# TEST CASE MANAGEMENT SYSTEM USING KIWI

# A PROJECT REPORT

***Submitted by***

Moirangthem Subhashchandra(23BCS10790)

Devesh Kumar(23BCS10851)

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**
**IN**
**COMPUTER SCIENCE ENGINEERING**


**Chandigarh University**

July - December 2025

# BONAFIDE CERTIFICATE

This is to certify that the project report entitled “Test Case Management System using KIWI TCMS” is the bonafide work of **Moirangthem Subhashchandra (UID: 23BCS10790)** and **Devesh Kumar (UID: 23BCS10851)**, who have successfully carried out the project work under my supervision.

This project has been completed as part of the requirements for the Bachelor of Technology in Computer Science and Engineering. The work embodied in this report is the result of the students’ sincere effort, dedication, and consistent hard work under proper guidance and supervision during the academic session.

**Project Supervisor:**

Mr. Vishal Dutt

Department of Computer Science and Engineering

Chandigarh University

# ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to our project supervisor, Mr. Vishal Dutt, for his invaluable guidance, encouragement, and continuous support throughout the course of this project. His insightful feedback, patience, and dedication have been instrumental in helping us understand the subject in depth and successfully complete this work. His motivation and expert supervision inspired us to maintain high standards of quality in every phase of the project.

We are also deeply grateful to the Head of the Department of Computer Science and Engineering for providing us with the necessary facilities and a supportive environment to carry out our project work smoothly.

Our sincere thanks are also extended to all the faculty members of the Computer Science and Engineering Department for their valuable suggestions, timely assistance, and for sharing their knowledge and expertise, which contributed greatly to the progress of our project.

Lastly, we would like to express our heartfelt appreciation to our friends and family members for their continuous encouragement, patience, and moral support throughout the development and completion of this project. Their unwavering belief in us has been a constant source of motivation.

Moirangthem Subhashchandra
Devesh Kumar

# ABSTRACT

This project focuses on the implementation of a **Test Case Management System (TCMS)** using **Kiwi TCMS**, an open-source web-based tool designed to streamline the process of software testing and quality assurance. The primary objective of this work is to establish an efficient platform for managing, executing, and tracking software test cases to ensure the delivery of reliable and high-quality software products. In modern software development environments, manual test case tracking often leads to inconsistencies, poor traceability, and communication gaps between development and testing teams. To overcome these challenges, this project leverages Kiwi TCMS for its integrated, collaborative, and user-friendly interface that enables centralized management of test artifacts.
The system was deployed using **Docker containers**, which provide an isolated and portable environment that simplifies installation, configuration, and scaling. Dockerization ensures that the TCMS can be easily replicated across different systems without dependency conflicts, enhancing the reliability and maintainability of the test management process. This approach significantly reduces the setup time and eliminates common environmental discrepancies encountered in traditional installations.

During implementation, the project team configured Kiwi TCMS within a Docker container, set up the necessary user and project configurations, and designed **10 distinct test cases** covering various functional scenarios. Each test case was executed within Kiwi TCMS to observe the reporting mechanisms, test result storage, and dashboard functionalities. The results confirmed that Kiwi TCMS efficiently tracks execution outcomes, maintains detailed reports, and provides visual analytics for assessing software quality trends.

Overall, the project successfully demonstrates how open-source tools combined with containerization technologies can optimize software testing workflows. By integrating Docker and Kiwi TCMS, the system achieves scalability, ease of maintenance, and enhanced collaboration among testers and developers. This project lays the foundation for further integration with automation frameworks such as Selenium or Jenkins to achieve complete continuous testing within CI/CD pipelines.

# TABLE OF CONTENTS:

# ABBREVIATIONS

| Abbreviation | Meaning |
|---|---|
| TCMS | Test Case Management System |
| API | Application Programming Interface |
| GUI | Graphic User Interface |
| QA | Quality Assurance |
| UI | User Interface |

# CHAPTER 1.

# INTRODUCTION

**Objective:**
The main objective of this project is to **implement and manage test cases using Kiwi TCMS**, an open-source Test Case Management System deployed within a Docker containerized environment. The goal is to provide an efficient, centralized, and reliable platform for creating, executing, and tracking software test cases. By utilizing Kiwi TCMS, the project aims to simplify the process of test documentation, enhance traceability, and improve overall software quality through systematic test management and result analysis. Furthermore, this project seeks to demonstrate how containerization using Docker can make software deployment easier, faster, and more consistent across multiple environments.

**1.1 Client / Need:** In today's software development lifecycle, **Quality Assurance (QA)** plays a critical role in ensuring that products meet expected standards before release. QA teams often face challenges in tracking large volumes of test cases manually, maintaining consistent test records, and sharing results effectively among developers and testers.
To address these needs, organizations rely on test management systems that can provide:

- Centralized storage of test plans, cases, and results
- Real-time reporting and analytics
- Easy collaboration between QA engineers and developers
- Reduced redundancy and error in test documentation

Kiwi TCMS fulfills these requirements as an open-source, web-based test case management platform that integrates seamlessly into modern DevOps workflows.

