should return the login page.	Verifies that the login page (login.html) is displayed correctly when a user navigates to /login.
Test 2: Show Registration Form (GET /register)	When accessing the registration page URL.	The system should return the registration page with an empty UserDto object in the model.	Verifies that the registration form (register.html) is correctly rendered with an empty user model for input.

Test 3: Successful User Registration (POST /register/save)	When submitting a valid user registration form with new user details.	The system should save the user and redirect to the register?success page.	Verifies that when valid user data is submitted, a new user is created and redirected to the success page.
Test 4: User Registration with Existing Email (POST /register/save)	When submitting a user registration form with an already registered email.	The system should reject the registration, show a validation error for the email, and stay on the registration page.	Verifies that when an email already exists in the system, a validation error is triggered, preventing the user from registering again.
Test 5: Registration Form Validation Errors (POST /register/save)	When submitting an incomplete or invalid user registration form (e.g., missing fields).	The system should show the registration page with error messages next to the invalid fields.	Verifies that invalid input (e.g., missing required fields) triggers appropriate validation errors on the registration form.

Project Management

Test Case	**Test Scenario**	**Expected Outcome**	**Test Description**
Test 1: Get All Projects (GET /projects)	When accessing the projects list page URL.	The system should return a list of all projects in the model and render the projects/list view.	Verifies that when a user navigates to /projects, a list of all projects is fetched and displayed correctly.
Test 2: Show Create Project Form (GET /projects/new)	When accessing the new project	The system should return a project creation	Verifies that the project creation form (projects/create.html) is

	creation form URL.	form with title, status options, and available managers.	displayed with necessary attributes, including a list of managers.
Test 3: Create Project (POST /projects)	When submitting a valid new project creation form with project details.	The system should create the project and redirect to the project list page (/projects).	Verifies that when valid project data is submitted, a new project is created, and the user is redirected to the project list.
Test 4: Show Edit Project Form (GET /projects/edit/{id})	When accessing the edit project page with an existing project ID.	The system should return the project edit form with the project details, managers, and status options.	Verifies that when navigating to /projects/edit/{id}, the project edit page is displayed with the correct project data and available managers.
Test 5: Update Project (POST /projects/update/{id})	When submitting the edited project form with updated project details.	The system should update the project and redirect to the project list page (/projects).	Verifies that when an updated project form is submitted, the changes are saved, and the user is redirected to the project list.
Test 6: Delete Project (GET /projects/delete/{id})	When accessing the delete project URL with an existing project ID.	The system should delete the project and redirect to the project list page (/projects).	Verifies that when a project delete request is made, the project is removed, and the user is redirected to the project list.

Module Management

Test Case	**Test Scenario**	**Expected Outcome**	**Test Description**
Test 1: Get All Modules (GET /modules)	When accessing the modules list page URL.	The system should return a list of all modules, grouped by project, and render the modules/list view.	Verifies that when a user navigates to /modules, a list of all modules is fetched, grouped by project, and displayed correctly.
Test 2: Show Create Module Form (GET /modules/new)	When accessing the new module creation form URL.	The system should return a module creation form with title and a list of available projects.	Verifies that the module creation form (modules/create.html) is displayed with the necessary attributes, including available projects.
Test 3: Create Module (POST /modules)	When submitting a valid new module creation form with module details.	The system should create the module and redirect to the module list page (/modules).	Verifies that when valid module data is submitted, a new module is created and the user is redirected to the module list.
Test 4: Show Edit Module Form (GET /modules/edit/{id})	When accessing the edit module page with an existing module ID.	The system should return the module edit form with the module details and available projects.	Verifies that when navigating to /modules/edit/{id}, the module edit page is displayed with the correct module data and available projects.

Test 5: Update Module (POST /modules/update/{id})	When submitting the edited module form with updated module details.	The system should update the module and redirect to the module list page (/modules).	Verifies that when an updated module form is submitted, the changes are saved, and the user is redirected to the module list.
Test 6: Delete Module (GET /modules/delete/{id})	When accessing the delete module URL with an existing module ID.	The system should delete the module and redirect to the module list page (/modules).	Verifies that when a module delete request is made, the module is removed, and the user is redirected to the module list.
Test 7: Get Modules By Project (GET /modules/by-project/{projectId})	When accessing the modules by project URL with an existing project ID.	The system should return a list of modules belonging to the project in the form of ModuleDto objects.	Verifies that when the user requests modules by project ID, the correct modules are returned in JSON format.

Test Case Management

Test Case	Test Scenario	Expected Outcome	Test Description
showCreateForm	Verifying that the form for creating a new test case is shown correctly.	The form should display an empty TestCaseDto and include dropdown lists for selecting a project, user, and tags. The page title should be "Create Test Case."	This test ensures that when a user navigates to the test creation page (/tests/new), they are presented with a form containing all necessary fields, including projects,

			users, and tags for selection.
createTestCase	Verifying that a new test case is created successfully.	The test case is saved in the database with correct attributes (title, description, priority, status, due date, etc.). If an assignee is selected, an email notification is sent. The user is redirected to /tests/all after creation.	This test verifies that when a user submits the form to create a test case, the data is saved properly, including any attachments, and the user is redirected to the test case list. If the assignee is different from the logged-in user, an email is sent.
updateTestCase	Verifying that an existing test case can be updated successfully.	The test case is updated with new details (title, description, priority, status, due date, etc.). If the assignee changes, an email notification is sent. The user is redirected to the updated test case details page.	This test checks the functionality of updating a test case. After submitting updated details, the test case is saved, and if the assignee changes, an email is triggered. The user should be redirected to the test case's detailed page after the update is completed.
getAllTestCases	Verifying that all test cases are listed with proper filtering options (project, module, status, assigned user).	The test cases are displayed with filtering options, and the list is filtered based on the selected criteria. The view includes test case details such as title, description, status, priority, and the ability to filter by project, module, and assignee.	This test ensures that users can view a list of all test cases filtered by project, module, status, and assigned user. The appropriate filtering options should be available, and the list should reflect the applied filters.
editTestCase	Verifying that an existing test case's details can be edited and pre-filled in the form.	The form is populated with the current values of the selected test case, allowing the user to edit and save the changes. The page title should be "Edit Test Case."	This test ensures that when a user clicks on the "edit" button for a test case, the details of the test case are pre-filled in an editable form. After editing, the user should be able to save changes and the updated test case should reflect the new data.
showTestCase	Verifying that a specific test case's details are	The selected test case's details are displayed, including title,	This test checks that when a user clicks on a test case, they are directed to a

	displayed correctly.	description, priority, status, assigned user, and tags.	page where the full details of that test case are displayed, including title, description, assigned user, and tags.
deleteTestCase	Verifying that a test case can be deleted by users with proper roles.	Only users with ADMIN or MANAGER roles can delete a test case. After deletion, the user is redirected to the /tests/all page.	This test ensures that a test case can only be deleted by users with ADMIN or MANAGER roles. After successful deletion, the user should be redirected to the list of all test cases. Unauthorized users should be denied access to the delete operation.

d. User Manual

This section provides instructions for end-users on how to install and use the application.

System Requirements

- Java 17
- Maven 3
- Any IDE (e.g., IntelliJ IDEA, VSCode)
- Postgres installed on your PC (recommended: use Docker for Postgres setup. Compose file is provided in the project)
- Docker Desktop (if using Docker for Postgres)

Installation Steps

1. Clone the repository from the main branch (all code is merged in the main branch).
2. Ensure Java 17 and Maven 3 are installed on your PC.
3. Install Docker Desktop if you prefer to use Docker for PostgreSQL.
4. Navigate to the project directory and run the following command to start PostgreSQL using Docker:

```
docker-compose up -d
```

5. Run the application using Maven:

```
./mvnw spring-boot:run
```

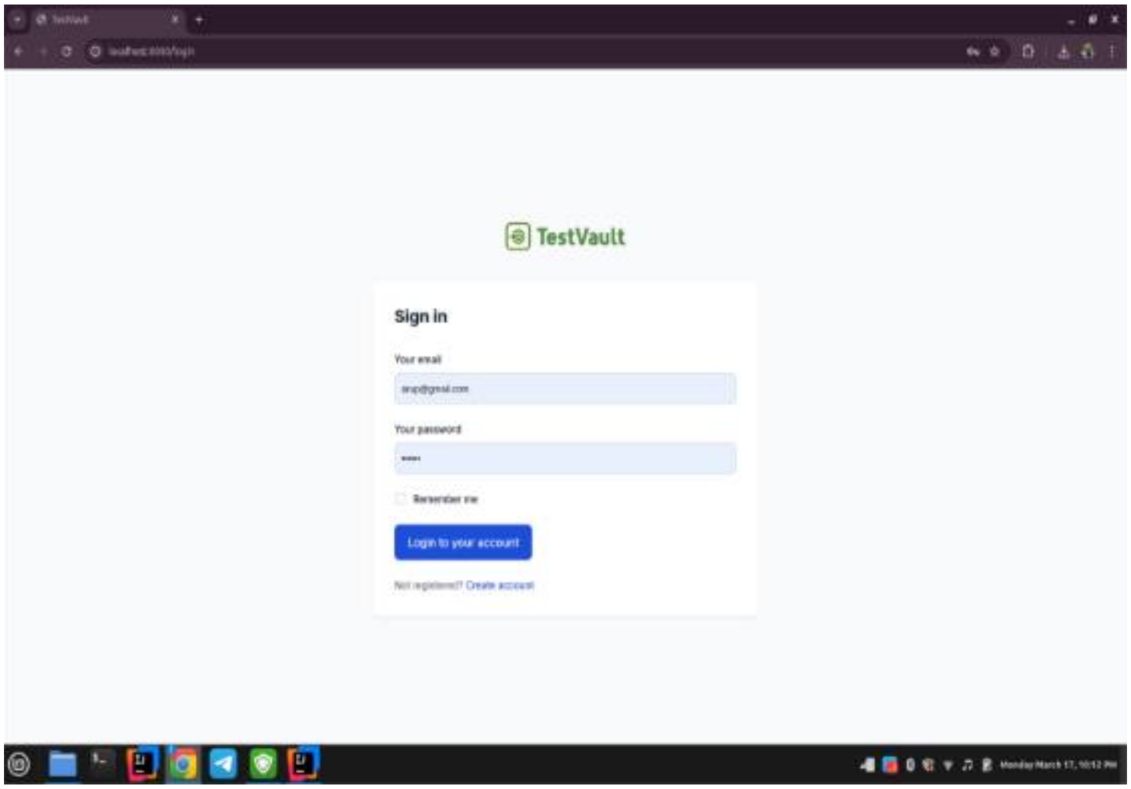
6. Open your browser and navigate to http://localhost:8080

