Software Architecture Document

Model-Based Testing Bundle

Vo Xuan Tien

Version 1.0

04/29/2019

Revision History

Version	Description of Versions / Changes	Responsible Party	Date
1.0	Initial version	Vo Xuan Tien	4/29/19

Approval Block

Version	Comments	Responsible Party	Date

Table of Contents

Software Architecture Document

1. Introduction

1.1. Purpose

This document provides a comprehensive architectural overview of the Model-Based Testing Bundle (MBT Bundle), using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system. This document is for users, developers and other stakeholders who want to learn how MBT Bundle work.

1.2. Scope

This document explains the architecture of MBT Bundle version 1.6.

1.3. Definitions, Acronyms, and Abbreviations

- **MBT** – Model Based Testing
- **SAD -** Software Architecture Document
- **User -** This is any user who is registered a new account on MBT Bundle. It's usually a tester who test a system

1.4. References

Software Architecture Document template:
https://projects.cecs.pdx.edu/attachments/download/3146/Software_Architecture_Document.docx

Software Architecture Document:
http://www.se.rit.edu/~raindelay/Documents/Artifacts/Software%20Architecture%20Document.doc

Sample Software Architecture Document: http://faculty.csupueblo.edu/rick.huff/cis432/sad-onlinecateringservice.doc

Software Architecture Document: http://www.se.rit.edu/~royalflush/documents/SAD.doc

1.5. Overview

In order to fully document all the aspects of the architecture, the Software Architecture Document contains the following subsections.

Section 2: describes the use of each view

Section 3: describes the architectural goals and constraints of the system

Section 4: describes the most important use-case realizations

Section 5: describes logical view of the system including interface and operation definitions.

Section 6: describes significant persistence elements.

Section 7: describes how the system will be deployed.

2. Architectural Representation

This document details the architecture using the views defined in the "4+1" model [Kruchten]. The views used to document the MBT Bundle are:

Use Case view

Audience: all the stakeholders of the system, including the end-users.
Area: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system.
Related Artifacts: Use-Case diagram, Sequence diagram

Logical view

Audience: Designers.
Area: Functional Requirements: describes the design's object model. Also describes the most important use-case realizations.
Related Artifacts: State diagram

Data view (optional)

Audience: Data specialists, Database administrators
Area: Persistence, describes the architecturally significant persistent elements in the data model
Related Artifacts: Data diagram.

Implementation view

Audience: Programmers.
Area: Software components: describes the layers and subsystems of the application.
Related Artifacts: Component diagram

Process view

Audience: Integrators.
Area: Non-functional requirements: describes the design's concurrency and synchronization aspects.
Related Artifacts: Activity diagram.

Deployment view

Audience: Deployment managers.
Area: Topology, describes the mapping of the software onto the hardware and shows the system's distributed aspects.
Related Artifacts: Deployment diagram.

3. Architectural Goals and Constraints

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

1. The project is an open source project. Therefore, one goal of this document is to be useful to our stakeholders in the future.

2. The system must be easy to extend and customize by other developers.

3. The system will be written using PHP language, and re-use as much as possible open source projects that has compatible license (MIT). Therefore, there are some technical debts from those open source projects that we have to accept and work around.

4. The system must communicate with multiple third-party APIs like Slack. Defining how the system interfaces with these third-party systems is a concern of the architecture to make the system more user-friendly.

4. Use-Case View

The purpose of the use-case view is to give additional context surrounding the usage of the system and the interactions between its components. For the purposes of this document, each component is considered a use-case actor. Section 4.1 lists the current actors and gives a brief description of each in the overall use context of the system. In section 4.2, the most common use-cases are outlined and illustrated using UML use-case diagrams and sequence diagrams to clarify the interactions between components.

4.1. Actors

Tester

The person who want to test the system under test using MBT Bundle.

Admin

The person who want to manage other testers.

User

Both testers and administers.

MBT Bundle

This system includes admin, API, worker.

Database

The system that store data (e.g. MariaDB).

Queue

The system that store sequence of messages (e.g. RabbitMQ).

Continuous Integration

The system that automate the build and test of code every time a team member commits changes to version control.

4.2. Use-Case Realizations

4.2.1. Tasks

Create and manage tasks.