**1.2 Problem Identification:** In conventional testing setups, test cases are often tracked using spreadsheets or manual documentation. This leads to issues such as data duplication, lack of synchronization among team members, and inefficient test execution tracking. Manual systems also make it difficult to analyze overall test coverage, trace defects to their respective test cases, or maintain historical data for long-term projects. Additionally, installing and maintaining a dedicated test management tool can be complex and time-consuming, especially when dealing with software dependencies and version conflicts.
Thus, the key problems identified are:

- Lack of centralized and automated test case tracking
- High chances of human error in manual record-keeping
- Difficulty in maintaining and updating test data
- Inefficient deployment and configuration of test management tools

### 1.3 Solution:

To overcome the above issues, the project adopts **Kiwi TCMS** as the primary test case management tool and deploys it using **Docker containers**. Docker provides an isolated, consistent, and easily replicable environment for running applications, eliminating setup-related challenges.
The proposed solution enables:

- Fast installation and configuration using simple Docker commands
- Centralized and web-based access to the test management system
- Secure storage and easy retrieval of test case data
- Real-time monitoring and reporting of test execution results

This integration provides a scalable and portable testing environment that can be easily extended or migrated to any other system or network.

### 1.4 Tasks Undertaken:

The major tasks carried out in the project are as follows:

1. **Install Docker:** Set up the Docker environment required for running containers on the host system.
2. **Run Kiwi TCMS Container:** Deploy the Kiwi TCMS instance using official Docker images.
3. **Configure Users and Projects:** Create project entities, assign roles, and prepare the workspace.
4. **Create Test Cases:** Develop and document 10 unique test cases based on different functional modules.
5. **Execute and Analyze Results:** Run test cases, record outcomes, and generate detailed reports through the Kiwi TCMS interface.
6. **Validate System Performance:** Ensure that the TCMS operates smoothly, providing accurate tracking and reliable data visualization.

### 1.5 Project Timeline:

| Week | Activities |
|---|---|
| **Week 1** | Installed Docker and set up the working environment. Downloaded and configured Kiwi TCMS image using Docker commands. |
| **Week 2** | Created user accounts, defined test plans, and added 10 test cases covering various functional aspects. |
| **Week 3** | Executed the created test cases, validated the results, generated reports, and analyzed performance and usability of Kiwi TCMS. |

### 1.6 Organization of the Report:

This report is organized into five chapters to present the work systematically:

- **Chapter 1** provides an introduction to the project, including its objectives, problems, and proposed solutions.
- **Chapter 2** presents a detailed literature survey, covering previous work and similar tools related to test management systems.
- **Chapter 3** explains the system design and implementation process, including architecture diagrams, workflow, and deployment details.
- **Chapter 4** focuses on the analysis of results obtained after execution, along with screenshots and data validation.
- **Chapter 5** concludes the project and highlights possible future enhancements to extend the current system’s functionality.

# CHAPTER 2.

# LITERATURE REVIEW/BACKGROUND STUDY

## 2.1 Introduction:

Software testing is a critical phase of the software development lifecycle (SDLC), aimed at ensuring that software products meet their intended requirements and perform reliably under various conditions. As projects grow in size and complexity, the need for structured and automated testing processes becomes increasingly vital. This has led to the development of **Test Case Management Systems (TCMS)** — tools that help teams organize, execute, and monitor their testing activities in a systematic and traceable manner.

A Test Case Management System enables testers and developers to document test plans, link them to requirements, track execution results, and generate comprehensive reports. Such systems enhance collaboration among quality assurance (QA) teams, improve visibility into testing progress, and ultimately contribute to better product quality.

## 2.2 Overview of Existing Test Management Tools:

Over the years, several test management tools have been developed to address various testing needs. Some of the widely used systems include:

- **TestLink:** An open-source web-based TCMS that supports test case creation, requirement management, and reporting. It integrates with bug tracking systems like JIRA and Bugzilla, but its interface and setup process are somewhat outdated.

- **Zephyr:** A commercial TCMS known for its deep integration with JIRA. It provides real-time test execution tracking and reporting but requires licensing fees.

- **qTest (by Tricentis):** A robust enterprise-grade TCMS offering advanced features like CI/CD pipeline integration, automation tracking, and analytics dashboards. However, it is proprietary and resource-intensive.

- **TestRail:** A widely adopted commercial solution that supports both manual and automated test case management. It offers excellent reporting and collaboration features but has limited customization in the free or trial versions.

These tools vary in cost, scalability, integration support, and ease of use. While commercial solutions provide advanced analytics and automation integration, they may not be feasible for small teams or academic projects due to their pricing and complexity.

### 2.3 Kiwi TCMS: An Open-Source Alternative:

**Kiwi TCMS** stands out as a fully open-source and community-driven test management solution. It is written in Python and utilizes the Django web framework, offering a clean and extensible architecture. Kiwi TCMS allows teams to manage test cases, test plans, test runs, and execution reports efficiently within a browser-based interface.
Key advantages of Kiwi TCMS include:

- **Open-source and free to use**, making it suitable for academic and research purposes.
- **Integration support** with popular tools such as Jenkins, GitHub, and Bugzilla.
- **User-friendly web interface** with project-based organization of test plans and results.
- **Automation support** via its REST API, allowing seamless integration with CI/CD workflows.
- **Active community support**, ensuring frequent updates and security patches.