Using the Application

- **Login**: Use your credentials to log in. If you don't have an account, register using the "Sign Up" page.
- **Create a Project**: Navigate to "Create Project", fill out the form, and submit.
- **Manage Modules**: Select a project, then create, edit, or delete modules.
- **Manage Test Cases**: Select a project then select a module, then create, edit, or delete test cases. You can also attach files and add tags.
- **List Views**: Use the "List Projects", "List Modules", and "List Test Cases" pages to view and manage your data.
- **Logout:** By clicking the logout option, the application will take to the logout page.

Screenshot of the Application and functionalities (to help with the navigation)

1. Screenshot of login page



Sign in

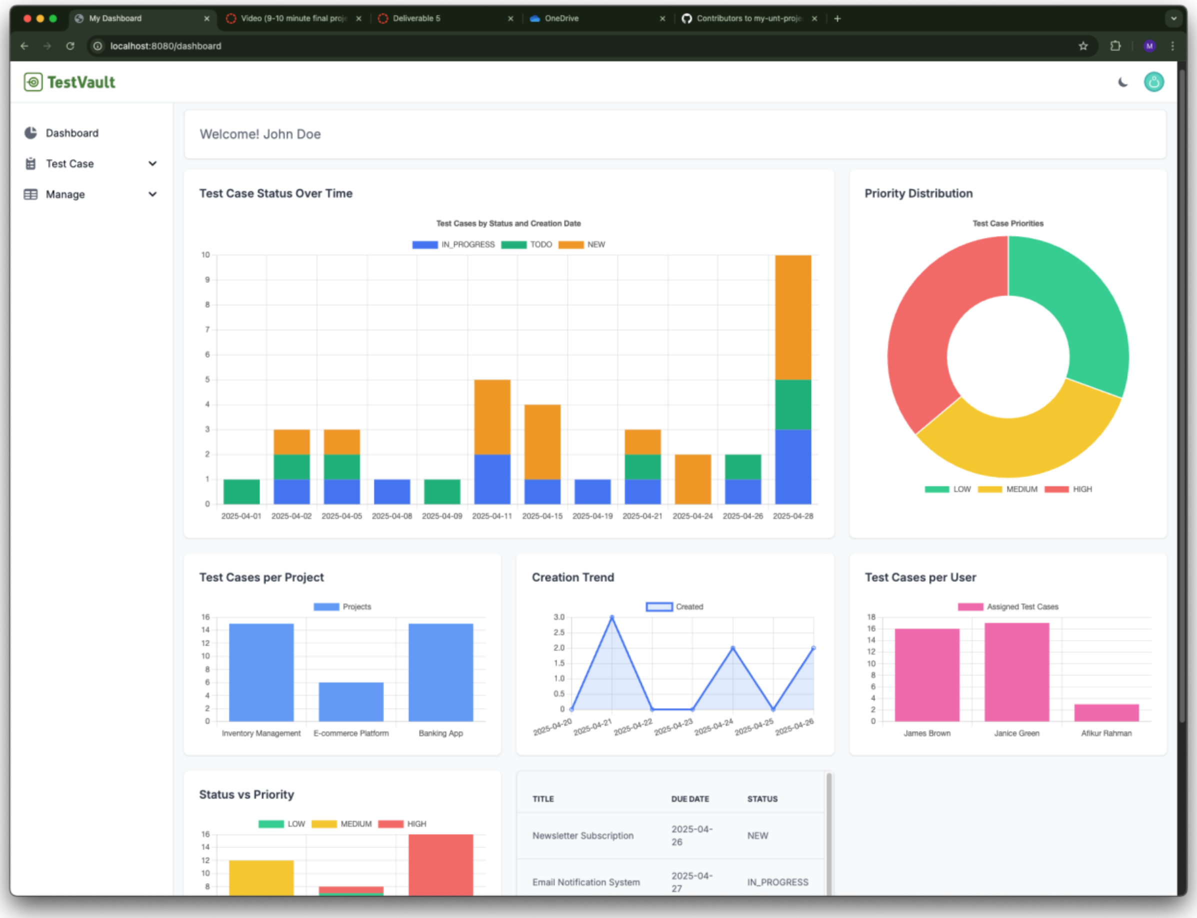
Your email

arup@gmail.com

Your password

•••••

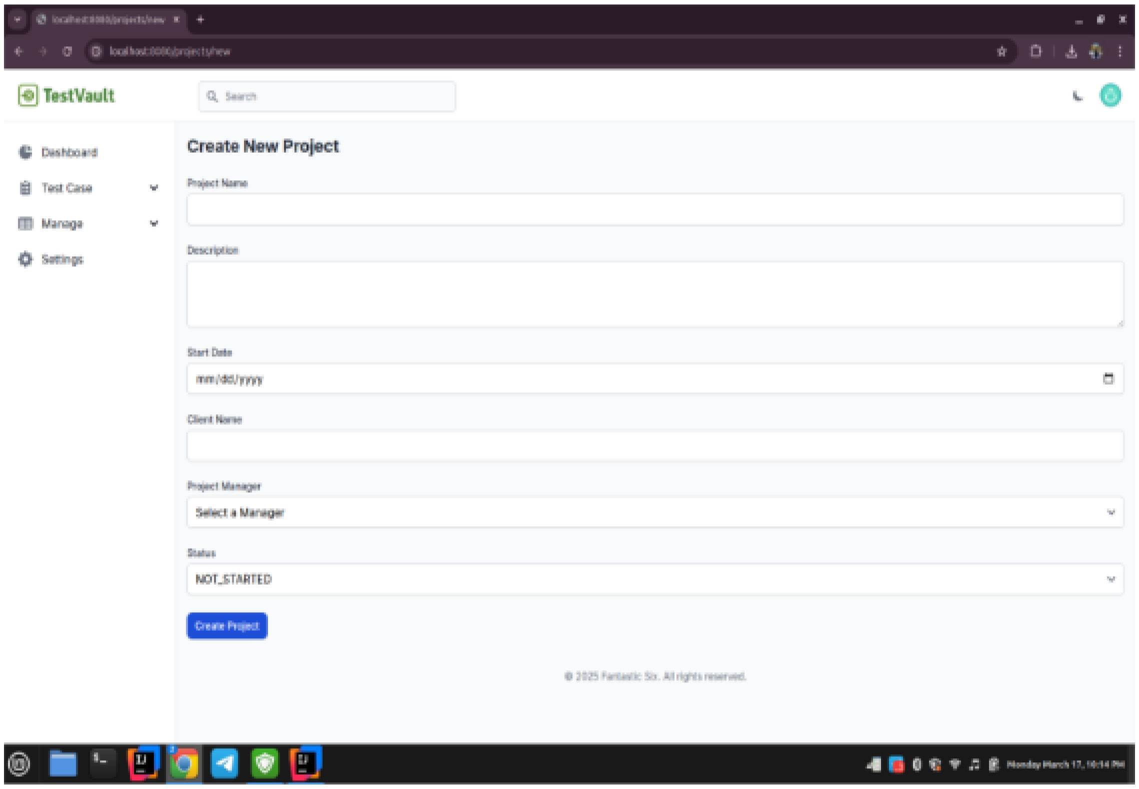
2. Screenshot of the Dashboard page:



Welcome! John Doe

TITLE	DUE DATE	STATUS
Newsletter Subscription	2025-04-26	NEW
Email Notification System	2025-04-27	IN_PROGRESS