Figure 4.1 Tasks Use Case Diagram

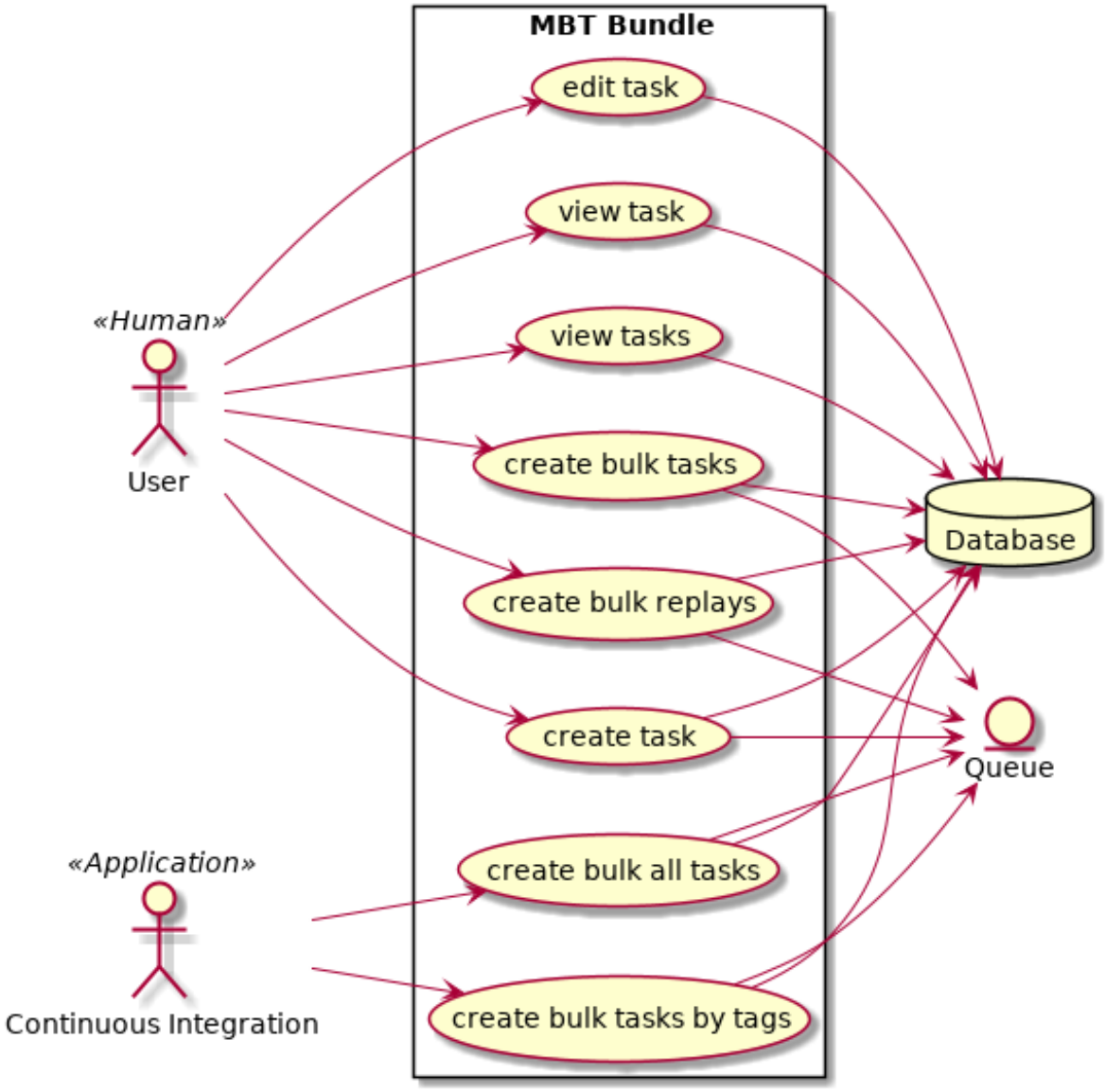
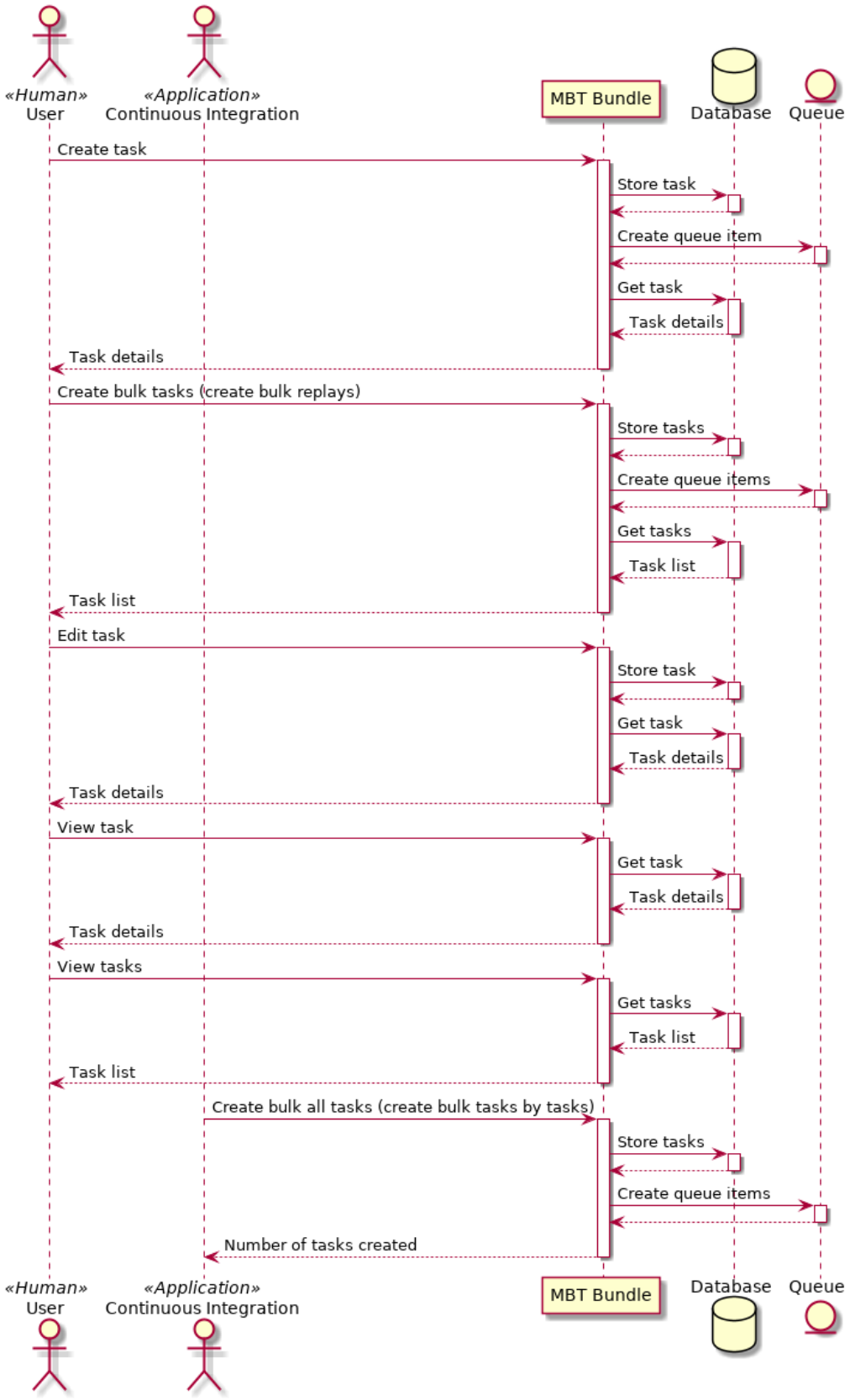