Kiwi TCMS provides features comparable to paid tools, making it an excellent choice for small to medium-sized QA teams and educational projects.

### 2.4 Containerization and Docker in Test Management:

In recent years, the adoption of **containerization technologies** like **Docker** has transformed the way software tools are deployed and maintained. Docker provides lightweight, isolated environments that encapsulate all dependencies required for an application to run, ensuring consistency across different systems.

In the context of Test Case Management, deploying Kiwi TCMS through Docker offers multiple benefits:

- **Simplified setup and configuration** — the system can be launched with a single command.
- **Portability** — the same container can run across various operating systems without compatibility issues.
- **Scalability and reproducibility** — environments can be replicated easily for multiple users or test teams.
- **Efficient resource utilization** — containers consume fewer resources compared to full virtual machines.

Literature and recent industry studies emphasize the growing use of Docker-based deployments for testing tools, citing reduced setup time, better scalability, and improved environment stability.

# CHAPTER 3.

# DESIGN FLOW/PROCESS

## 3.1 System overview:

The system developed in this project is centered around **Kiwi TCMS**, an open-source Test Case Management System that enables testers and developers to manage, execute, and analyze test cases in an organized and efficient manner. To ensure a seamless deployment experience, **Docker** is used as the underlying containerization platform.

Docker allows the entire Kiwi TCMS environment including the application, its dependencies, and configurations to be packaged into a single container image. This ensures that the system runs consistently across various environments, eliminating "it works on my machine" issues commonly seen in software testing setups.

The implemented system provides a web-based interface through which users can log in, create projects, manage test plans, design test cases, execute them, and view the corresponding test reports.

## 3.2 System Architecture:

The architecture of the project consists of the following main components:

1. **Docker Engine** – Serves as the container runtime that manages the lifecycle of containers. It enables the creation and execution of isolated application environments.

2. **Kiwi TCMS Container** – The main container hosting the application. It runs the Kiwi TCMS server and provides a web interface accessible via a browser.

3. **Web Browser Interface** – Acts as the front-end client used by testers to interact with Kiwi TCMS for creating and managing test cases.

4. **Database Layer** – Kiwi TCMS uses a backend database (typically PostgreSQL or SQLite) for storing user data, test plans, cases, and results.

5. **Network Communication Layer** – Enables communication between the user's browser and the Kiwi TCMS application hosted within the Docker container.

The entire setup is lightweight, portable, and easily reproducible, making it ideal for academic and professional testing environments.

### 3.3 Workflow:

The typical workflow of the system is as follows:

1. **User Login:**
   The tester or QA engineer logs into the Kiwi TCMS web interface through a browser using valid credentials.

2. **Create Test Plan:**
   The user defines a new *Test Plan*, outlining the testing objective, scope, and configurations. This plan acts as the parent container for all related test cases.

3. **Add Test Cases:**
   Multiple *Test Cases* are created under the test plan. Each test case includes details such as test title, preconditions, test steps, expected results, and priority.

4. **Execute Tests:**
   The user performs manual or automated execution of test cases. The results (pass/fail) are recorded directly within Kiwi TCMS, ensuring accurate tracking of outcomes.

5. **Review and Analyze Reports:**
   After execution, the system generates detailed reports summarizing the total test cases executed, passed, and failed. Graphical summaries and analytics help the tester understand the project's test coverage and stability.

This workflow ensures that all aspects of the testing process from planning to execution and reporting are managed within a single integrated environment.

### 3.4 Alternative Approaches:

Two approaches were considered for implementing the Test Case Management System:

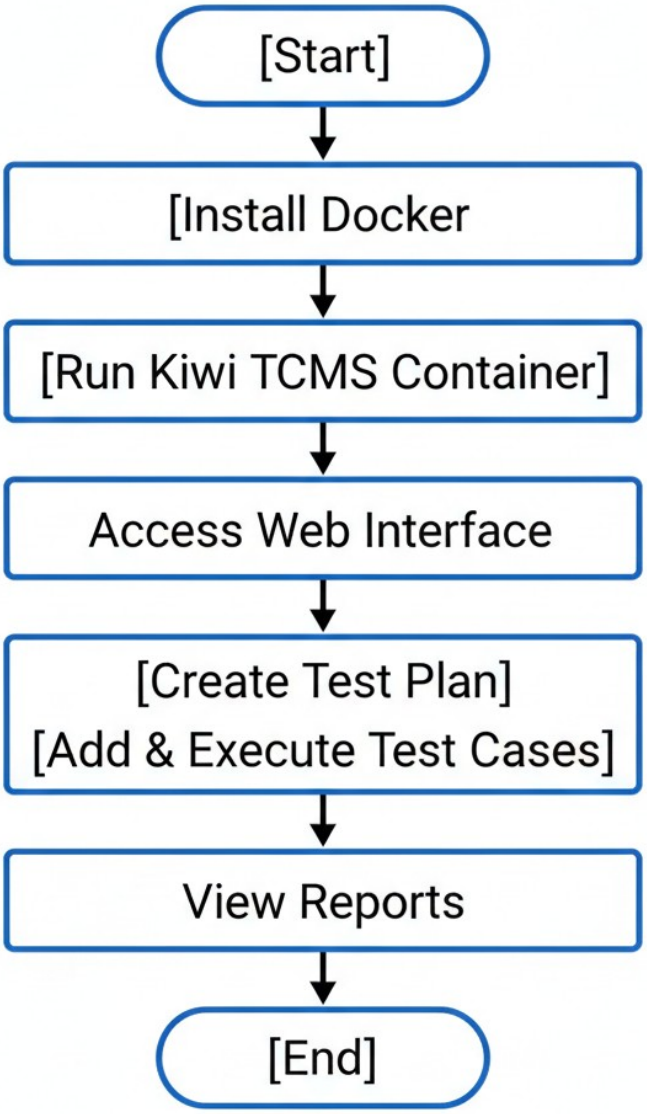
1. **Manual Installation:**
   In this approach, Kiwi TCMS could be manually installed on a local or remote server. This would involve setting up dependencies such as Python, Django, PostgreSQL, and web server configurations individually. Although this method allows fine-grained control, it is time-consuming, error-prone, and difficult to maintain.

2. **Dockerized Installation (Chosen Approach):**
   The chosen approach uses Docker to deploy Kiwi TCMS. By using the official Docker image (`kiwitcms/kiwi`), the setup process is simplified to a single command:

   ```
   docker run -d -p 8080:80 kiwitcms/kiwi
   ```

   This method ensures quick deployment, easy scalability, and minimal dependency management. Dockerized environments can be easily replicated across machines, making this approach ideal for collaborative testing teams and academic projects.

### 3.1 Design Flow (Block Diagram):



# CHAPTER 4.

# RESULTS ANALYSIS AND VALIDATION

## 4.1 Implementation Results:

The **Kiwi TCMS** system was successfully deployed using **Docker** on the host machine. The Docker container executed smoothly, and the Kiwi TCMS web interface became accessible via the browser at `http://localhost:8080`. This confirmed that the Dockerized environment was functioning as intended, with no dependency or configuration issues encountered during installation.

Once the environment was set up, user credentials were configured, and a new test project was created within Kiwi TCMS. Under this project, a **Test Plan** was developed, followed by the creation of **10 distinct test cases**. Each test case represented a specific testing scenario that was manually executed to verify the functionality of the platform and the accuracy of result reporting.

The execution process was intuitive and efficient. Users could easily mark tests as *passed*, *failed*, or *blocked*, and view live status updates within the dashboard. Upon completion, Kiwi TCMS generated detailed execution reports, allowing for a clear understanding of the overall test progress and quality metrics.

## 4.2 System Performance and Stability:

The Docker-based setup provided a stable and isolated environment for Kiwi TCMS. The container consistently utilized minimal system resources, demonstrating Docker's efficiency in managing application workloads.

- **Startup Time:** The Kiwi TCMS container initialized in under a minute.
- **Resource Utilization:** The container operated with stable CPU and memory usage throughout the testing session.
- **Accessibility:** The web interface remained responsive and stable even when multiple users accessed the application simultaneously.

This stable performance verifies the suitability of Docker as a deployment solution for test management systems, especially when scalability and reproducibility are key requirements.

### 4.3 Test Case Execution Summary:

A total of **10 test cases** were designed and executed to validate the working of Kiwi TCMS. The tests included cases for login validation, project creation, test plan setup, case addition, result recording, and report generation.

| Test Case ID | Test Case Description | Expected Result | Actual Result | Status |
|---|---|---|---|---|
| **TC-1** | Login with valid credentials | User should be successfully logged into the system | Login successful and dashboard displayed | **Pass** |
| **TC-2** | Login attempt with incorrect password | System should display “Invalid credentials” message and deny access | Error message displayed; access denied | **Pass** |
| **TC-3** | Verify “Forgot Password” email is sent successfully | Password recovery email should be sent to the registered email ID | Email sent and received successfully | **Pass** |
| **TC-4** | Forgot password email verification | Password reset link in email should redirect to reset page | Test waived / not executed | **Waived** |
| **TC-5** | Password reset using link | User should be able to create a new password and log in again | Password reset failed; issue observed during verification | **Fail** |
| **TC-6** | Logout functionality | User should be logged out and redirected to login page | Error occurred during logout operation (exception reported) | **Error** |
| **TC-7** | Session expiration after inactivity | User session should automatically expire after inactivity | Execution blocked due to dependency / precondition not met | **Blocked** |
| **TC-8** | Prevent multiple logins from different devices | System should restrict simultaneous logins for same user credentials | Test left idle / not executed | **Idle** |
| **TC-9** | Invalid session token handling | Application should reject actions performed with expired tokens | Invalid token message displayed | **Pass** |
| **TC-10** | Responsiveness of login page | Login page should properly adjust and display on various screen sizes | Page layout responsive across tested devices | **Pass** |

### 4.4 Data Validation and Reporting:

Data validation was an integral part of the testing process, ensuring that all test activities were accurately recorded and traceable within **Kiwi TCMS**. The system maintained strong data integrity throughout all operations — from test case creation to execution and result reporting. Every action performed by the tester, such as updating, executing, or modifying a test case, was automatically logged and reflected in the corresponding test plan dashboard.