3. Screenshot of Project create page:


localhost:8080/projects/new

TestVault

Search

Dashboard
Test Case
Manage
Settings

Create New Project

Project Name

Description

Start Date
mm/dd/yyyy

Client Name

Project Manager
Select a Manager

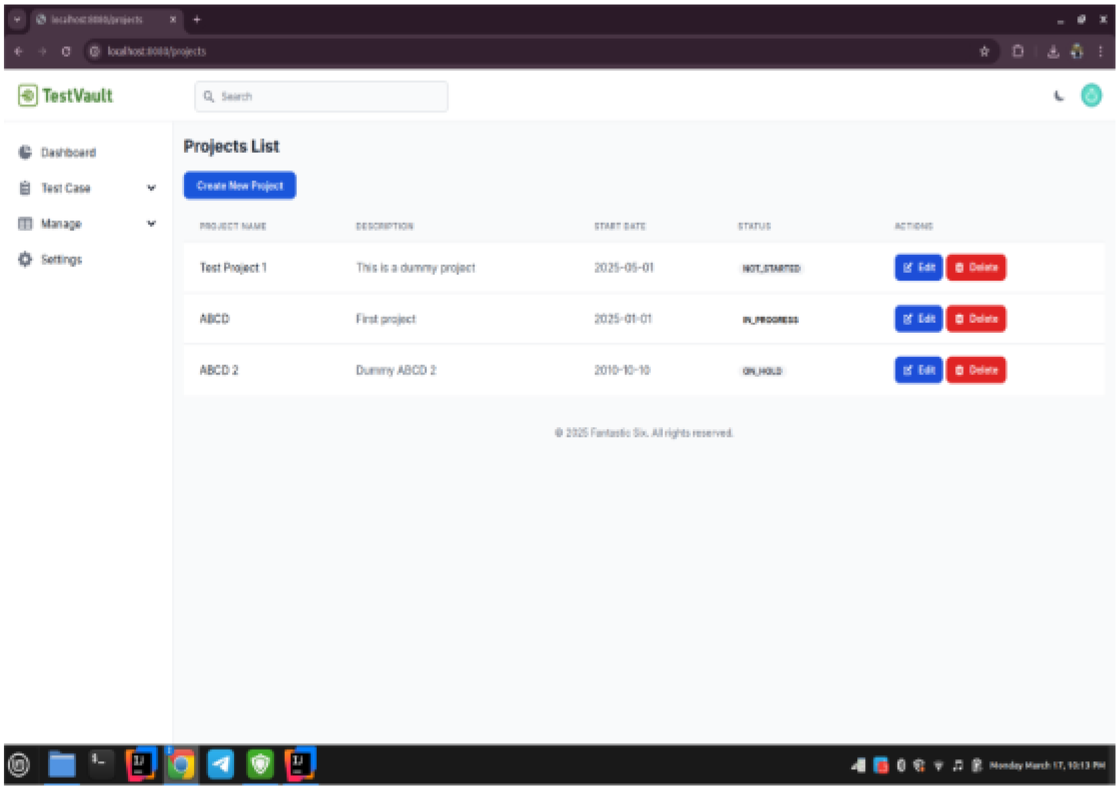
Status
NOT_STARTED

Create Project

© 2025 Fantastic Six. All rights reserved.

Monday March 17, 10:14 PM

4. Screenshot of Project list page:


localhost:8080/projects

TestVault

Search

Dashboard
Test Case
Manage
Settings

Projects List

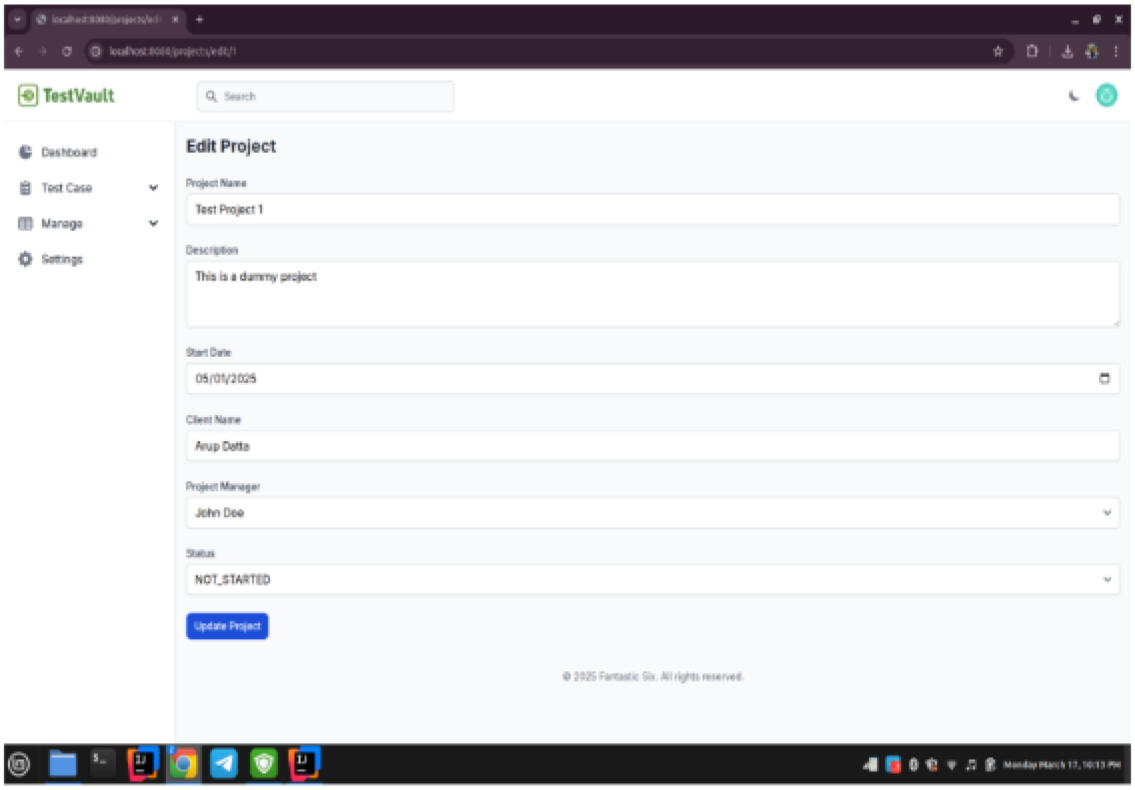
Create New Project

PROJECT NAME	DESCRIPTION	START DATE	STATUS	ACTIONS
Test Project 1	This is a dummy project	2025-05-01	NOT_STARTED	Edit Delete
ABCD	First project	2025-01-01	IN_PROGRESS	Edit Delete
ABCD 2	Dummy ABCD 2	2010-10-10	ON_HOLD	Edit Delete

© 2025 Fantastic Six. All rights reserved.

Monday March 17, 10:13 PM

5. Screenshot of Project edit option:



localhost:8080/projects/edit/1

TestVault

Dashboard
Test Case
Manage
Settings

Edit Project

Project Name: Test Project 1

Description: This is a dummy project

Start Date: 05/01/2025

Client Name: Arup Datta

Project Manager: John Doe

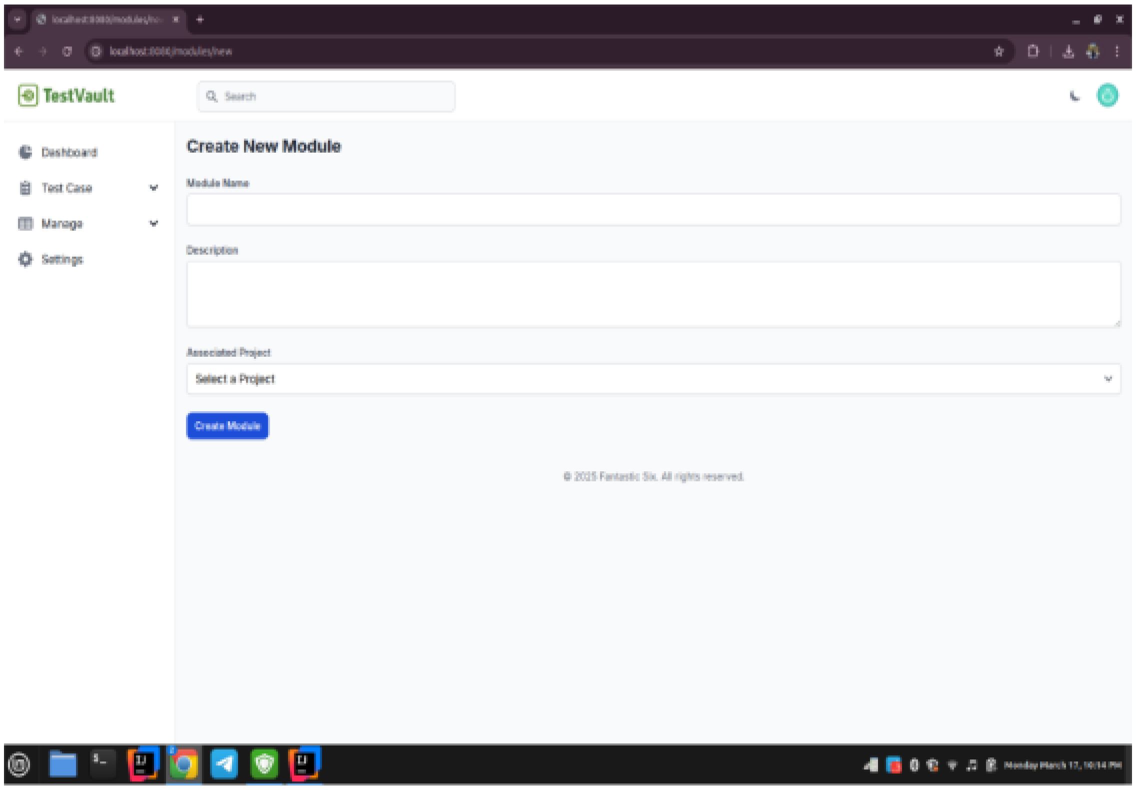
Status: NOT_STARTED

Update Project

© 2025 Fantastic Six. All rights reserved.