Figure 4.2 Bugs Sequence Diagram



4.2.2. Bugs

Manage bugs.

Figure 4.3 Bugs Use Case Diagram

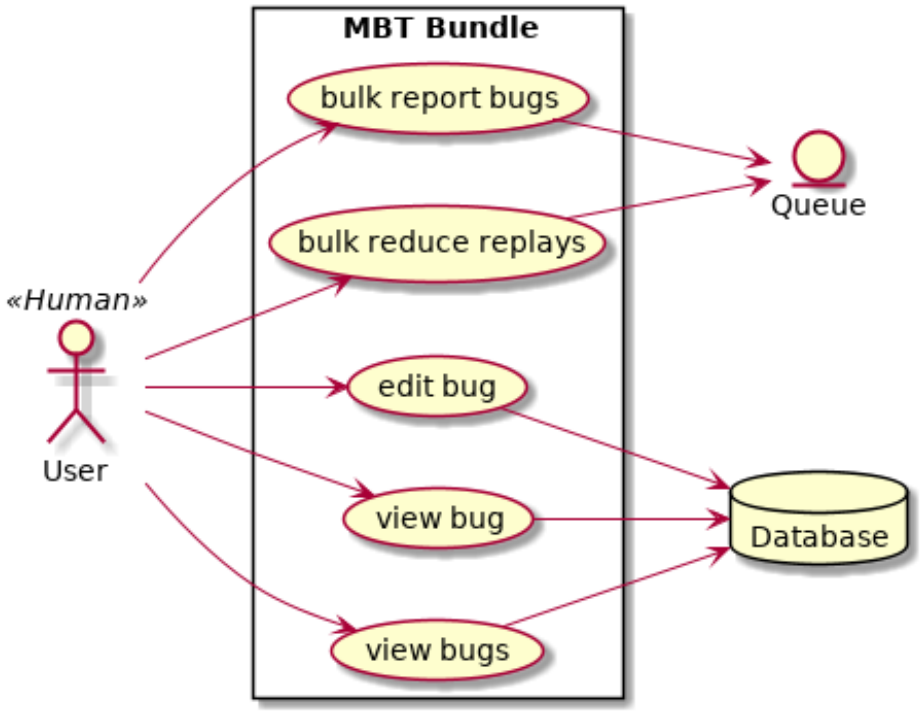
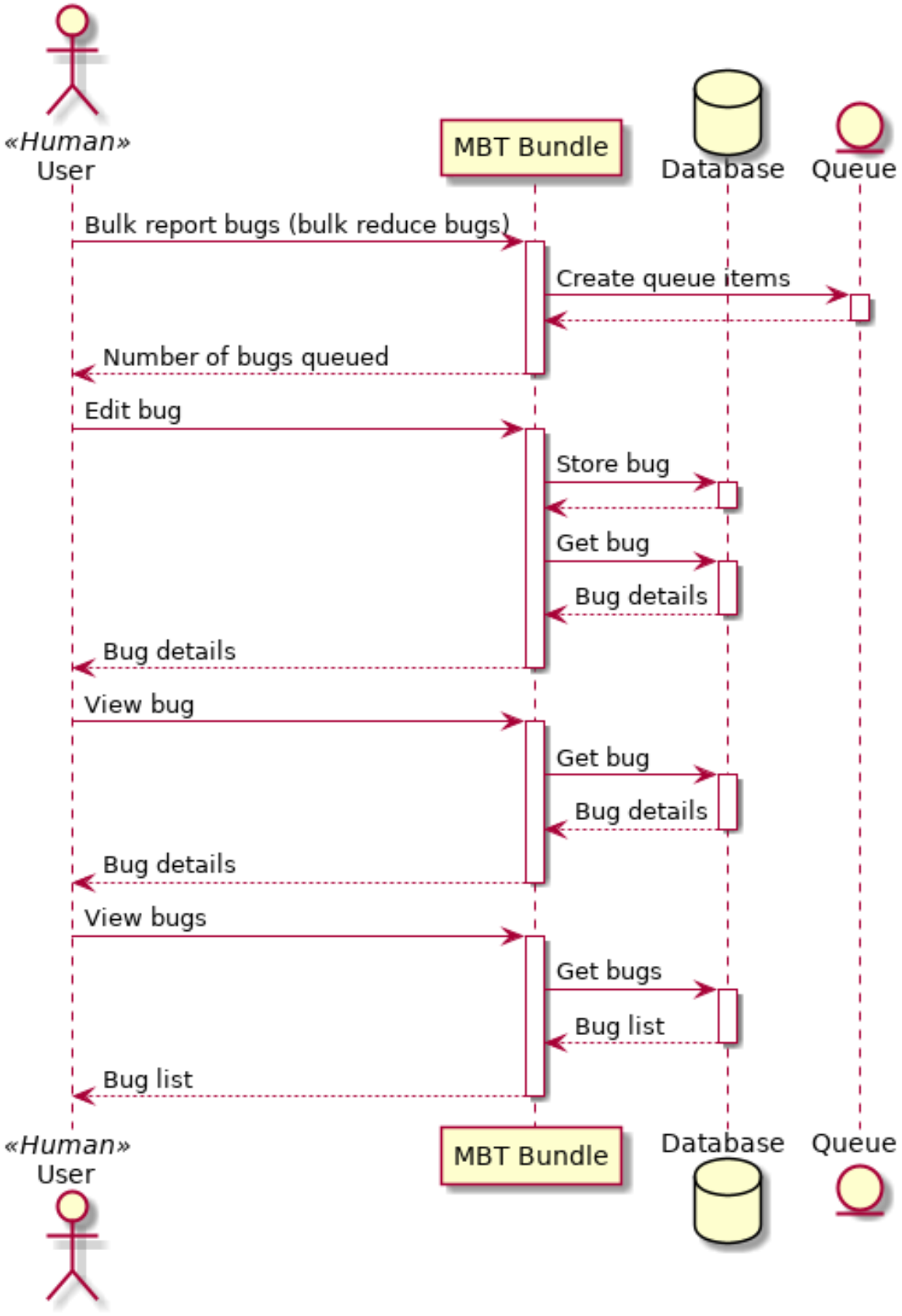