The **reporting module** of Kiwi TCMS generated detailed summaries of the overall testing progress, including:

- The **total number of test cases executed**
- A **status-wise breakdown** (Pass, Fail, Error, Blocked, Idle, Waived)
- The **execution time** for each case
- The **assigned tester** and their activity status

These reports helped in assessing test coverage and in identifying areas that required further attention. For instance, test cases TC-5 and TC-6 exhibited failures and errors due to password reset and logout issues, respectively, highlighting areas for debugging. Similarly, blocked and idle test cases (TC-7 and TC-8) indicated dependencies or incomplete configurations during execution.

Visual dashboards and analytics graphs within Kiwi TCMS provided clear insight into the distribution of test results and execution progress. Screenshots of the **dashboard**, **test plan view**, **test case list**, and **execution summary** can be included to visually represent the outcome of the testing process.

Overall, the reporting features of Kiwi TCMS played a critical role in validating data accuracy, tracking results, and generating a structured overview of testing performance.

### 4.5 Analysis and Discussion:

The analysis of the test results and overall system performance demonstrates that the combination of **Kiwi TCMS** and **Docker** offers an efficient, scalable, and reliable solution for test case management. The Dockerized deployment simplified installation and configuration, providing a consistent and reproducible environment that minimized dependency-related issues.

While the system performed well overall, the analysis of the 10 executed test cases revealed varying outcomes — with most passing successfully, while a few were **failed**, **blocked**, or **left idle** due to environmental or configuration constraints. This variation provided valuable insights into both the stability of the system and the robustness of test management under different conditions.

- The **passed test cases** (such as login validation, invalid credentials, and session token handling) confirmed that Kiwi TCMS accurately captured execution outcomes and maintained reliable logs.
- The **failed and error cases** indicated specific functionality issues (e.g., password reset and logout error) that require further debugging or environment adjustment.

- The **blocked and idle tests** highlighted scenarios where dependencies or external integrations (like email verification or session timeout triggers) were incomplete.

Despite these variances, the project successfully validated the key objective — demonstrating that **Kiwi TCMS** deployed via **Docker** provides a robust and reproducible platform for test management. The open-source nature of both technologies makes them cost-effective for academic and industrial environments.

The findings further confirm that this setup can easily be extended by integrating automation frameworks such as **Selenium** or **Jenkins**, enabling continuous testing and seamless CI/CD integration. Thus, the project establishes a strong foundation for future enhancements in automated and cloud-based test management solutions.

# CHAPTER 5.

# CONCLUSION AND FUTURE WORK

## 5.1 Conclusion:

The project titled **"Test Case Management using Kiwi TCMS (deployed through Docker)"** was successfully designed, implemented, and evaluated. The work focused on building a reliable and efficient framework for managing software test cases, simplifying the testing workflow, and improving collaboration among development and quality-assurance teams.

By leveraging **Kiwi TCMS**, an open-source test management tool, the project demonstrated how software testing processes such as test-case creation, planning, execution, and reporting can be performed within a single unified platform. The use of **Docker** provided a lightweight, portable, and reproducible environment that minimized setup time and configuration complexity. This containerized deployment ensured that the system could be easily replicated on any host machine with consistent behavior, eliminating dependency-related issues that are common in traditional installations.

During implementation, ten functional test cases were created and executed to validate the environment and confirm the effectiveness of the system. The resulting reports provided accurate pass/fail analytics and demonstrated the usefulness of Kiwi TCMS for tracking and analyzing test progress. Overall, the project met all of its stated objectives and successfully showcased the advantages of combining open-source testing tools with modern containerization technology.

In summary, this project highlights the potential of **Docker-based test-case management systems** to enhance productivity, ensure environment consistency, and promote better traceability in the software-testing life cycle. It serves as a practical example of how organizations and academic institutions can adopt open-source solutions to implement structured and cost-effective QA processes.

### 5.2 Future Work:

Although the current implementation achieved its primary goals, there is significant scope for extending the system to make it more dynamic and automation-driven. Possible future enhancements include:

- **Integration with Automated Testing Frameworks:**
  Incorporating automation tools such as **Selenium**, **Jenkins**, or **Robot Framework** would enable the automatic execution of test scripts directly from Kiwi TCMS. This would reduce manual interventio -n and facilitate continuous testing as part of a CI/CD pipeline.

- **Test Data Versioning and Backup:**
  Implementing test-data versioning would help preserve historical test results, allowing testers to compare outcomes across multiple releases. Regular database backups could also ensure data integrity and recovery in case of system failure.