Monday March 17, 10:13 PM

6. Screenshot of Module create page:



localhost:8080/modules/new

TestVault

Dashboard
Test Case
Manage
Settings

Create New Module

Module Name:

Description:

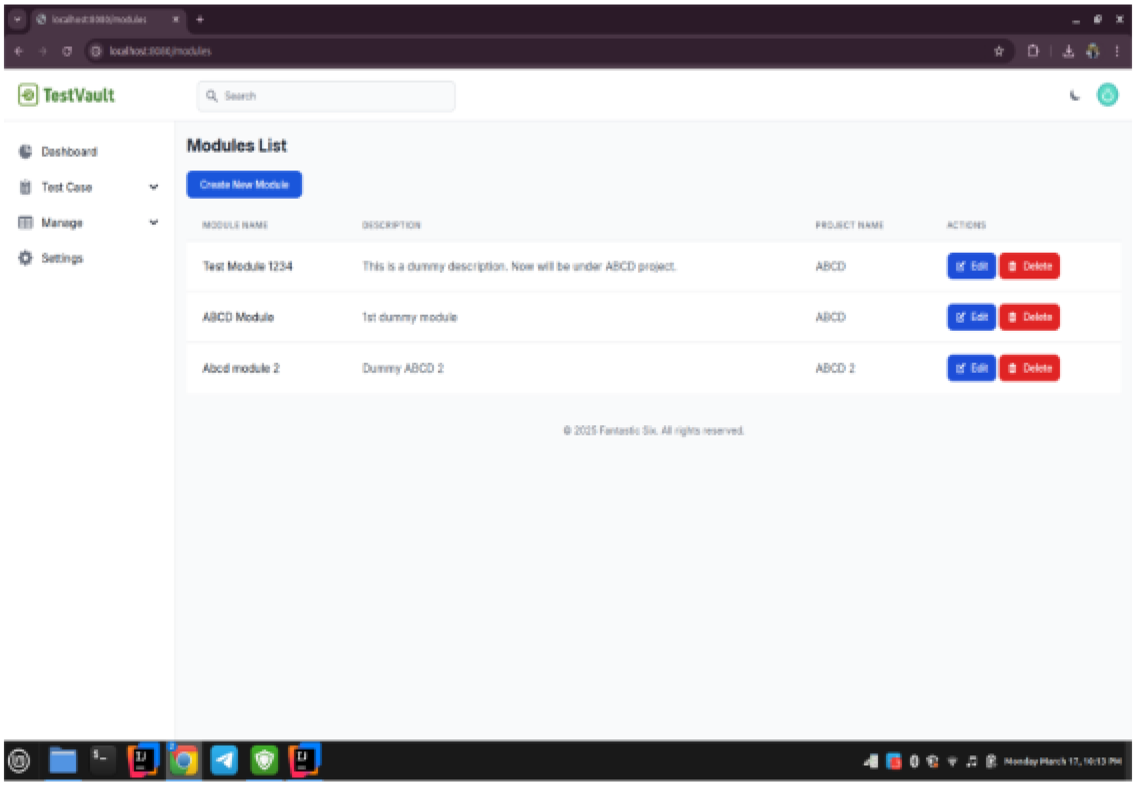
Associated Project: Select a Project

Create Module

© 2025 Fantastic Six. All rights reserved.

Monday March 17, 10:14 PM

7. Screenshot of Module list page:



Modules List

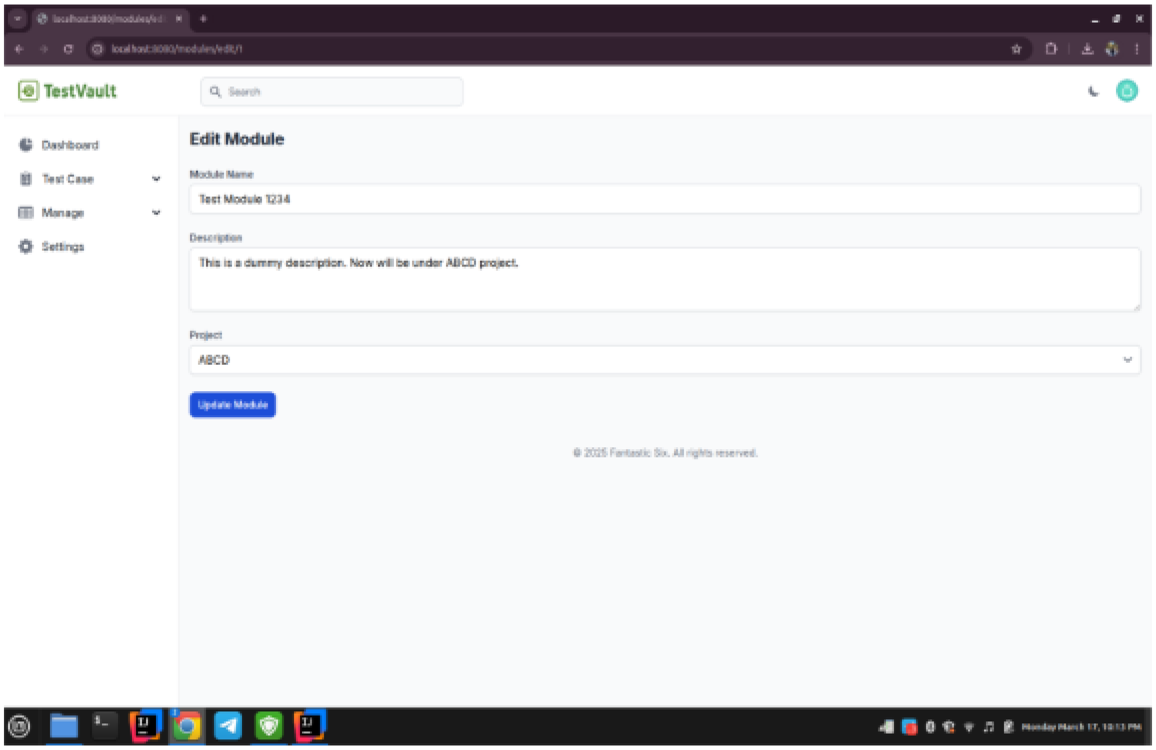
Create New Module

MODULE NAME	DESCRIPTION	PROJECT NAME	ACTIONS
Test Module 1234	This is a dummy description. Now will be under ABCD project.	ABCD	Edit Delete
ABCD Module	1st dummy module	ABCD	Edit Delete
Abcd module 2	Dummy ABCD 2	ABCD 2	Edit Delete

© 2025 Fantastic Six. All rights reserved.

Monday March 17, 10:13 PM

8. Screenshot of Module edit option:



Edit Module

Module Name: Test Module 1234

Description: This is a dummy description. Now will be under ABCD project.