Figure 4.4 Bugs Sequence Diagram



4.2.3. Models

Manage models.

Figure 4.5 Models Case Diagram

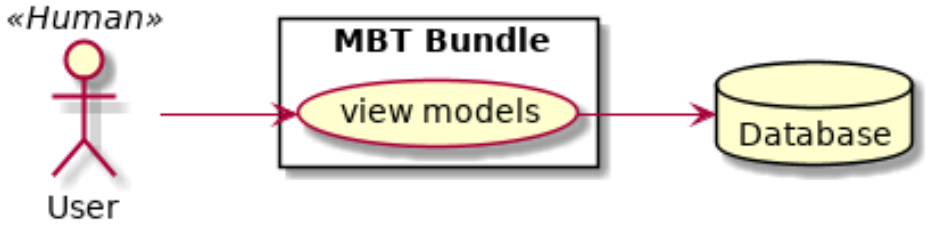
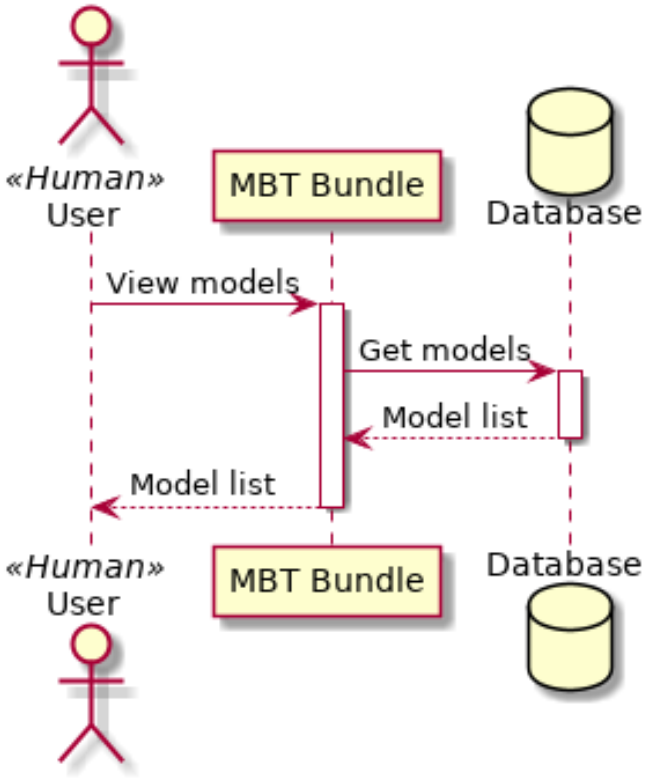


Figure 4.6 Models Sequence Diagram



4.2.4. Users

Create, register and manage users.