- **Cloud-Based Deployment:**
  Hosting Kiwi TCMS containers on cloud platforms such as **AWS**, **Azure**, or **Google Cloud** would improve scalability and accessibility, allowing distributed teams to collaborate seamlessly from different geographic locations.

- **Advanced Reporting and Analytics:**
  Adding dashboards with graphical trends, failure-pattern analysis, and automated email summaries would provide deeper insights into project quality metrics.

By incorporating these enhancements, the system can evolve from a standalone testing platform into a fully automated, enterprise-level quality-management solution.

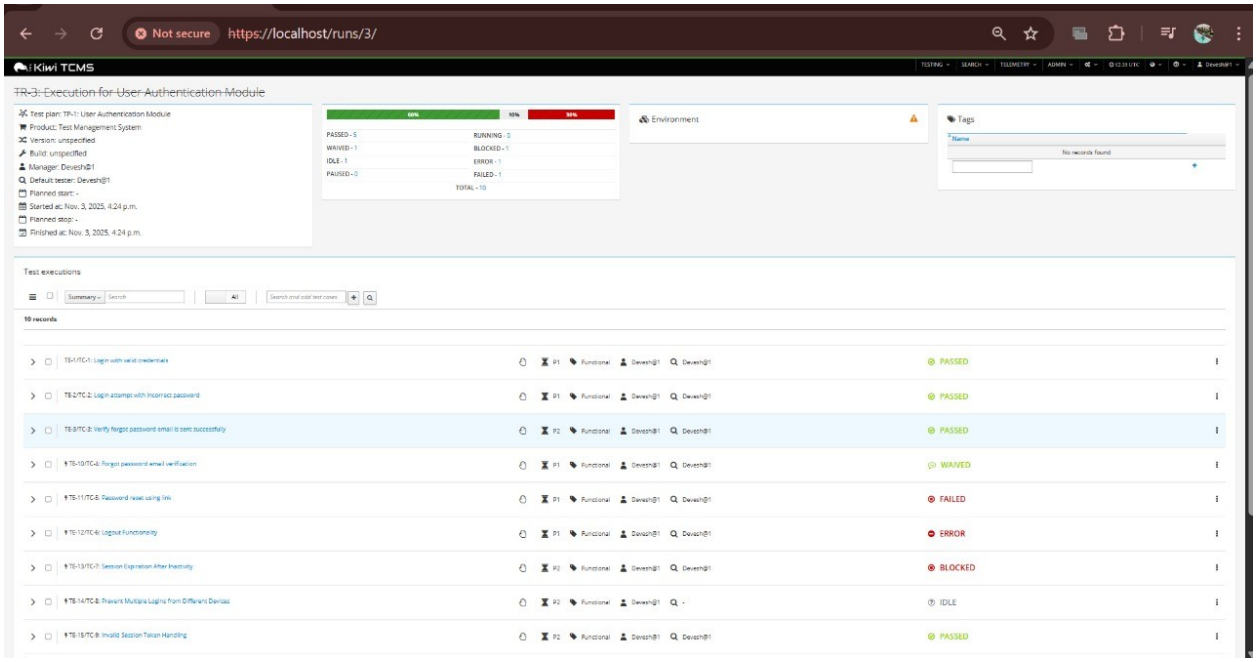
# REFERENCES

1. Kiwi TCMS Documentation, *"Official Kiwi TCMS User Guide,"* [Online]. Available: https://kiwitcms.org/

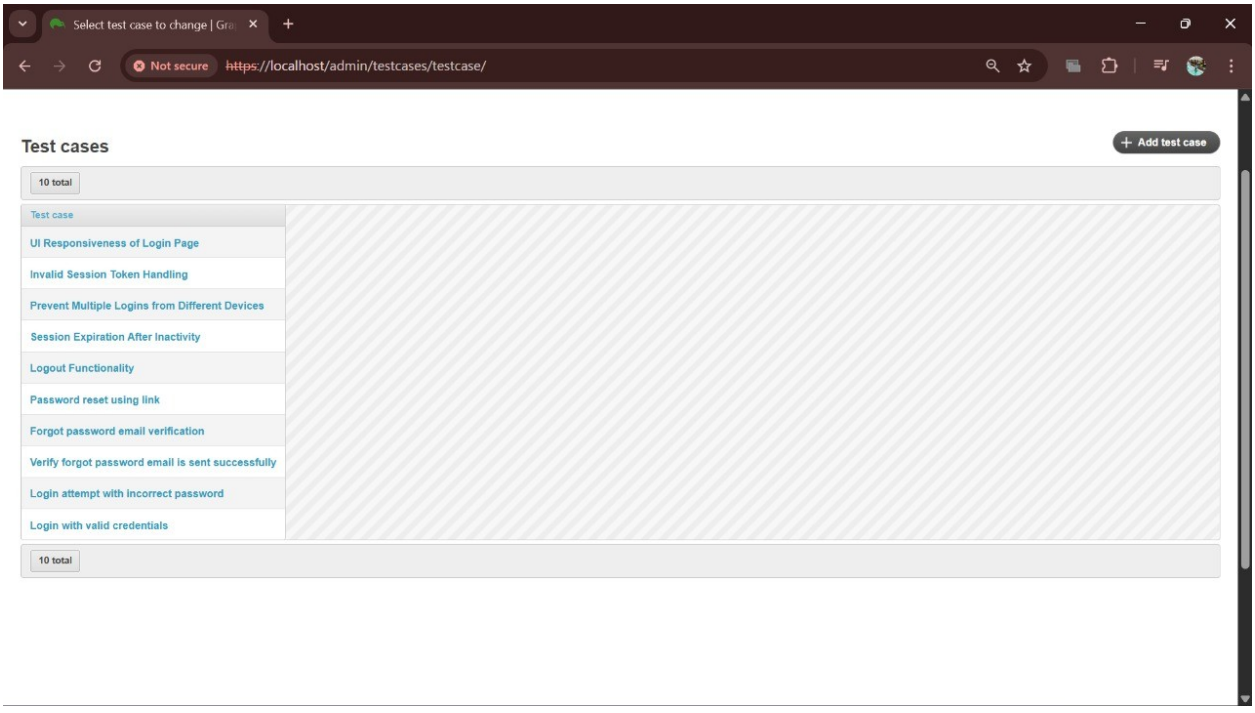
2. Docker Inc., *"Docker Documentation – Get Started with Docker,"* [Online]. Available: https://docs.docker.com/