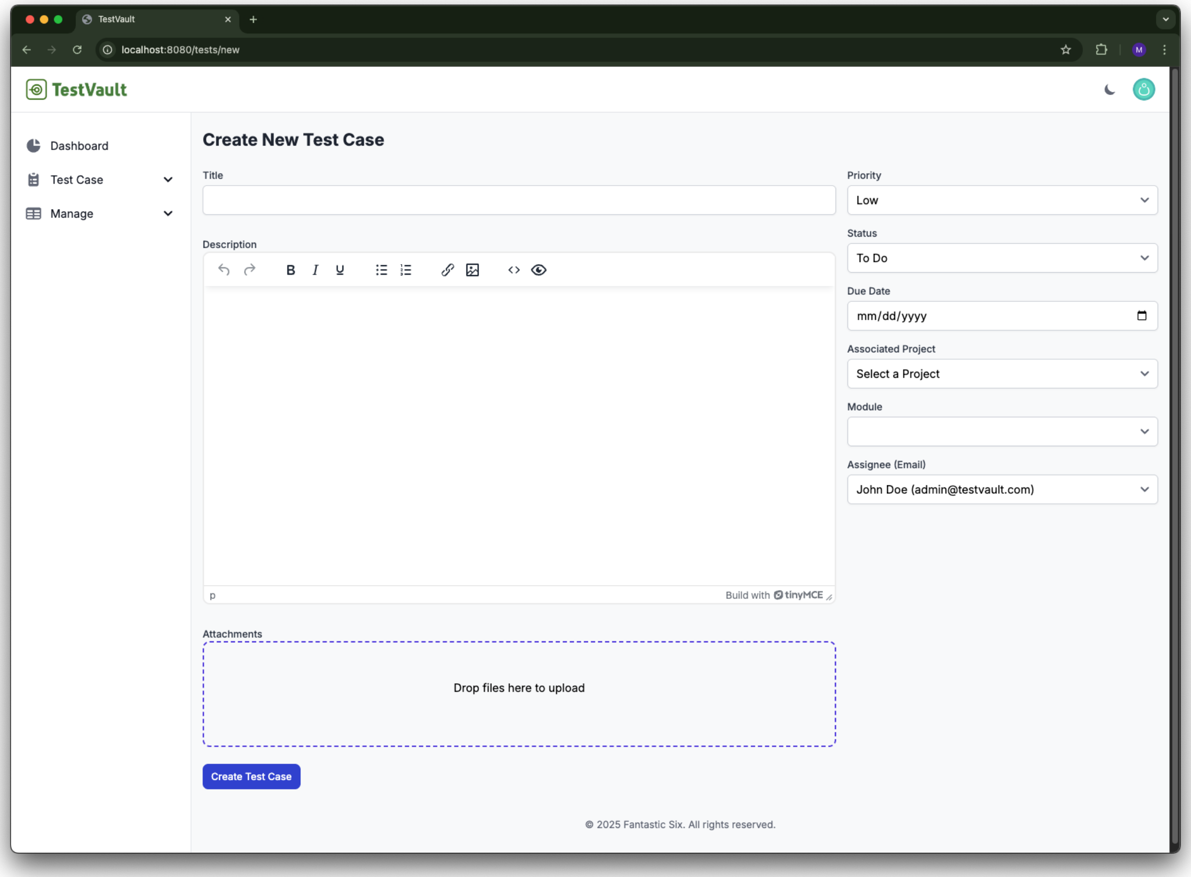
Project: ABCD

Update Module

© 2025 Fantastic Six. All rights reserved.

Monday March 17, 10:13 PM

9. Screenshot of Test case create page:



Create New Test Case

Title

Description

Build with tinyMCE

Attachments

Drop files here to upload

Priority: Low

Status: To Do

Due Date: mm/dd/yyyy

Associated Project: Select a Project

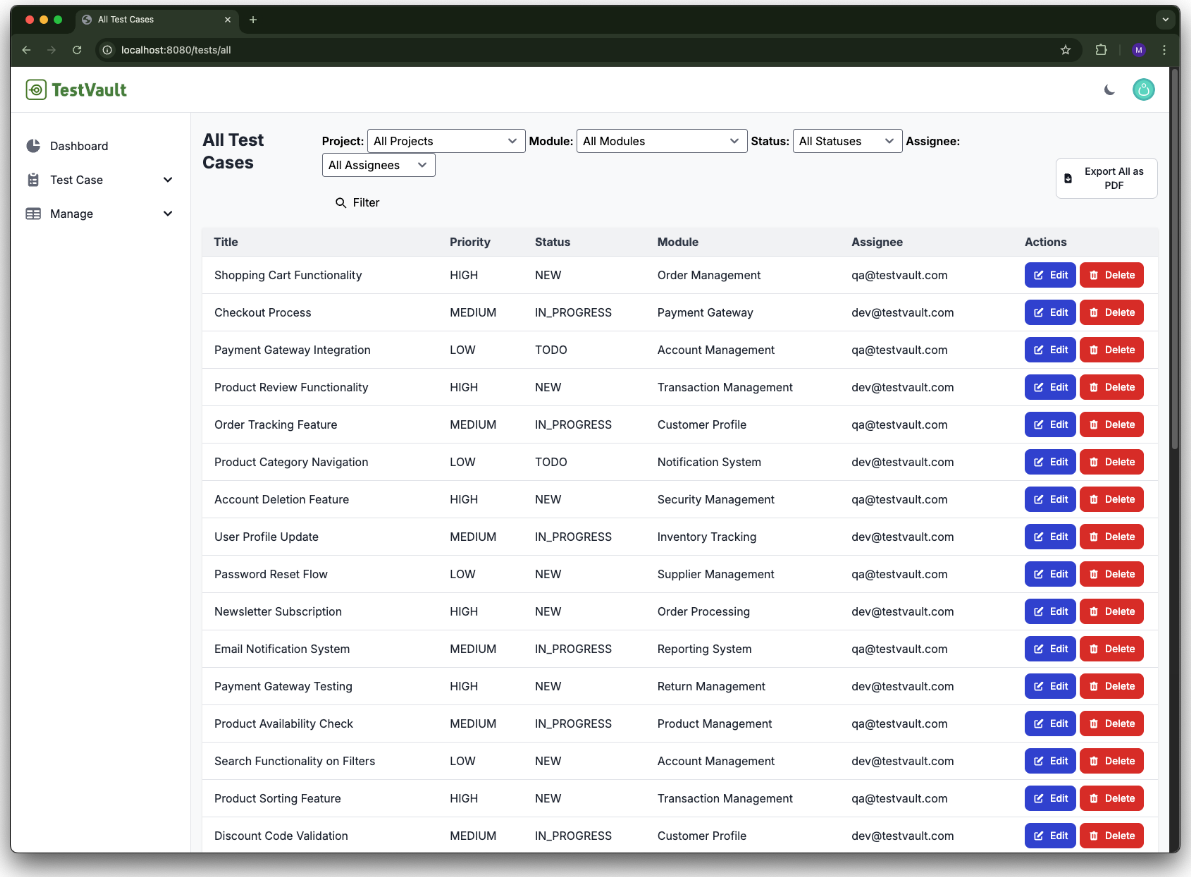
Module:

Assignee (Email): John Doe (admin@testvault.com)

Create Test Case

© 2025 Fantastic Six. All rights reserved.

10. Screenshot showing List all test cases page:



All Test Cases

localhost:8080/tests/all

TestVault

Dashboard

Test Case

Manage

All Test Cases

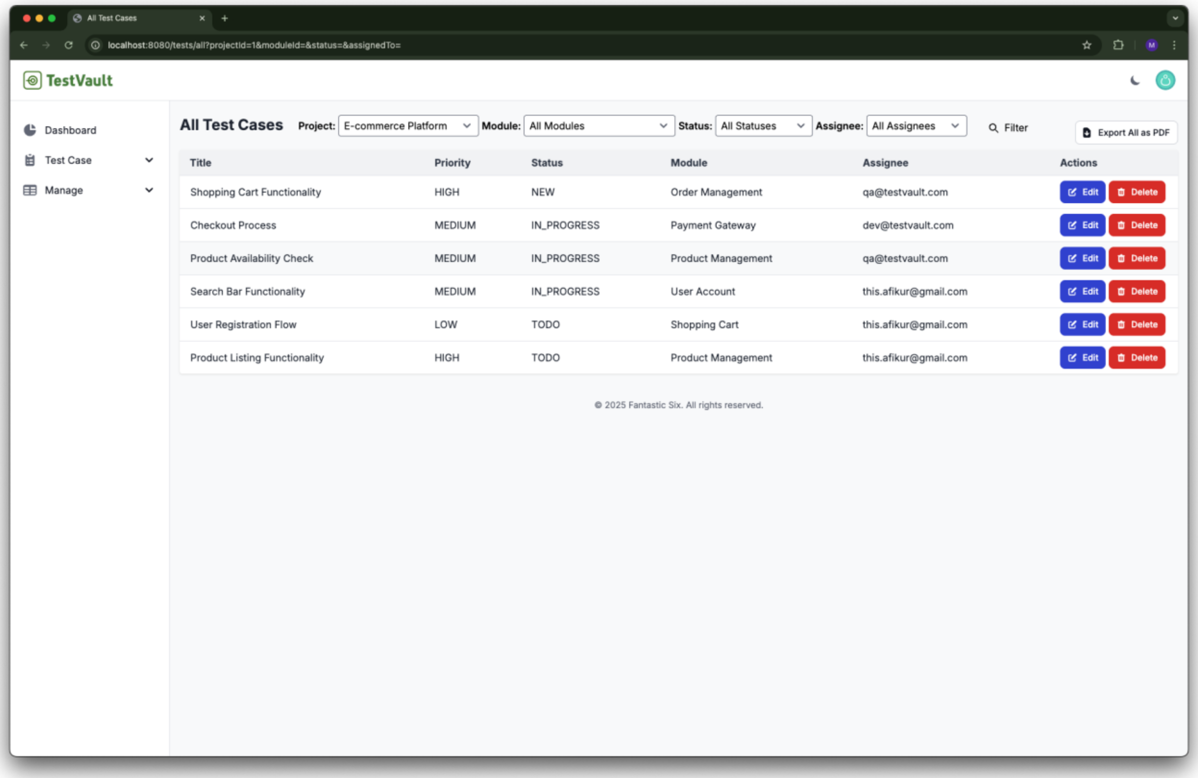
Project: All Projects Module: All Modules Status: All Statuses Assignee: All Assignees

