Maintenance and Support File

Car Rental System

Thanushka Praveen Wickramarachchchi
Student ID: 270599091

Yoobee College
Master of Software Engineering

Course: MSE800 Professional Software Engineering
Assignment 1: Object-Oriented Programming
Assignment Type: Individual Project

Date: February 3, 2025

Contents

1 Introduction

The Car Rental System is designed to enhance operational efficiency and improve the rental experience. This document outlines strategies for managing software maintenance, versioning, and backward compatibility to ensure long-term stability, reliability, and scalability.

2 Managing Software Maintenance

To ensure the system runs smoothly and remains reliable, the following strategies will be implemented:

2.1 Regular Updates

- **Bug Fixes and Improvements**: Continuously address bugs, improve features, and enhance security.
- **User Feedback**: Collect and analyze user feedback to prioritize the most critical issues.

2.2 Tracking Issues

- **Tools**: Utilize tools such as GitHub Issues or JIRA to log and manage bugs, feature requests, and tasks.
- **Prioritization**: Organize issues based on their importance (e.g., critical bugs, new features, performance optimizations).

  maintain

2.2.1 Issue Tracking Process

use **GitHub Project's Issue Area** to track bugs, feature requests, and general issues encountered during development. Each issue is assigned a **unique identifier** and includes relevant details such as:

- **Issue Title:** A concise description of the problem.
- **Description:** A brief explanation of the issue, including the expected vs. actual behavior.
- **Steps to Reproduce:** A step-by-step guide to replicate the issue.
- **Assigned Developer:** The team member responsible for resolving the issue.
- **Priority:** Categorized as High, Medium, or Low based on the impact.

Tagging System To streamline issue tracking, we use specific **tags (labels)** to categorize and prioritize issues. Below are some common tags and their explanations:

Tag Name	Description
`bug`	Identifies issues related to software defects.
`feature-request`	Requests for new functionality or enhancements.
`critical`	Urgent issues that must be fixed immediately.
`high-priority`	Important bugs that need quick resolution.
`medium-priority`	Issues that are important but not urgent.
`low-priority`	Minor issues that do not impact functionality significantly.
`UI/UX`	Issues related to user interface and user experience.
`backend`	Issues specific to backend logic and services.
`frontend`	Issues related to UI, components, and client-side functionality.
`documentation`	Improvements or corrections related to documentation.

Issue Resolution Workflow

1. **Create an Issue:** Developers or testers report issues with relevant details and assign appropriate tags.
2. **Assign and Prioritize:** The issue is assigned to a developer based on expertise and urgency.
3. **Work on Fixes:** The assigned developer works on the fix and updates the issue with progress notes.
4. **Code Review & Testing:** The fix undergoes a review and is tested before merging.
5. **Closing the Issue:** Once verified, the issue is marked as **Resolved** and closed.

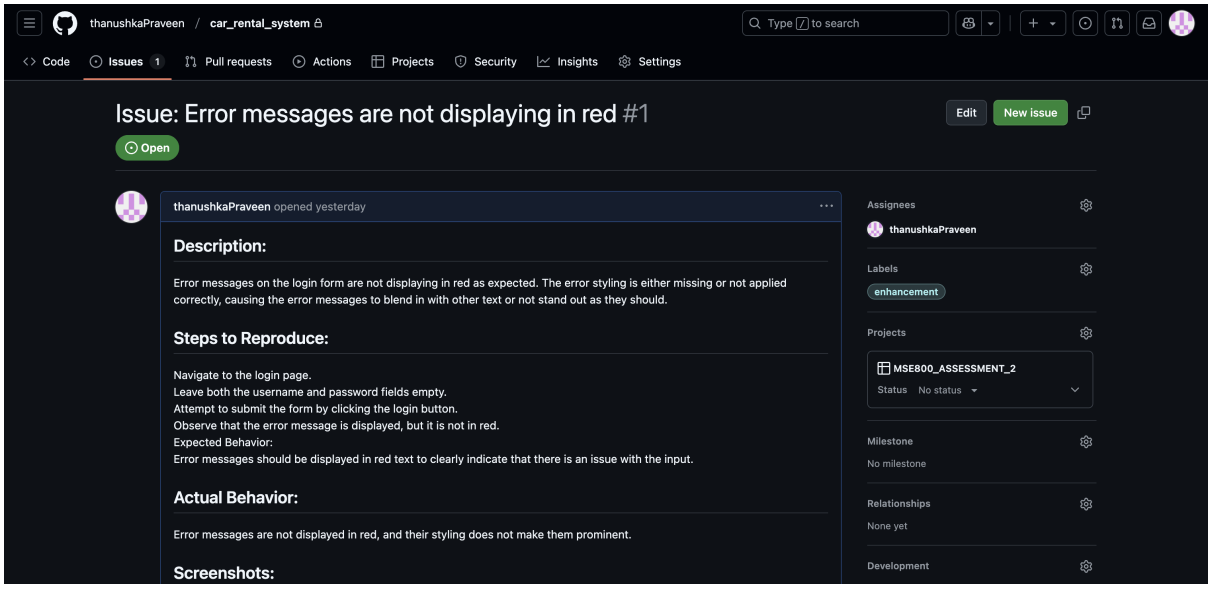

Figure 1: GitHub Issue Tracking Interface

By following this structured approach, we ensure better **organization, tracking, and resolution** of bugs and enhancements in the project.

2.3 Code Improvements

- **Code Reviews**: Regularly review and clean code to improve readability and maintainability.
- **Modular Design**: Use modular coding practices to ensure different parts of the system can be updated independently.

2.4 Updated Documentation

- **User Guides**: Keep user-facing documentation updated with each change.
- **Technical Documents**: Update technical documentation and maintain a detailed changelog for each version.

2.5 System Monitoring

- **Logging**: Implement logging mechanisms to identify and diagnose issues during runtime.
- **Performance Monitoring**: Continuously monitor system performance to detect and address potential problems early.

3 Versioning Strategy

Versioning ensures structured updates and a clear record of system changes. The Car Rental System follows **Semantic Versioning** (**MAJOR.MINOR.PATCH**):

3.1 Versioning Structure

- **MAJOR** – Significant changes that may break backward compatibility.
  Example: `v2.0.0` (Complete overhaul of the booking system)
- **MINOR** – New features added without breaking existing functionality.
  Example: `v1.1.0` (Added support for online payments)
- **PATCH** – Small fixes and optimizations.
  Example: `v1.0.1` (Fixed a calculation error in rental fees)

3.2 Release Process

- **Pre-release Testing:** New updates are first deployed in a testing environment.
- **Beta Release:** Select users test major updates before final deployment.
- **Stable Release:** Fully tested versions are deployed to production.
- **Rollback Mechanism:** Ability to revert to a previous version if critical issues arise.

4 Backward Compatibility

Ensuring backward compatibility is crucial for user satisfaction and system reliability. The following practices will be adopted:

4.1 Database Changes

- **Non-Destructive Updates**: Add new tables or fields rather than removing or renaming existing ones.
- **Migration Tools**: Use tools like Alembic for safe and seamless database migrations.

4.2 Modular Architecture

- **Separation of Concerns**: Implement features using separate functions and class-based models, ensuring that any feature can be modified or removed without affecting others.
- **Encapsulation**: Group related functionalities into modular components, making maintenance and enhancements easier.

4.3 Feature Toggles

- **Modular Feature Control**: Implement feature toggles to enable or disable functionalities dynamically without code modifications.
- **Environment-Based Activation**: Allow feature toggles to be configurable per environment (e.g., development, testing, production).
- **Granular Control**: Provide control at the module or component level so that individual features can be turned on or off as needed.
- **Backward Compatibility**: Ensure old features remain accessible through toggles while new features are introduced, preventing disruptions.
- **User-Specific Feature Access**: Implement role-based or user-specific toggles to activate features for specific user groups.

4.4 Testing

- **Comprehensive Testing**: Test updates rigorously to ensure existing features remain functional.

5 Key Principles

5.1 Scalability

- **Future Growth**: Design the system to handle increased users or data seamlessly.
- **Modular Code**: Keep the codebase modular to simplify scaling and updates.

5.2 Reliability

- **Thorough Testing**: Ensure updates are thoroughly tested before release.
- **Quick Bug Fixes**: Address discovered bugs promptly.

5.3 Maintainability

- **Clean Code**: Write clear and concise code to facilitate easy updates and debugging.
- **Comprehensive Documentation**: Maintain accurate and up-to-date documentation.

5.4 Extensibility

- **Flexible Design**: Enable easy addition of new features without disrupting existing ones.
- **Design Patterns**: Implement design patterns like Factory, Singleton, and MVC.

6 Maintenance and Support Plan

6.1 Maintenance

- **Monitoring**: Continuously monitor system performance and identify bugs.
- **Regular Updates**: Release consistent updates to address issues and introduce enhancements.

6.2 Versioning

- **Clear Numbering**: Use semantic versioning to manage updates effectively.
- **Automation**: Automate versioning processes for consistency.

6.3 Backward Compatibility

- **Non-Disruptive Updates**: Ensure updates do not break existing features.
- **Documentation**: Provide clear migration guides and documentation for users transitioning to updated versions.

7 Conclusion

By implementing these maintenance strategies, version control techniques, and backward compatibility practices, the Car Rental System will remain stable, efficient, and scalable. Following best practices in software engineering ensures that the system continues to evolve while maintaining high-quality standards.