

BUS USER SUPPORT SYSTEM

Architecture Design

Project Code: BUSS

Document Code: BUSS\_ Architecture\_Design\_v1.0\_EN

**Ha Noi, 10/06/2015**

Record of change

\*A - Added M - Modified D - Deleted

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Effective Date** | **Changed Items** | **A\*M, D** | **Change Description** | **New Version** |
| 10/06/2015 | Add this document | A | Add | v1.0 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table 1: Record of change

SIGNATURE PAGE

**ORIGINATOR:** Nguyen Thanh Nam 10/06/2015

Technical Leader

**REVIEWERS:** Trinh Thi Tuyet Mai 16/06/2015

Test Leader

**APPROVAL:**  Nguyen Van Sang 19/06/2015

Supervisor

TABLE OF CONTENTS

[1 Introduction 5](#_Toc422213151)

[1.1 Purpose 5](#_Toc422213152)

[1.2 Scope 5](#_Toc422213153)

[1.3 Definitions, Acronyms and Abbreviations 5](#_Toc422213154)

[1.4 References 5](#_Toc422213155)

[1.5 Overview 6](#_Toc422213156)

[2 Choice of Architecture design 6](#_Toc422213157)

[2.1 MVC Model 6](#_Toc422213158)

[2.2 Codeigniter 8](#_Toc422213159)

[3 Architectural Representation 10](#_Toc422213160)

[4 Architectural Goals and Constraints 11](#_Toc422213161)

[5 Use-Case View 12](#_Toc422213162)

[5.1 BUSS User module 12](#_Toc422213163)

[5.1.1 Account Management Module 12](#_Toc422213164)

[5.1.2 Common module 13](#_Toc422213165)

[5.1.3 Bus module 14](#_Toc422213166)

[5.1.4 Path Finder Module 15](#_Toc422213167)

[5.1.5 Reminder module 16](#_Toc422213168)

[5.1.6 Track module 17](#_Toc422213169)

[5.1.7 Favorite module 17](#_Toc422213170)

[6 Logical View 18](#_Toc422213171)

[6.1 Overview 18](#_Toc422213172)

[6.2 Architecturally Significant Design Packages 19](#_Toc422213173)

[7 Process View 22](#_Toc422213174)

[8 Deployment view 24](#_Toc422213175)

[9 Quality 24](#_Toc422213176)

# Introduction

## Purpose

This document provides a comprehensive architectural overview of the system, 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 that have been made on the system.

## Scope

The scope of this document is to depict the architecture of the Bus User Support System for Android application and web service created by BUSS capstone project team.

## Definitions, Acronyms and Abbreviations

|  |  |  |
| --- | --- | --- |
| Acronym | Definition | Note |
| BUSS | Bus User Support System |  |
| BL | BusLover |  |
| MVC | Model view control |  |
| IDE | Integrated development environment |  |
| Q&A | Question and answer |  |
| GUI | Graphic user interface |  |

Table 2: Definitions

## References

* BUSS\_Software\_Requirements\_Specification\_v1.0\_EN.docx
* BUSS\_Data\_Design\_v1.0\_EN.docx
* Sample Design at Introduction to software engineering (I2SE) course in cms.fpt.edu.vn
* Software Architecture Design Illuminated Book
* <http://en.wikipedia.org/wiki/Model-view-controller>

## Overview

The Software Architecture Document contains the following subsections:

* Section 1: Provide an overview of entire Software Architecture Document.
* Section 2: Choice of Architecture Design
* Section 3: Architectural Representation
* Section 4: Architectural Goals and Constraints
* Section 5: Use-Case view
* Section 6: Logical View
* Section 7: Process View
* Section 8: Deployment view.
* Section 9: Quality.

# Choice of Architecture design

## MVC Model

The purpose of BUSS is developing an Android application that supports users about buses. The system of BUSS is structured based on Android (MVC) combined with layered architecture.

#### **MVC model overview**

MVC is a software architecture pattern that separates the representation of information from user interaction with it. The model consists of application data, business rules, logic and functions. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible, such as bar chart for management and a tabular view for accountants. The controller mediates input, converting it to commands for the model or view.