Filter

Export All as PDF

Title	Priority	Status	Module	Assignee	Actions
Shopping Cart Functionality	HIGH	NEW	Order Management	qa@testvault.com	Edit Delete
Checkout Process	MEDIUM	IN_PROGRESS	Payment Gateway	dev@testvault.com	Edit Delete
Payment Gateway Integration	LOW	TODO	Account Management	qa@testvault.com	Edit Delete
Product Review Functionality	HIGH	NEW	Transaction Management	dev@testvault.com	Edit Delete
Order Tracking Feature	MEDIUM	IN_PROGRESS	Customer Profile	qa@testvault.com	Edit Delete
Product Category Navigation	LOW	TODO	Notification System	dev@testvault.com	Edit Delete
Account Deletion Feature	HIGH	NEW	Security Management	qa@testvault.com	Edit Delete
User Profile Update	MEDIUM	IN_PROGRESS	Inventory Tracking	dev@testvault.com	Edit Delete
Password Reset Flow	LOW	NEW	Supplier Management	qa@testvault.com	Edit Delete
Newsletter Subscription	HIGH	NEW	Order Processing	dev@testvault.com	Edit Delete
Email Notification System	MEDIUM	IN_PROGRESS	Reporting System	qa@testvault.com	Edit Delete
Payment Gateway Testing	HIGH	NEW	Return Management	dev@testvault.com	Edit Delete
Product Availability Check	MEDIUM	IN_PROGRESS	Product Management	qa@testvault.com	Edit Delete
Search Functionality on Filters	LOW	NEW	Account Management	dev@testvault.com	Edit Delete
Product Sorting Feature	HIGH	NEW	Transaction Management	qa@testvault.com	Edit Delete
Discount Code Validation	MEDIUM	IN_PROGRESS	Customer Profile	dev@testvault.com	Edit Delete

11. Screenshot of Filter test case page

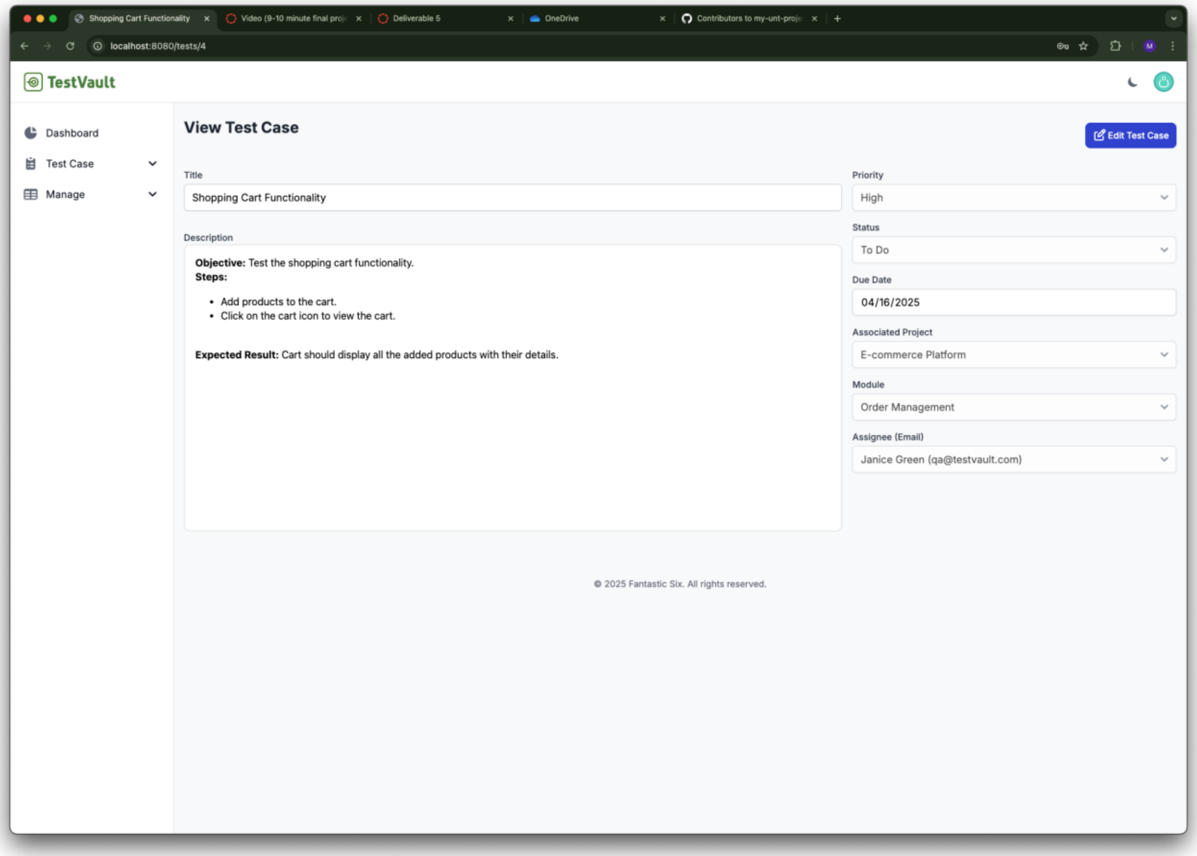

All Test Cases
localhost:8080/tests/all?projectId=1&moduleId=&status=&assignedTo=
TestVault
Dashboard
Test Case
Manage

All Test Cases **Project:** E-commerce Platform **Module:** All Modules **Status:** All Statuses **Assignee:** All Assignees Filter Export All as PDF

Title	Priority	Status	Module	Assignee	Actions
Shopping Cart Functionality	HIGH	NEW	Order Management	qa@testvault.com	Edit Delete
Checkout Process	MEDIUM	IN_PROGRESS	Payment Gateway	dev@testvault.com	Edit Delete
Product Availability Check	MEDIUM	IN_PROGRESS	Product Management	qa@testvault.com	Edit Delete
Search Bar Functionality	MEDIUM	IN_PROGRESS	User Account	this.afikur@gmail.com	Edit Delete
User Registration Flow	LOW	TODO	Shopping Cart	this.afikur@gmail.com	Edit Delete
Product Listing Functionality	HIGH	TODO	Product Management	this.afikur@gmail.com	Edit Delete

© 2025 Fantastic Six. All rights reserved.

12. View Test Case



View Test Case

Title: Shopping Cart Functionality

Description

Objective: Test the shopping cart functionality.
Steps:

- Add products to the cart.
- Click on the cart icon to view the cart.

Expected Result: Cart should display all the added products with their details.

Priority: High

Status: To Do

Due Date: 04/16/2025

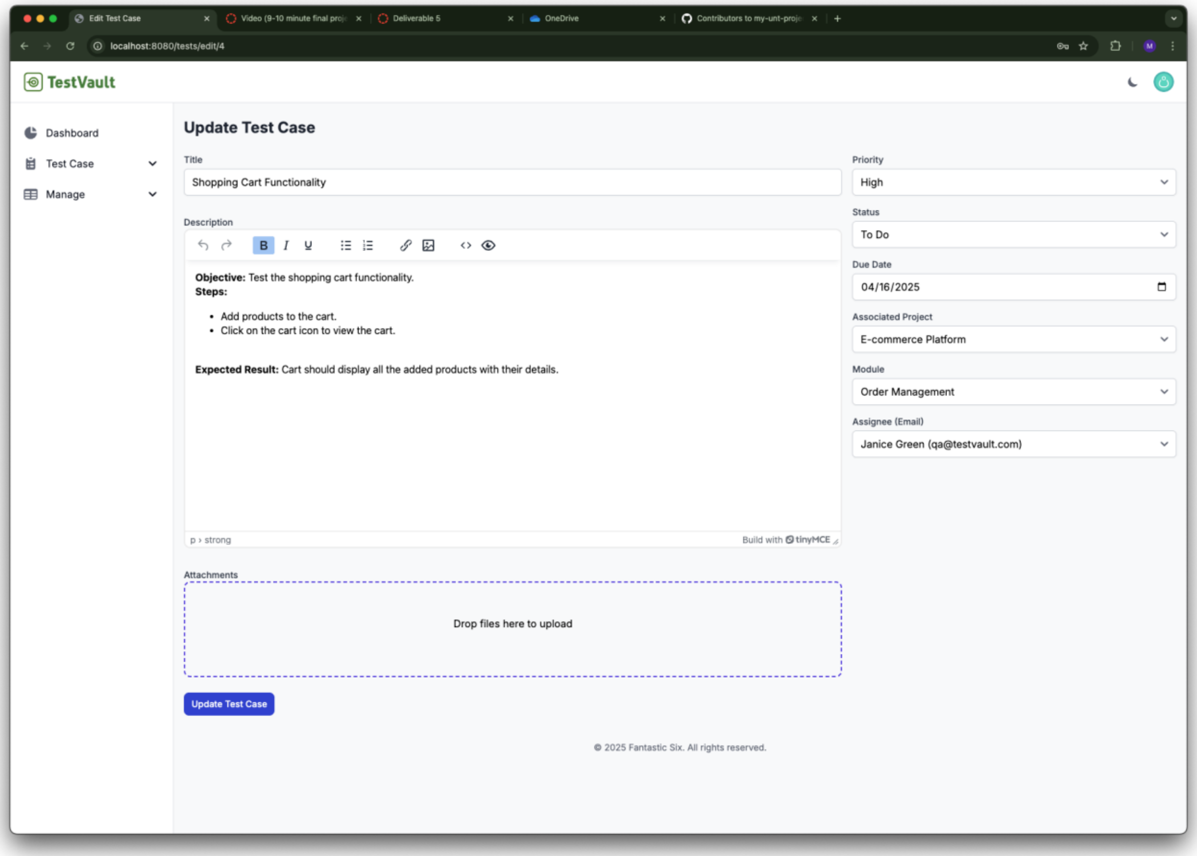
Associated Project: E-commerce Platform

Module: Order Management

Assignee (Email): Janice Green (qa@testvault.com)

© 2025 Fantastic Six. All rights reserved.