Figure 4.7 Users Use Case Diagram

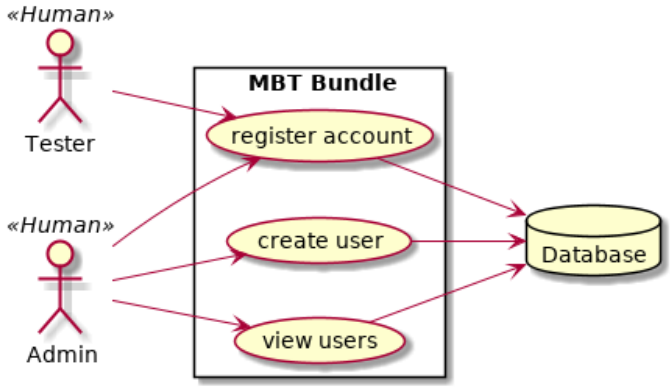
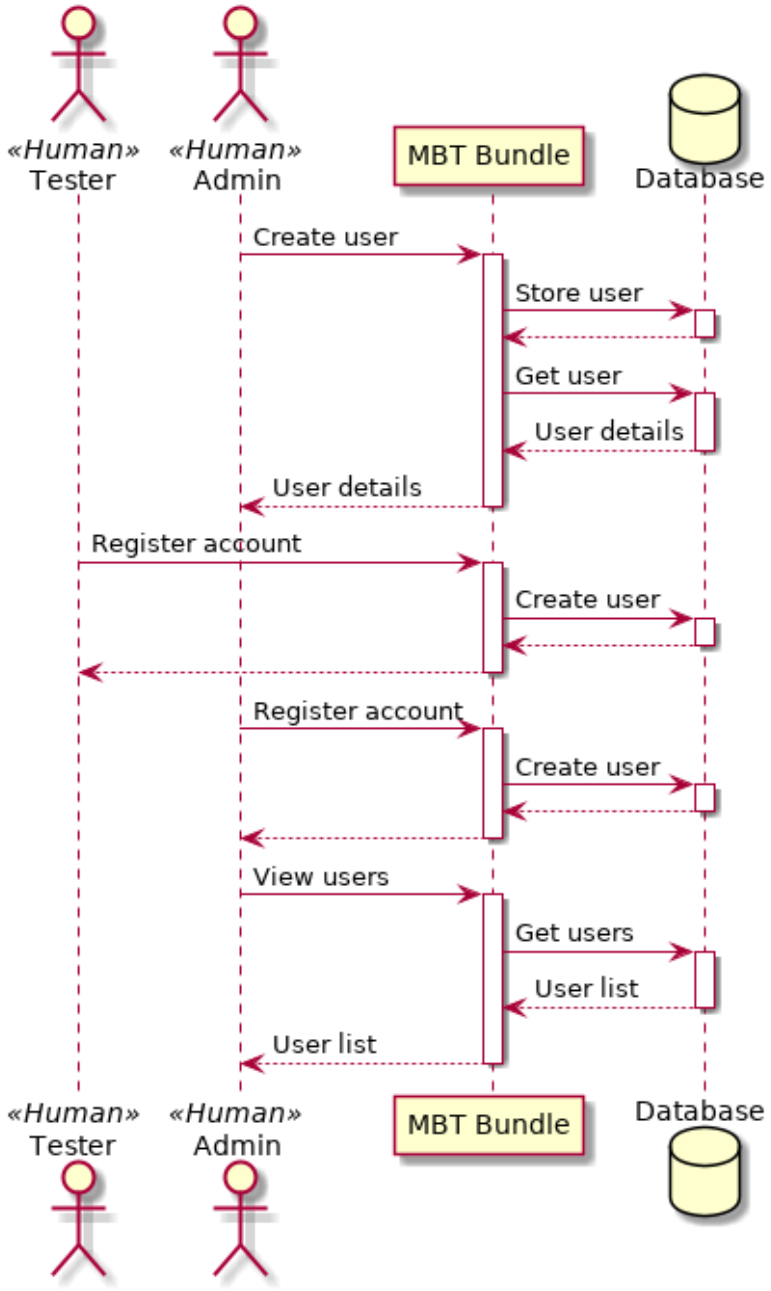


Figure 4.8 Users Sequence Diagram



4.2.5. Dashboard

View dashboard.

Figure 4.9 Dashboard Use Case Diagram

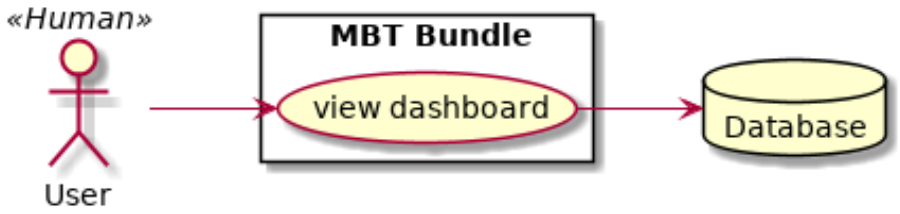
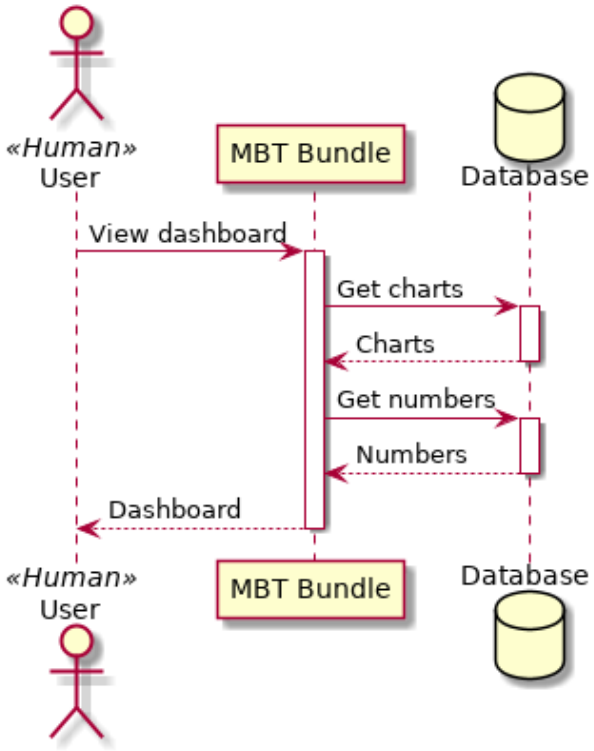


Figure 4.10 Dashboard Sequence Diagram



4.2.6. Configuration

Change configuration once log in.

Figure 4.11 Configuration Use Case Diagram

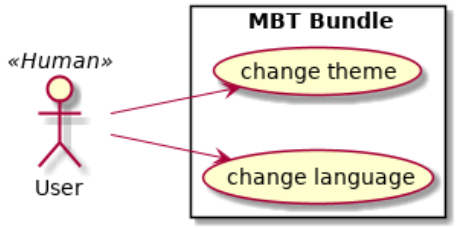
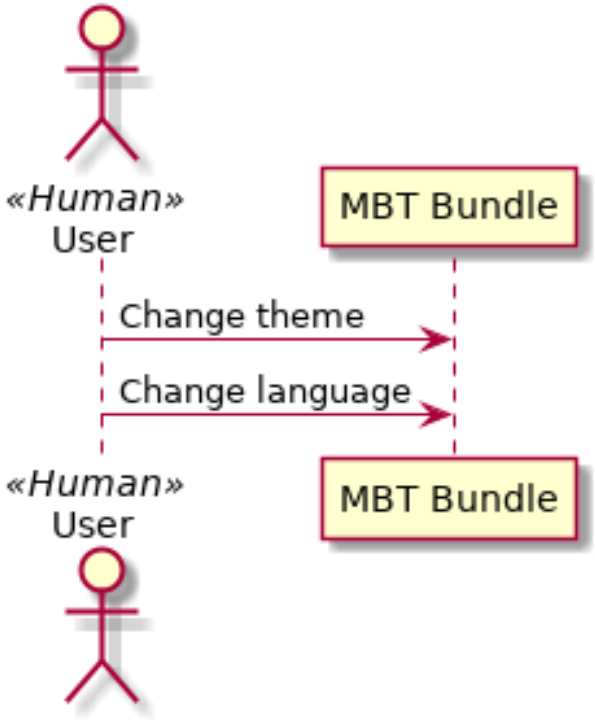


Figure 4.10 Configuration Sequence Diagram



4.2.7. Authentication

Log in, log out, register.

Figure 4.11 Authentication Use Case Diagram

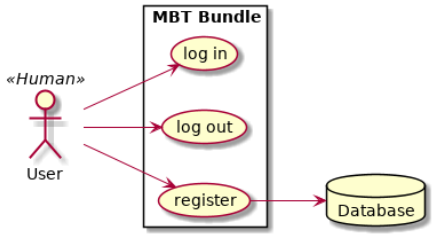
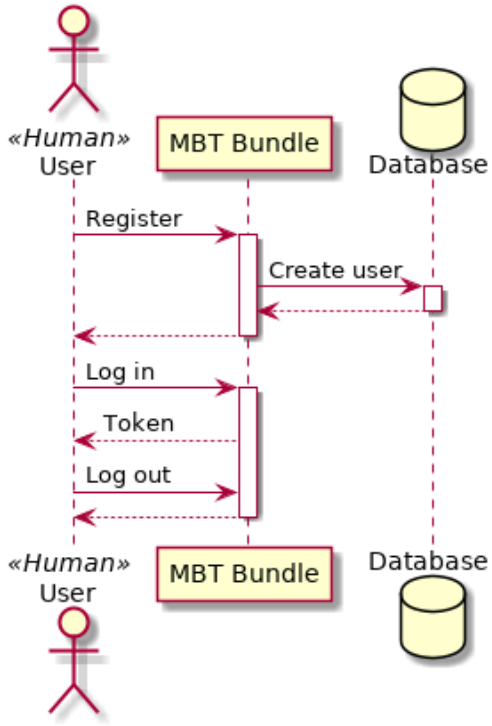


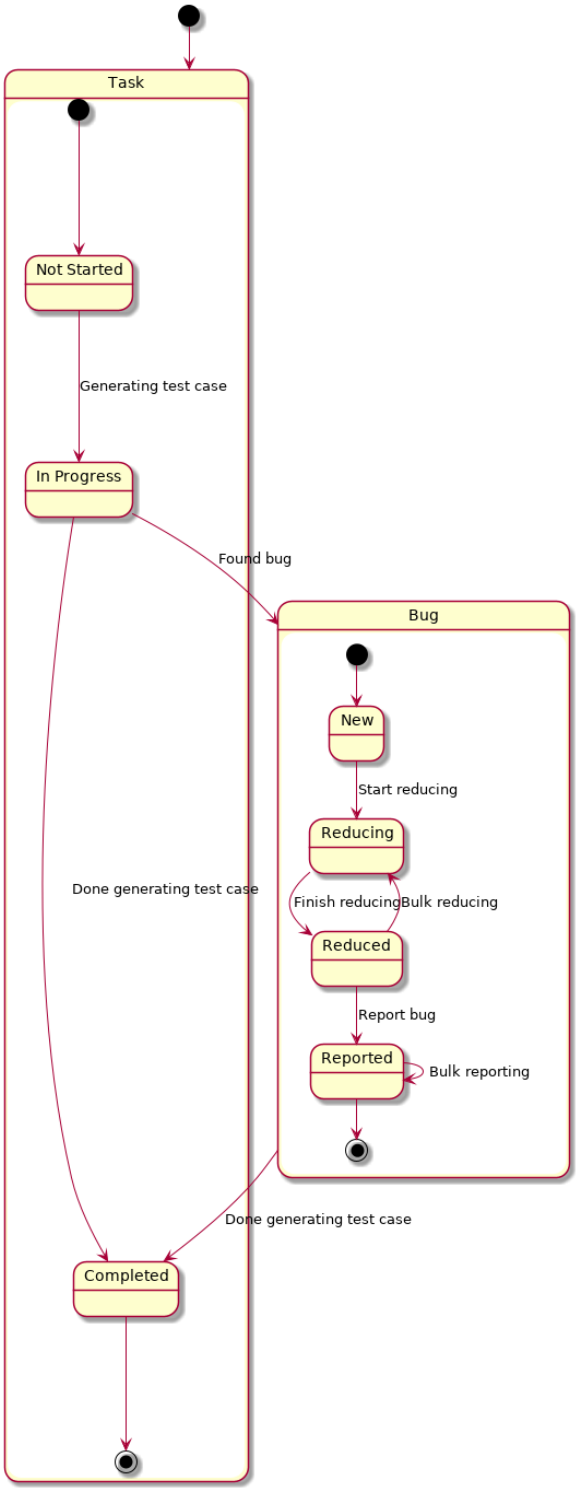
Figure 4.10 Authentication Sequence Diagram



5. Logical View

From creating task to reporting bug.