The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries. The collection of objects of a certain MVC type in an application sometimes referred to as a layer—for example, model layer.

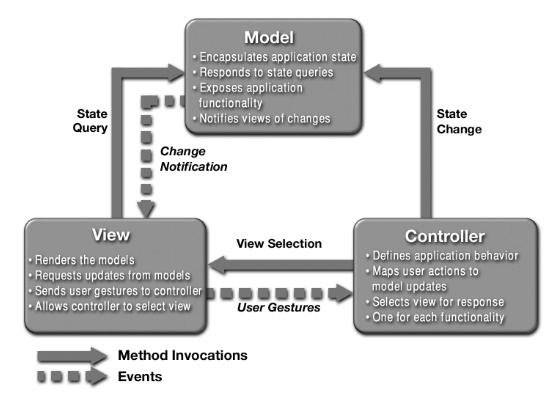


Figure 1: MVC Model

In addition to dividing the application into three kinds of components, the MVC design defines the interactions between them:

* **A controller:** can send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document). It can also send commands to the model to update the model's state (e.g., editing a document).
* **A model:** notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. A passive implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.
* **A view:** requests from the model the information that it needs to generate an output representation to the user.

#### **Advantages and disadvantages of MVC model**

* Advantages:
* The MVC model demonstrates professionalism in programming and design analysis. It is divided into independent components to help develop applications faster, simpler, easier upgrades and maintenance.
* Many MVC vendor framework tool kits are available.
* Multiple views synchronized with same data model.
* Easy to change or plug in new interface views, allowing updating of interface views with new technologies without overhauling the rest of system.

Very effective for deployment if graphic, programming and database development professionals are working in a team in a designed project.

* Disadvantages:
* For small projects that apply MVC model caused cumbersome, time consuming in development process.
* Time consuming to transits data between components.
* Not suitable for agent-oriented applications such as interactive mobile and robotics applications.
* Multiple pairs of controllers and views based on the same data model make data model change expensive.

The division between the View and the Controller is not clear in some cases.

#### **The reason of choosing MVC model**

* Easily manage the complexity of application by dividing the application into three components: model, view and controller.
* Better support for test-driven development.
* It is good support for application built by project team has many developers and designers but still managed application features.
* BUSS system is not complete system, now. We built the system that towards extensibility and maintainability in the future.

## Codeigniter

#### **Codeigniter overview**

CodeIgniter is an open source rapid development web application framework, for use in building dynamic web sites with PHP. CodeIgniter is loosely based on the popular MVC development pattern. While view and controller classes are a necessary part of development under CodeIgniter, models are optional.

#### **Advantages and disadvantages of Codeigniter**

Advantages:

* Designed according to the MVC Model
* CodeIgniter is a PHP framework have the fast speed
* CodeIgniter provides libraries cater to the most common tasks in web programming, such as database access, email, data checking, session management, image processing ... to the high function as XML-RPC, encryption, security ...
* Mechanism test data is closely, preventing XSS and SQL Injection of CodeIgniter helps minimize the security risk to the system.
* Disadvantages:
* No supports about some common module
* No supports about AJAX
* No support about **Event-Driven Programming**

#### **The reason of choosing Codeigniter**

* Consistent with UJD system
* Some modules need to reuse shared business logic layer.
* Designed according to the MVC Model

# Architectural Representation

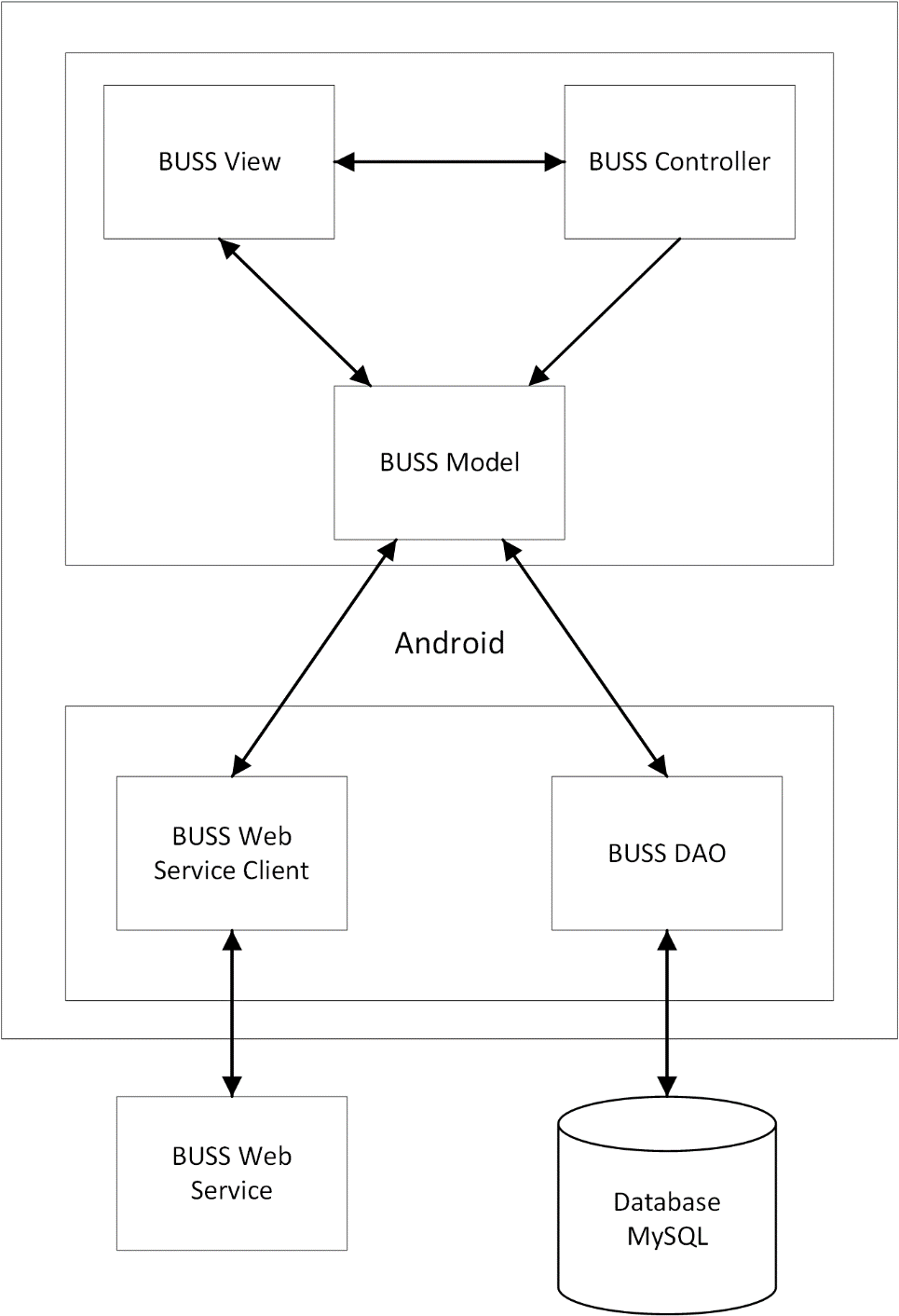


Figure 2: System overview

We follow MVC architecture for Android to implement the BUSS Project. MVC offers architectural benefits over standard - it helps us write better-organized and therefore more maintainable code.

**Model** is where the application’s data objects are stored. A model object is in charge of encapsulating application state, responding to state queries, exposes application functionality and notifying views of changes. When a model changes, typically it will notify its observers that a change has occurred. As with any data object it contains instance variables and getter/setter methods.

**View** is what is presented to the users and how users interact with the system. The view is expected to render the models in a meaningful way to the user, requests updates from models, send user gestures to controller and allow controller to select view.

**Controller** is the decision maker and the glue between the model and view; it handles user actions and gestures, responds to user events, define application behavior, map user actions to model updates and select view for response. For example, in BL, when a user clicks the “Add” button to add a new reminder, the controller for that action is invoked. The controller will then make changes to the contract model. The view will then render the modified contract model to the display so that user can view the new contract he added in the contract list.

# Architectural Goals and Constraints

**Availability:** The application must be available 95% of time. Users can access to it everywhere which has internet.

Maintainability:

* Coding standards and naming conventions
  + Output of the project must include coding standards and naming conventions documentations. Implementation code must be easy to maintain.
  + All code must be clearly commented, including class, method documentations.
  + If some components are reused, the documentations of those components must also be included.
* Design
  + The design of the system must be loosely coupled that chances on some module will not affect others.
* Logging
  + All the errors should be logged, supporting for bug fixing and maintenance.
  + All strange or sensitive situations should also be logged.

**Usability:** Usability Requirements support the following from the perspective of its primary users:

* *Efficiency of use***:** user can complete each function in less than 10 actions
* *Intuitiveness***:** all help/error messages are simple to understand; user can know exactly how to do each feature after one time using it.

**Capacity and scalability:** throughput, storage and growth requirements.

# Use-Case View

## BUSS User module

### **5.1.1 Account Management Module**



Figure 3: Account management module

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Log in | User | Log in to the application |
| 2 | Sign up | Guest | Create new account |
| 3 | Log out | User | Log out |
| 4 | Forgot password | User | Retrieve password |
| 5 | Change account info | User | Change password |

### **5.1.2 Common module**



Figure 4: Common module

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | Change app settings | User | Set up language, time and distances. |

### **Bus module**



Figure 5: Bus module

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View bus list | User | View a list of buses |
| 2 | Search bus in list | User | Search bus |
| 3 | View bus info | User | View information of a bus |
| 4 | View bus route | User | View route of a bus |
| 5 | View nearest bus | User | View the location of the nearest bus on the map |
| 6 | View bus map | User | View bus routes on map |
| 7 | View bus near me | User | View all the buses near your location |

### **5.1.4 Path Finder Module**



Figure 6: Path finder module

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | Find path | User | Search path from a place to another place |
| 2 | View paths found | User | View result after searching path |
| 3 | View path detail | User | View detail of this path |
| 4 | View direction | User | View direction of this path |

### **5.1.5 Reminder module**



Figure 7: Reminder module

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View bus reminder | User | View all reminder |
| 2 | Add reminder | User | Add new reminder |
| 3 | Edit reminder | User | Edit a reminder |
| 4 | Delete reminder | User | Delete a reminder |
| 5 | Suspend/ Unsuspend all reminder | User | Suspend/ Unsuspend all reminder |
| 6 | Suspend/ Unsuspend a chosen reminder | User | Suspend/ Unsuspend a chosen reminder |

### **5.1.6 Track module**



Figure 8: Track module

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | Set track | User | Set track of a bus |
| 2 | Cancel track | User | Cancel a track |

### **5.1.7 Favorite module**



Figure 10: Grammar management

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | Search grammar | Admin | Search grammar |
| 2 | Add grammar | Admin | Add new grammar |
| 3 | Edit grammar | Admin | Edit grammar |
| 4 | Delete grammar | Admin | Delete grammar |

# Logical View

## Overview

Logical View includes Package diagram and Class diagram. Package diagram describes the organization of packages and elements. Class Diagram provides an overview of the target system by describing the objects and classes inside the system and the relationships between them. It provides a wide variety of usages; from modeling the domain-specific data structure to detailed design of the target system

* Controller contain the interface between
* Associated models
* Associated views
* The input devices (e.g., keyboard, pointing device, time).
* Send commands to the model to update the model's state.
* Model is:
  + the domain-specific software simulation
  + Or implementation of the application's central structure.
* View deal with everything graphical
* Requests data from their model
* Display the data

## Architecturally Significant Design Packages

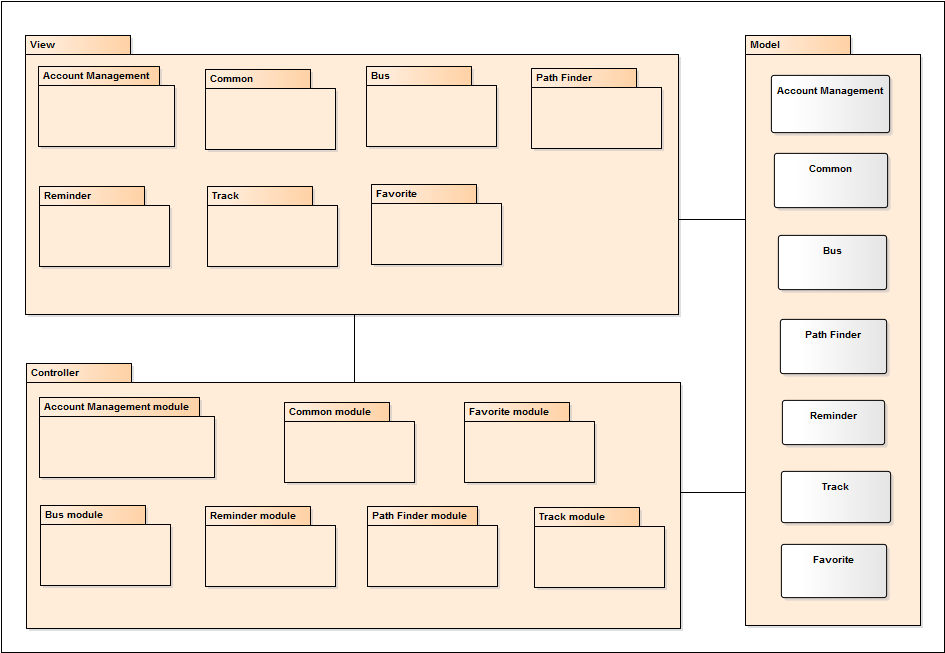


Figure 17: Package Diagram

* Model

|  |  |  |
| --- | --- | --- |
| **No.** | **Model class** | **Role** |
| 1 | Account Management | Description entity of Contact in database |
| 2 | Common | Description entity of Conversation in database |
| 3 | Bus | Description entity of Grammar in database |
| 4 | Path Finder | Description entity of Reading in database |
| 5 | Reminder | Description entity of Sentence in database |
| 6 | Track | Description entity of Test in database |
| 7 | Favorite | Description entity of User in database |

* Controller

|  |  |  |
| --- | --- | --- |
| **No.** | **Controller** | **Role** |
| 1 | Account Management module | * Receive request contact’s information * Handle request and call method in Account Management Model to get data from Database. * Respond data back to Account Management View. |
| 2 | Common module | * Receive request conversation’s information * Handle request and call method in Common Model to get data from Database. * Respond data back to Common View. |
| 3 | Bus module | * Receive request grammar’s information * Handle request and call method in Bus Model to get data from Database. * Respond data back to Bus View. |
| 4 | Path Finder module | * Receive request reading’s information * Handle request and call method in Path Finder Model to get data from Database. * Respond data back to Path Finder View. |
| 5 | Reminder module | * Receive request sentence’s information * Handle request and call method in Reminder Model to get data from Database. * Respond data back to Reminder View. |
| 6 | Track module | * Receive request test’s information. * Handle request and call method in Track Model to get data from Database. * Respond data back to Track View. |
| 7 | Favorite | * Receive request user’s information. * Handle request and call method in Favorite Model to get data from Database. * Respond data back to Favorite View. |

* View

Include many .xml files

# Process View



Figure 18: Sequence diagram

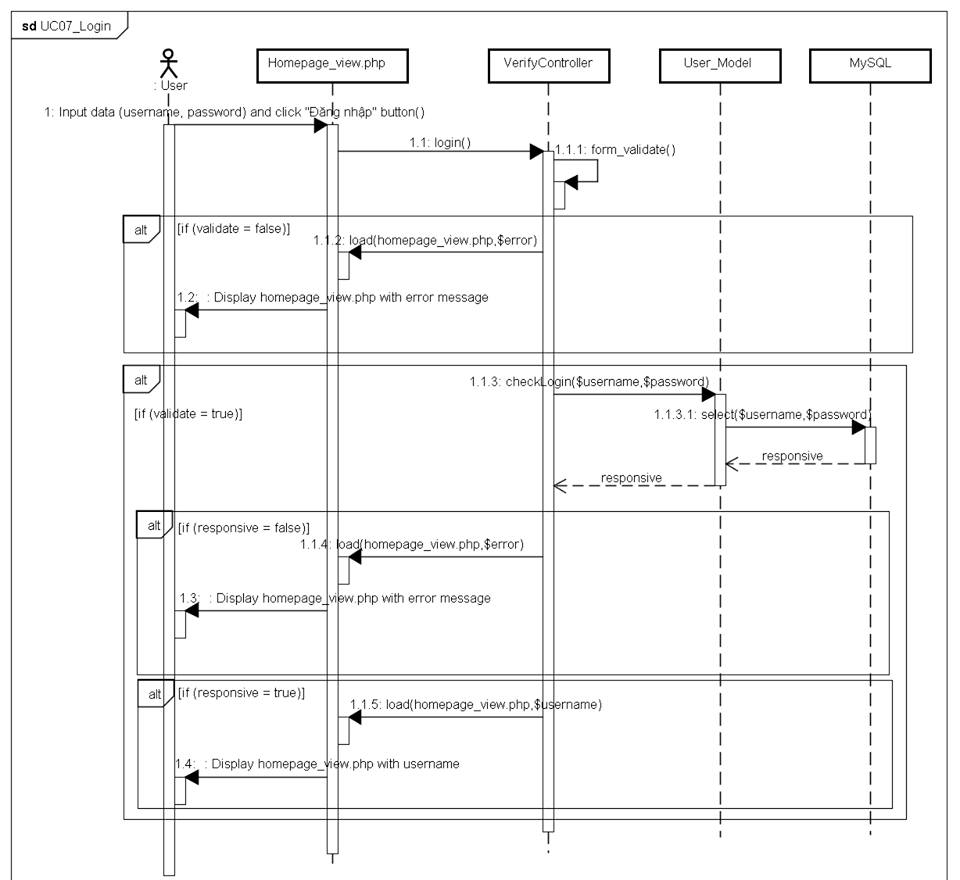


Figure 19: Login sequence

User input data(username, password) and click “Đăng nhập” button, user send a request login to verifyController class call function login()

If data entered validate is false, function login() will call function load(homepage\_view.php.$error) in verfifyController

If data enterd is validate is true, userModel will call function checkLogin($username,$password) in verifyController and send request select($username,$password) to MySQL. MySQL will response to ViewController

If responsive is false, function load(homepage\_view.php,$error) is called, display homepage\_view.php with error message to User

If responsive if tru, function load(homepage\_view.php,$username) is called, display homepage\_view.php with username

# Deployment view

Deployment view of website

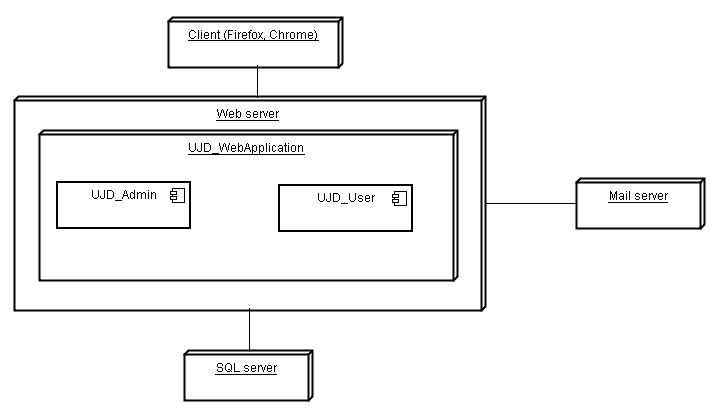


Figure 19: Deployment view

|  |  |  |
| --- | --- | --- |
| No. | Name | Description |
| 1 | MySQL sever | MySQL server use to store system’s data. MySQL5.6.16 or higher |
| 2 | Mail server | Mail server use to interact with user’s email |
| 3 | Web server | Web server is host system’s website. |
| 4 | Client | Client is web browser to use system. Firefox 20, Chrome 36 or higher. |

# Quality

Reference to: “BUSS\_Software\_Requirement\_Specification\_v1.0\_EN”