13. Edit Test Cases



Update Test Case

Title: Shopping Cart Functionality

Priority: High

Status: To Do

Due Date: 04/16/2025

Associated Project: E-commerce Platform

Module: Order Management

Assignee (Email): Janice Green (qa@testvault.com)

Description

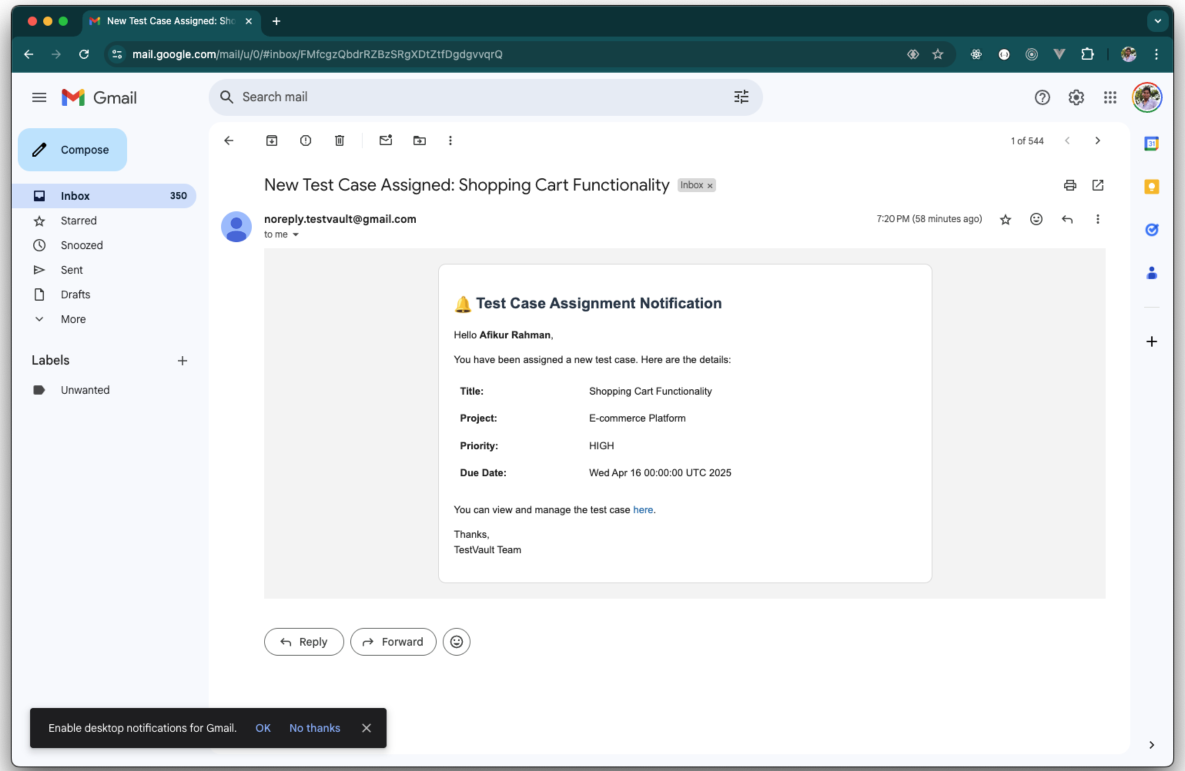
Objective: Test the shopping cart functionality.
Steps:

- Add products to the cart.
- Click on the cart icon to view the cart.

Expected Result: Cart should display all the added products with their details.

© 2025 Fantastic Six. All rights reserved.

14. Email Notification Screenshots


New Test Case Assigned: Sho
mail.google.com/mail/u/0/#inbox/FMfcgzQbdrRZBzSRgXDtZtfDgdgvvqrQ
Gmail
Search mail
Compose
Inbox 350
Starred
Snoozed
Sent
Drafts
More
Labels
Unwanted
1 of 544
Inbox
7:20 PM (58 minutes ago)
Reply
Forward
Enable desktop notifications for Gmail. OK No thanks

New Test Case Assigned: Shopping Cart Functionality

noreply.testvault@gmail.com
to me

🔔 Test Case Assignment Notification

Hello **Afikur Rahman**,

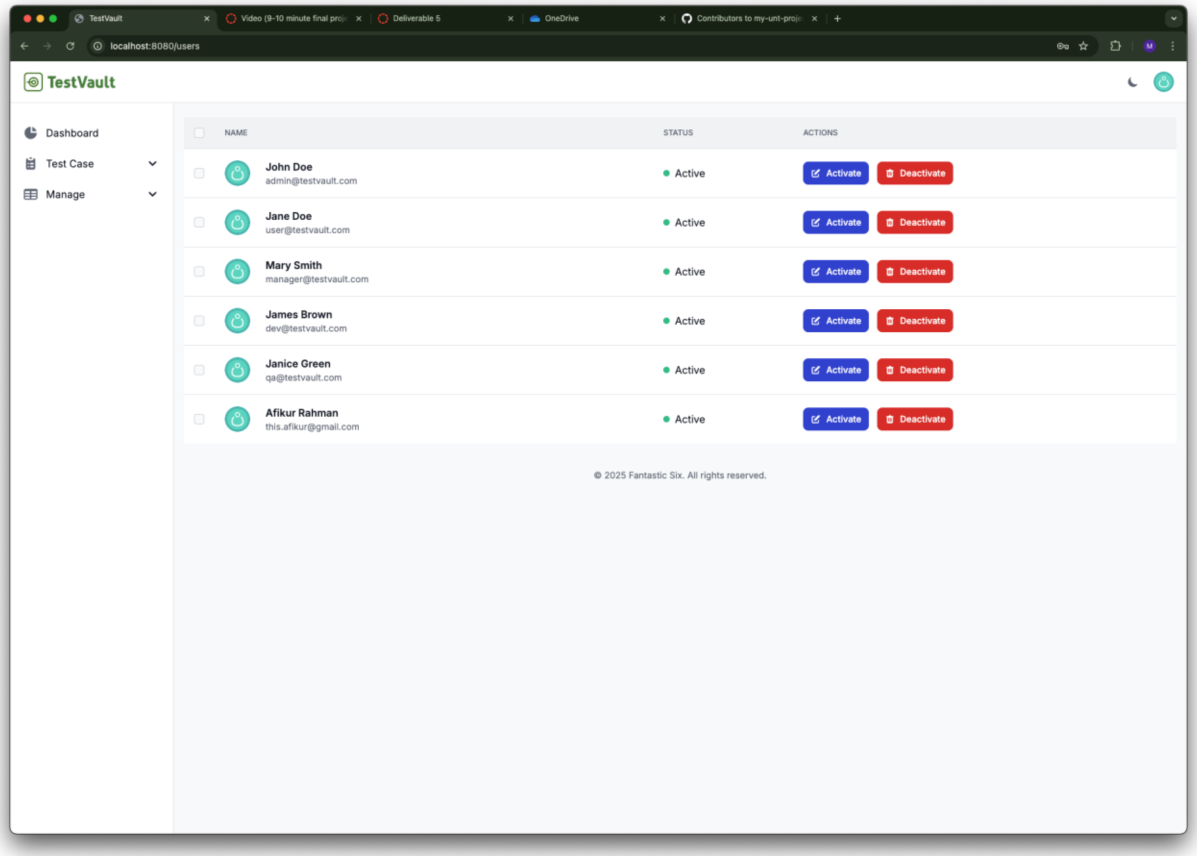
You have been assigned a new test case. Here are the details:

Title:	Shopping Cart Functionality
Project:	E-commerce Platform
Priority:	HIGH
Due Date:	Wed Apr 16 00:00:00 UTC 2025

You can view and manage the test case here.

Thanks,
TestVault Team

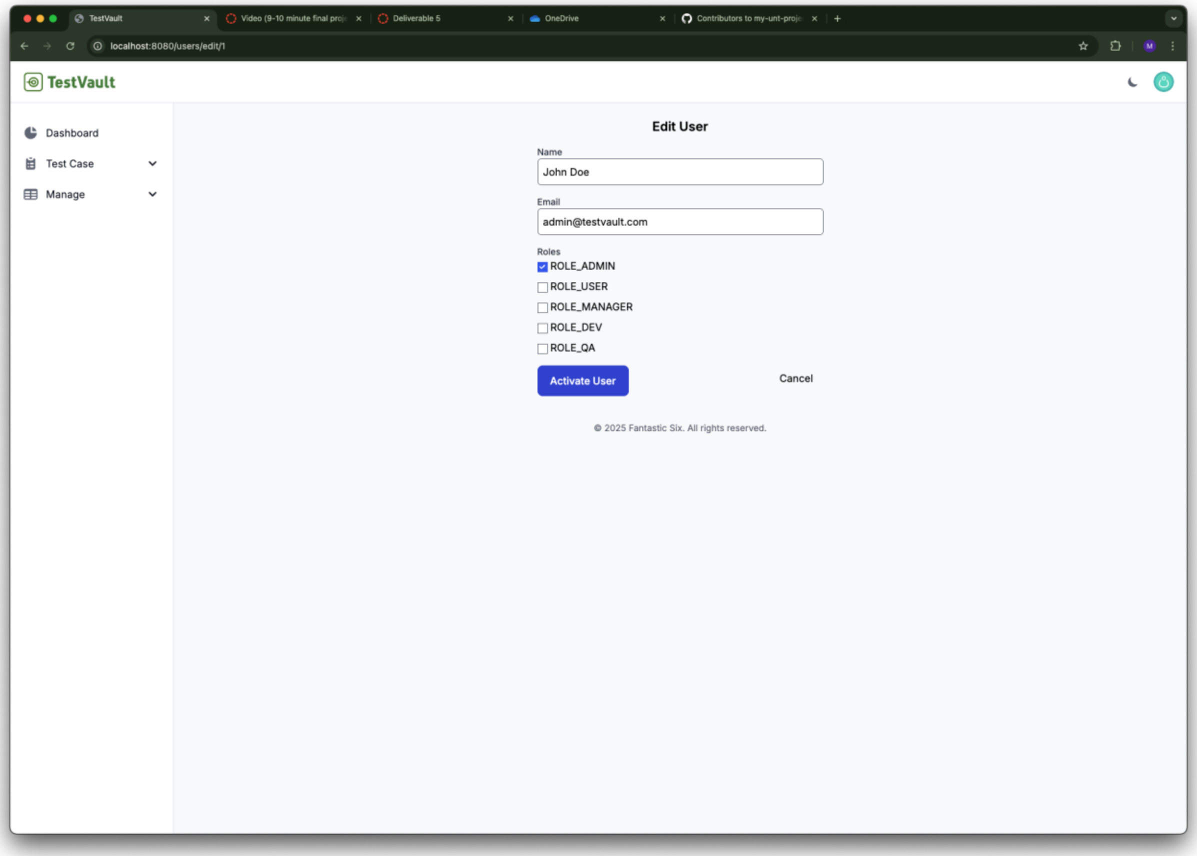
15. Users List Page



NAME	STATUS	ACTIONS
John Doe admin@testvault.com	Active	Activate Deactivate
Jane Doe user@testvault.com	Active	Activate Deactivate
Mary Smith manager@testvault.com	Active	Activate Deactivate
James Brown dev@testvault.com	Active	Activate Deactivate
Janice Green qa@testvault.com	Active	Activate Deactivate
Afikur Rahman this.afikur@gmail.com	Active	Activate Deactivate

© 2025 Fantastic Six. All rights reserved.

16. User Activation Page


TestVault
Video (9-10 minute final proj
Deliverable 5
OneDrive
Contributors to my-unt-proje
localhost:8080/users/edit/1
TestVault
Dashboard
Test Case
Manage

Edit User

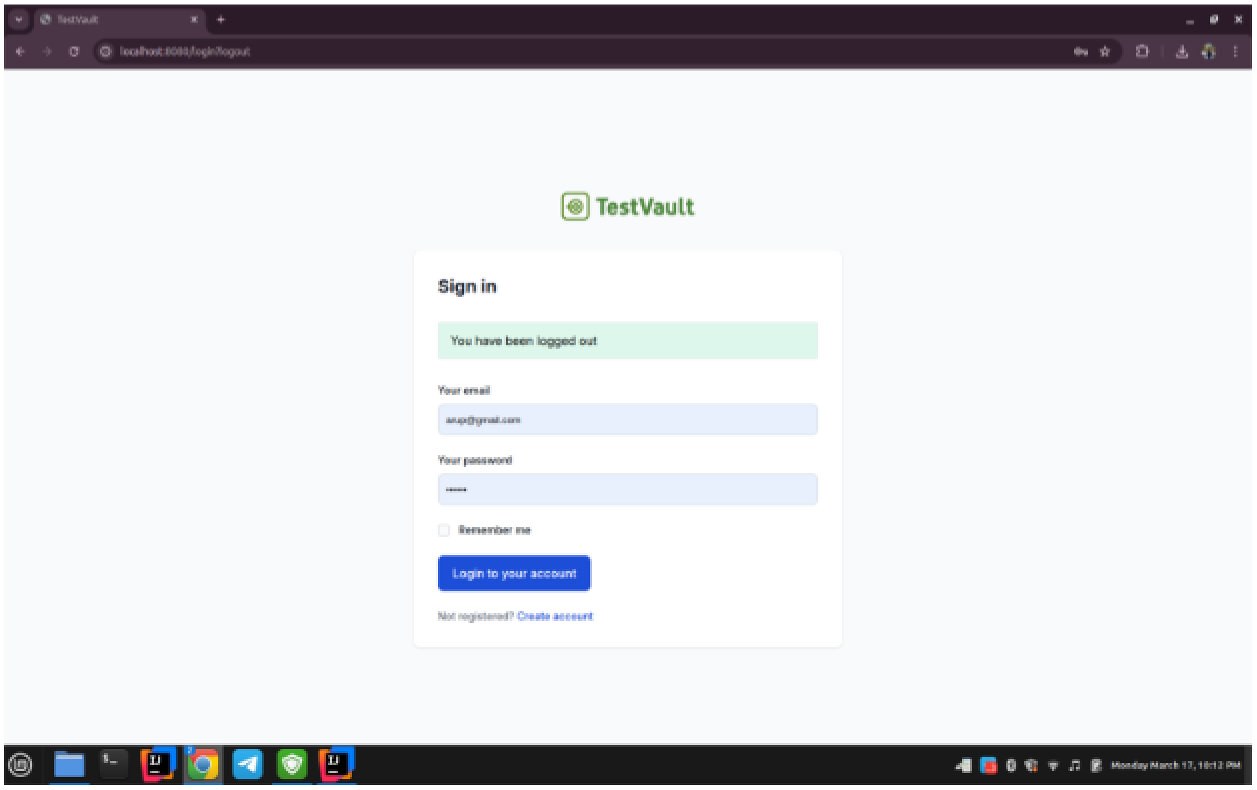
Name: John Doe

Email: admin@testvault.com

Roles
- [x] ROLE_ADMIN
- [ ] ROLE_USER
- [ ] ROLE_MANAGER
- [ ] ROLE_DEV
- [ ] ROLE_QA

© 2025 Fantastic Six. All rights reserved.

17. Screenshot of Logout page:



TestVault

Sign in

You have been logged out

Your email

arup@gmail.com

Your password

••••••

Remember me

Login to your account

Not registered? Create account

e. Instructions to Compile/Run Program and Test Cases

Compile and Run the Program

1. Ensure Java 17 is installed.
2. Install docker desktop from https://www.docker.com/products/docker-desktop/
3. Clone the repository from the main branch.
4. Navigate to the project directory.
5. Start MySQL using Docker:
   docker-compose up -d
6. Compile and run the application:
   ./mvnw spring-boot:run
7. Access the application at http://localhost:8080.

Run Test Cases

1. Ensure the application is running (see above).

2. Run unit tests using Maven: After opening the project from Intellij IDE, you can run them directly from by clicking on the Test class. To run the tests from the terminal, run

   ```
   ./mvnw spring-boot:test
   ```

   Or simply run "**mvn test"** from the terminal.

f. Feedback from Code Inspection Workshop

During the code inspection workshop, the following feedback was received and either accepted or rejected:

Accepted Feedback:

- **Code Comments:** Agreed to enhance inline documentation.
- **Class Diagram:** Clarify user-project relationships and role specifications.

Rejected/Deferred Feedback:

Multiple Project Managers: The one-to-one relationship remains unchanged due to scope constraints.

g. Reflection

Accomplishments

- Successfully completed all Phase 3 requirements:
  - Designed and implemented UI templates for test case management using Thymeleaf.
  - Developed full CRUD operations for test cases with backend integration.
  - Enabled file attachments and implemented tagging with filtering functionality.
  - Integrated robust role-based access control using Spring Security.
  - Seeded the system with initial test cases and user roles for testing/demo.
  - Implemented test case assignments to team members with email notifications.

    - Conducted comprehensive integration and performance testing.
- Ensured the application is secure, fully functional, and user-friendly across all modules.

What Went Well

- Spring Boot and Thymeleaf integration enabled rapid and consistent UI development.
- Dockerized Postgres ensured stable and portable database environments.
- Early unit and integration testing accelerated bug detection and resolution.
- Tagging, filtering, and notifications added real-world usability to test case management.

Areas for Improvement

- UI/UX can be enhanced further to improve user interactions and responsiveness.
- Backend code structure can be modularized further for better scalability and maintainability.

h. Member Contribution Table

Member Name	Contribution Description	Overall Contribution (%)	Note (if applicable)
Afikur Rahman Khan	Added feature for project-based module dropdown in the create test case page. Added role-based access control and added JUnit tests for the controller and service layer. In the report, I added the test cases (unit test) summary table. Created the class diagram. Reviewed the report.	20%	
Arup Datta	Helped in preparing the deliverable 5 document. Prepared sequence diagrams for Dashboard Controller. Prepared the use case diagram. Tested the application from a user perspective. Reviewed the report.	20%	
Jarin Tasnim Ishika	Document preparation. Built the UI with Thymeleaf and set up full backend CRUD support for managing test cases.	20%	

Nhi Lee		0%	
Vaishnavi Sabna	Responsible for the UML diagrams and helped in the document.	20%	
Rachelle Blosser	Initiated documentation	20%	