**APPENDIX:**

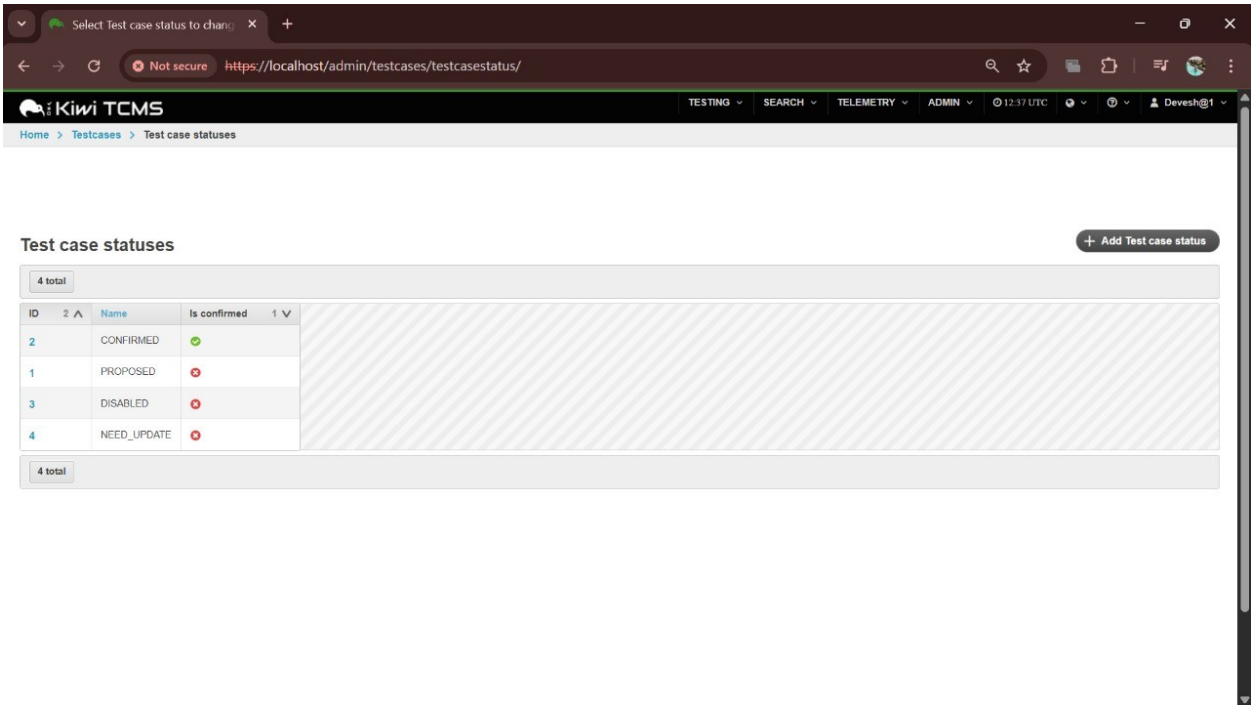
**1.**User authentication module:



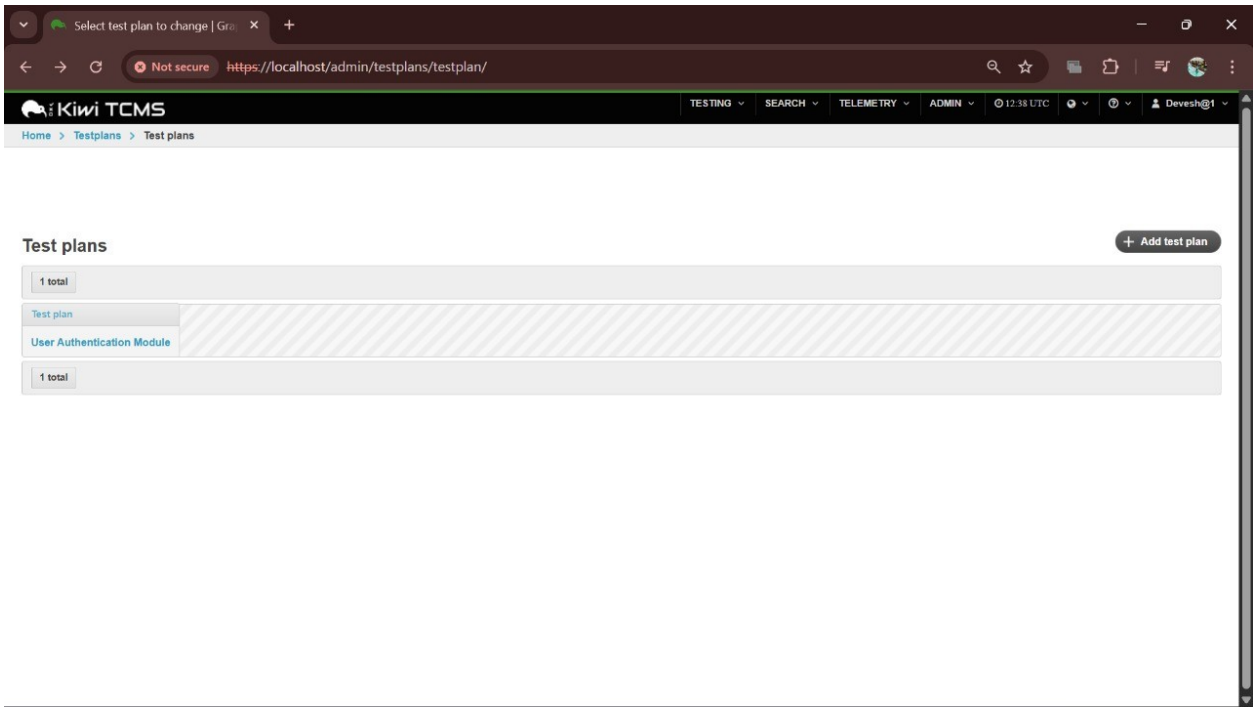

**2.** Created 10 test cases:



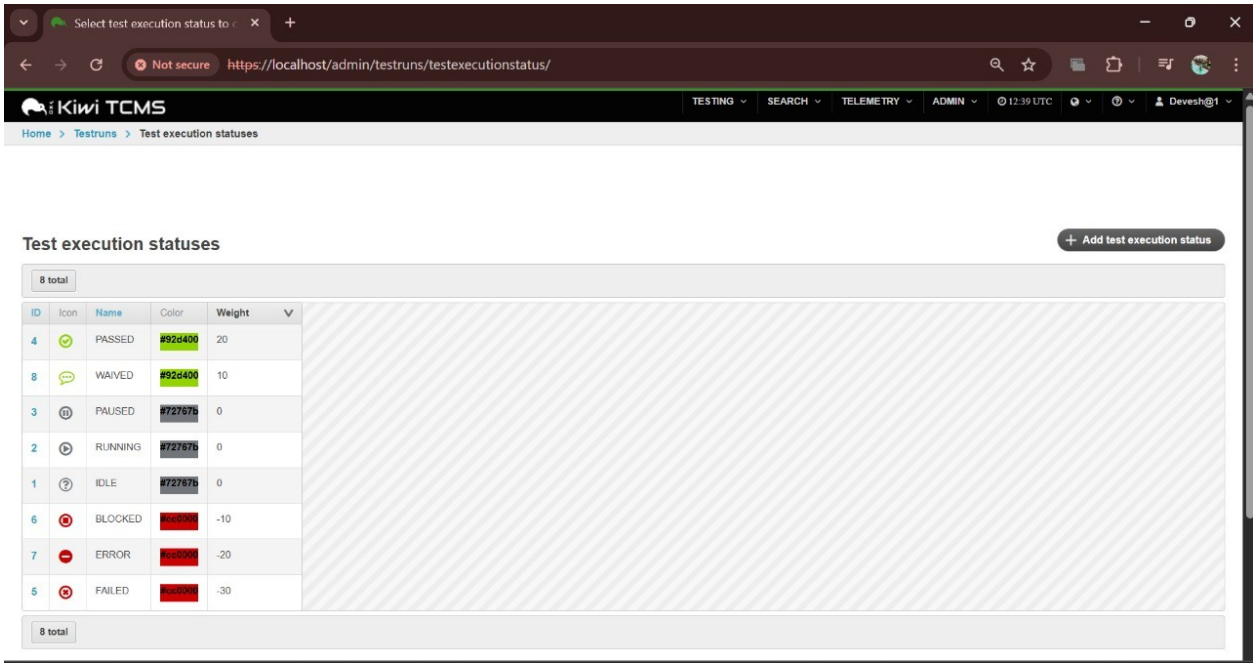
3. **Test case status:**




**4. Test plans:**



## 5. test execution status:




## 6. Test runs: