Client Management System Project

Developer: Syed Arfaat (sarfaat@cisco.com)

Project Overview

The objective of this project is to design and develop a comprehensive Client Management System that efficiently manages client data, projects, tasks, meetings, and portfolios. The system is intended to streamline operations and improve user interaction within the client management portal. The project will be executed using the Agile methodology, divided into two iterative sprints, allowing for continuous feedback and improvements.

Project Requirements

The system should enable users to:

- **User Registration and Authentication:** Allow new users to register and existing users to log in securely to access the client management portal.
- **Portfolio Management:** Enable users to create and manage portfolios to organize their client projects effectively.
- **Task Management:** Provide functionality for users to add and track project tasks, ensuring that all project activities are monitored.
- **Meeting Scheduling:** Facilitate the scheduling of meetings with clients and employees, integrating calendar functionalities for better planning.

Sprint 1: Analysis and Requirements (2 weeks)

Goals:

- Gather and define detailed project requirements through collaboration with stakeholders.
- Identify and prioritize user stories that capture the core functionalities needed.
- Create acceptance criteria for each user story to ensure that requirements are met during development.

User Stories and Acceptance Criteria:

1. **User Registration**
    - **Title:** Register
    - **Description:** As a new user, I want to register an account to access the client management portal.
    - **Acceptance Criteria:**
        - The system must validate the provided username, email, and password.
        - Successful registration should redirect the user to the login page.

        - An appropriate success message should be displayed to the user.
2. **User Login**
    - **Title:** Login
    - **Description:** As an existing user, I want to log in to manage clients.
    - **Acceptance Criteria:**
        - The system must validate the email and password.
        - Successful login should redirect the user to the portfolios page.
        - A success message should confirm the login.
3. **Portfolios Management**
    - **Title:** Portfolios
    - **Description:** As a user, I want to add portfolios to organize client projects.
    - **Acceptance Criteria:**
        - Users must be able to input a portfolio name and description.
        - The portfolio must be saved in the database and displayed in the portfolio list.
4. **Project Tasks**
    - **Title:** Project Tasks
    - **Description:** As a user, I want to add project tasks to manage project progress.
    - **Acceptance Criteria:**
        - Users must be able to save a task title and its current status.
        - The task must be displayed in the project tasks list for tracking.
5. **Schedule Meetings**
    - **Title:** Schedule Meetings
    - **Description:** As a user, I want to schedule meetings with clients and employees.
    - **Acceptance Criteria:**
        - Users must be able to save meeting details including title, date, and time.
        - The scheduled meeting must be displayed in the meetings list.

Sprint 2: Development and Testing (2 weeks)

Goals:

- Develop the client management system according to the defined requirements.
- Conduct rigorous testing and validation to ensure functionality and reliability.

Development Tasks:

Frontend Development:

- Utilize **Angular**, **HTML**, and **CSS**,to create an intuitive user interface.
- Implement user registration, login, portfolio management, project tasks, and meeting scheduling features. Below are some essential Angular commands and configurations used:

Tools and Technologies

- **Frontend:** Angular, HTML, CSS, Bootstrap.
- **Backend:** Express, MySQL.
- **Testing:** Java, Selenium, Cucumber.
- **Version Control:** Git, GitHub.
- **IDE:** Visual Studio Code.

How To Run The Project

Prerequisites:

To successfully run the Client Management System, ensure that the following software is installed:

- **Node.js**: Download from Node.js.
- **Angular CLI**: Install globally using the command `npm install -g @angular/cli`.
- **MySQL**: Download from MySQL.

```
# Navigate to the frontend directory
cd client-management-system/client

# Install necessary dependencies
npm install

# Create a new component for portfolios
ng g c portfolios

# Create a new service for project tasks
ng g s projectTasks

# Serve the application and open it in the browser
ng serve -o
```

Backend Development:

- Use **Express** and **MySQL** to set up the server and manage database interactions.
- Develop RESTful APIs to handle user registration, login, portfolio management, project tasks, and meeting scheduling. Key steps include:

```
# Navigate to the backend directory
cd client-management-system/server

# Install necessary dependencies
npm install

# Set up MySQL database and configure connection
# Create database and update the credentials in
`server/config/db.config.js`

# Run the server
node server.js
```

Access The Application

- Open a web browser and navigate to http://localhost:4200.
- Register or log in to access the client management portal.

Deliverables

- A fully functional client management system that meets all specified requirements.
- A thoroughly tested and validated system ensuring reliability.
- Comprehensive documentation detailing system usage and maintenance for future reference

Testing Framework Overview

For the Client Management System, a robust testing framework was implemented using **Eclipse IDE**, **JUnit**, and **Cucumber** to ensure that the application functions as intended and meets the specified requirements. This section outlines how these tools were utilized in the project.

Eclipse IDE

Eclipse is a widely used integrated development environment (IDE) that supports various programming languages and is particularly popular for Java development. In this project, Eclipse was used to:

- Write and manage code efficiently.
- Create and run test cases using the JUnit framework.
- Organize feature files and step definitions for behavior-driven development (BDD) using Cucumber.

JUnit Testing

JUnit is a testing framework for Java that allows developers to write and run repeatable tests. It is instrumental in ensuring that the individual components of the application function correctly. Key aspects of JUnit testing in this project included:

1. **Unit Tests**: Tests were created for individual components of the application, such as user registration, login, portfolio management, and project tasks. Each unit test focuses on a specific functionality to validate its correctness.
2. **Annotations**: JUnit uses annotations to define test methods and setup configurations. Common annotations used include:
    - `@Test`: Indicates that a method is a test method.
    - `@Before`: Sets up the conditions before each test method runs.
    - `@After`: Cleans up after each test method completes.
3. **Assertions**: JUnit provides various assertion methods to verify the expected outcomes, such as:
    - `assertEquals()`: Checks if two values are equal.
    - `assertTrue()`: Checks if a condition is true.
    - `assertNotNull()`: Verifies that an object is not null.

Cucumber and BDD

Cucumber is a tool that supports behavior-driven development (BDD), allowing for the creation of tests based on user stories. This approach ensures that the application is developed with the user's perspective in mind.

Feature Files: Feature files are written in a natural language format (Gherkin) that describes the functionality of the application in terms of scenarios. Each feature file corresponds to a particular functionality. For example, a feature file for user registration might look like this:

```
Feature: User Registration

Scenario: Successful registration
  Given I am on the registration page
  When I enter valid user details
  Then I should see a success message
  And I should be redirected to the login page
```

Step Definitions: Step definitions are Java methods that implement the steps defined in the feature files. Each step in a scenario corresponds to a method that performs the required actions or checks. For example, a step definition for the above scenario might look like this:

```
@Given("I am on the registration page")
public void i_am_on_the_registration_page() {
    // Code to navigate to the registration page
}

@When("I enter valid user details")
public void i_enter_valid_user_details() {
    // Code to fill in registration form
}

@Then("I should see a success message")
public void i_should_see_a_success_message() {
    // Code to check for success message
}
```