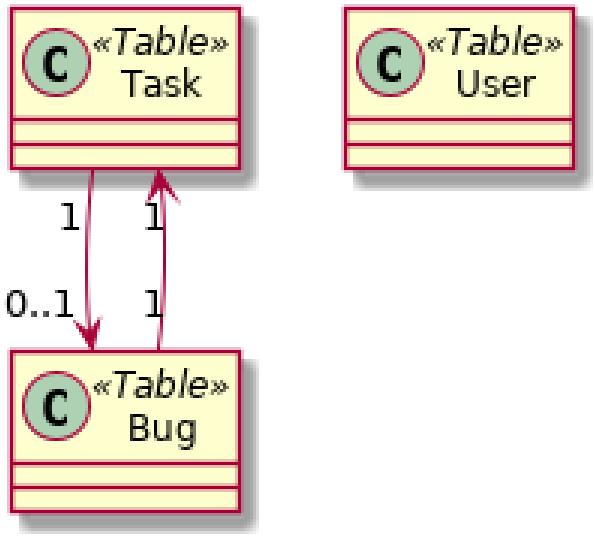
Figure 5.1 Task and Bug State Diagram.



6. Data View

Figure 6.1 Data Structure Diagram

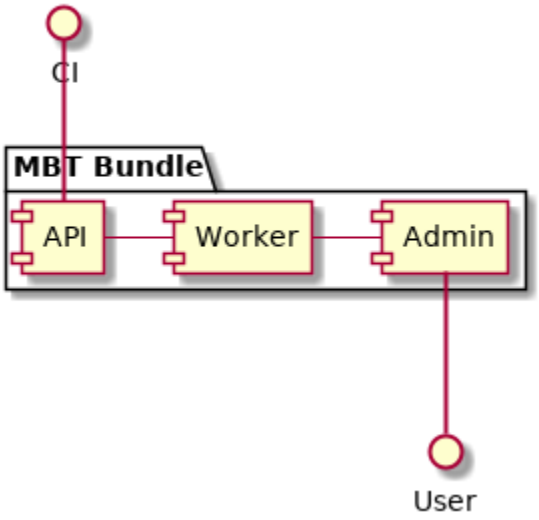
This diagram illustrates the static data structure and relationships of the main entities that will be stored by the application in its database. Each element nominally represents a database table. Relationship cardinality is denoted with UML multiplicity notation.



7. Implementation View

There are 3 basic components of the system.

Figure 7.1 Implementation View Diagram



8. Process View

Because of the limitation of PHP, every time worker received a new message, a new background process will be created to handle that message. There are multiple processes, but here are 2 most notable one:

Figure 8.1 Generator Activity Diagram

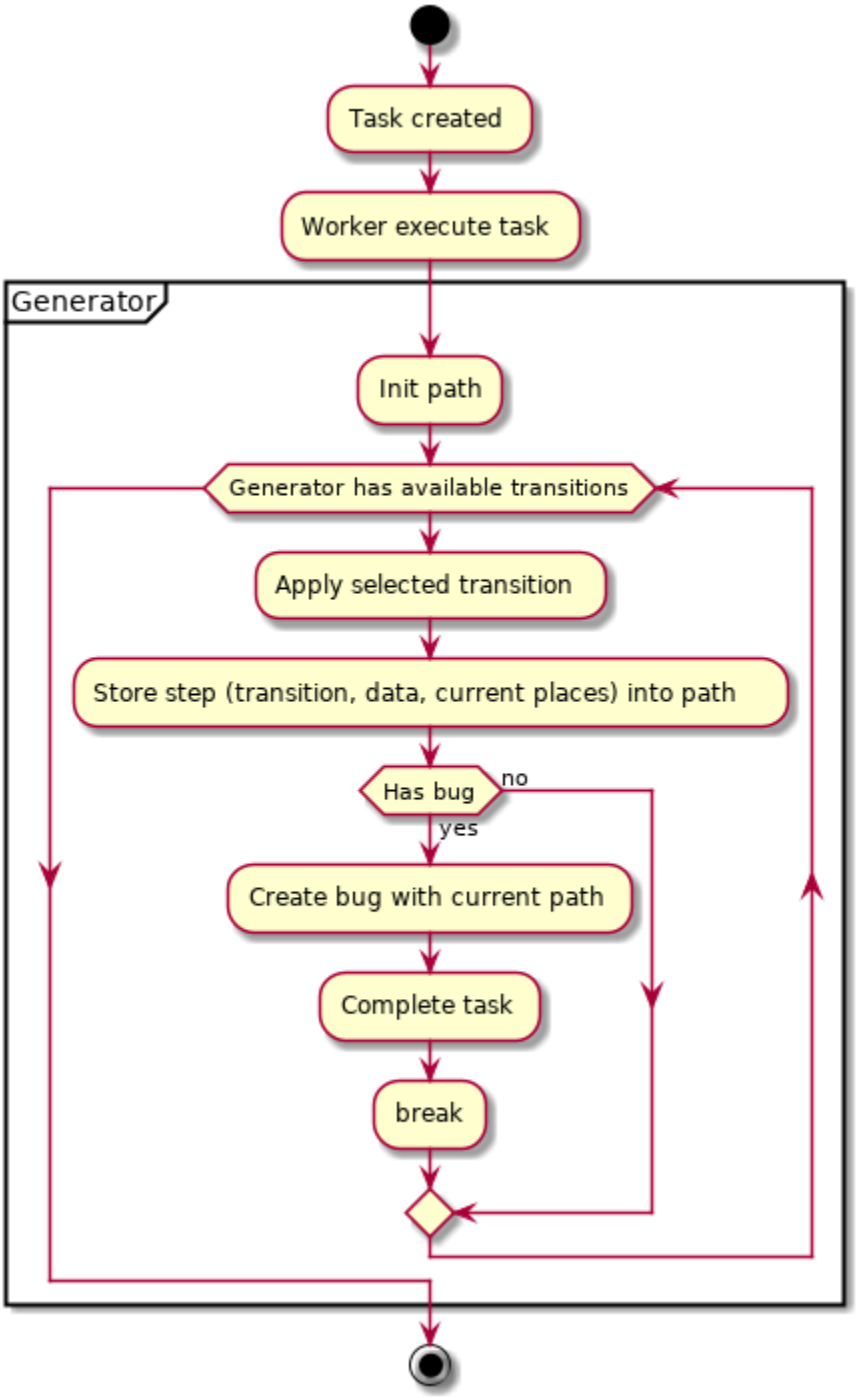
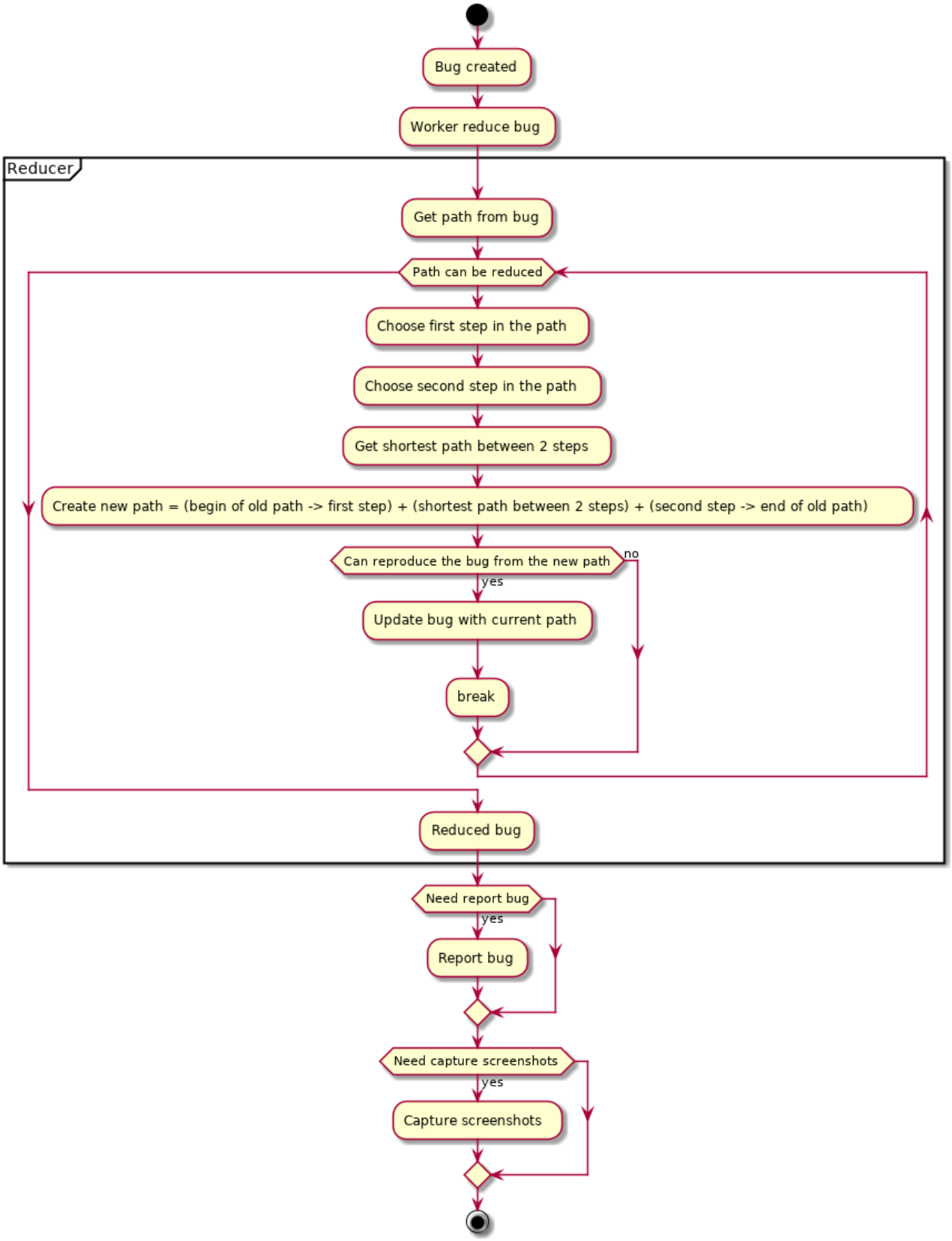


Figure 8.2 Reducer Activity Diagram



9. Deployment View

The system is split into multiple microservices. On local machine (development stage), we can use Docker Compose to put all microservices and dependencies into a single machine to see how it work. To deploy to multiple machines in order to scale, there are 2 easiest ways to do it: Docker Swarm and Compose On Kubernetes. Both tools support the docker-compose.yml file.

The application's deployment specifics can be seen below.

Figure 9.1 Deployment View Diagram