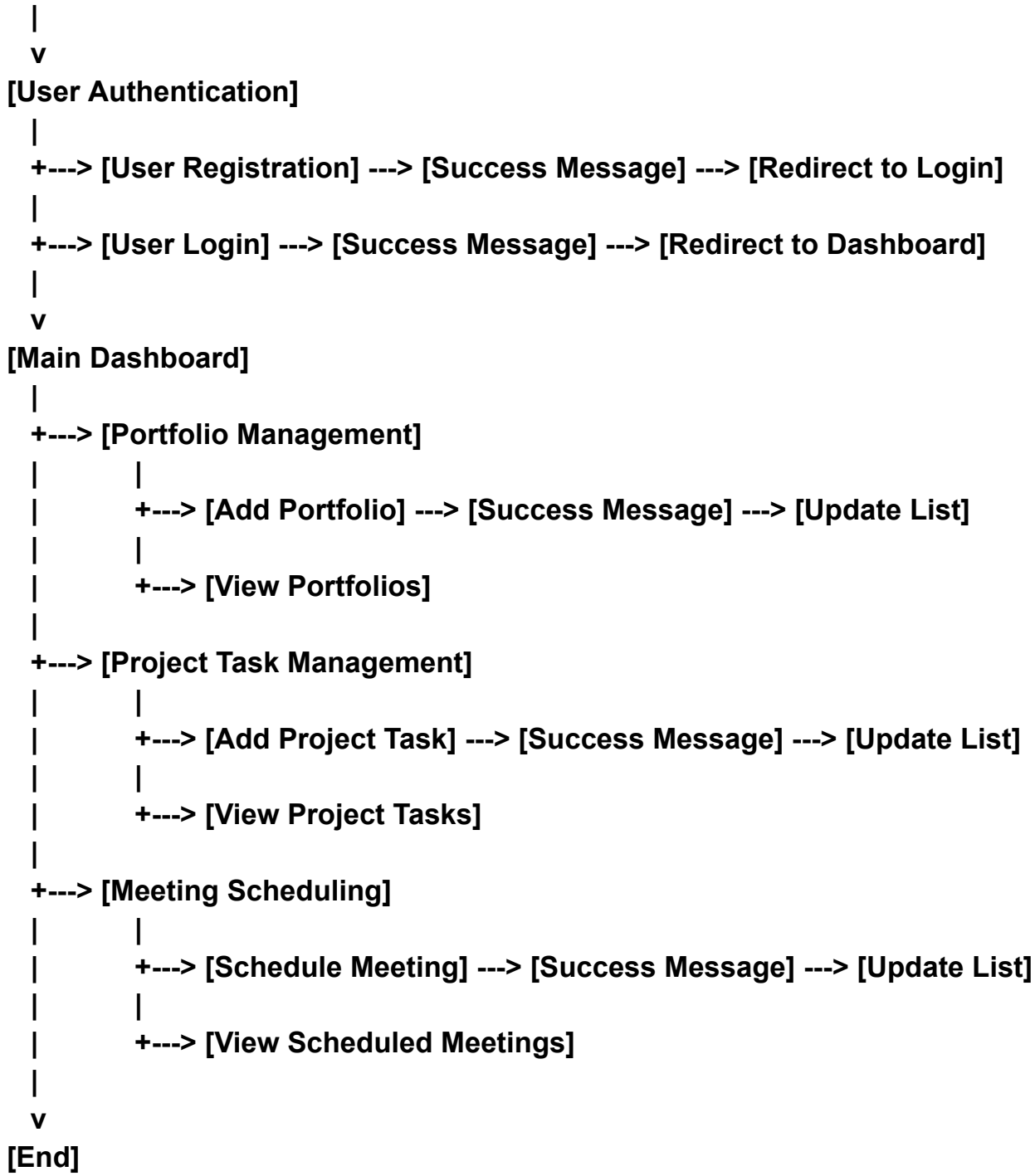
1. **Integration with JUnit**: Cucumber tests can be run with JUnit, allowing for seamless integration of BDD tests into the overall testing framework. This means that all tests—unit tests and BDD tests—can be executed together, providing comprehensive coverage of the application.

Running Tests

1. **JUnit Test Runner**: In Eclipse, JUnit test cases can be run using the JUnit test runner. This provides a user-friendly interface to execute tests and view results.
2. **Cucumber Integration**: Cucumber tests can also be executed from within Eclipse, and results are reported in a readable format. Cucumber generates a detailed report of test outcomes, indicating which scenarios passed and which failed.

Flow Diagram:

[Start]


[User Authentication]
[User Registration] ---> [Success Message] ---> [Redirect to Login]
[User Login] ---> [Success Message] ---> [Redirect to Dashboard]
[Main Dashboard]
[Portfolio Management]
[Add Portfolio] ---> [Success Message] ---> [Update List]
[View Portfolios]
[Project Task Management]
[Add Project Task] ---> [Success Message] ---> [Update List]
[View Project Tasks]
[Meeting Scheduling]
[Schedule Meeting] ---> [Success Message] ---> [Update List]
[View Scheduled Meetings]
[End]