

BUS USER SUPPORT SYSTEM

Architecture Design

Project Code: BUSS

Document Code: BUSS\_Architecture\_Design\_v1.1\_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 |
| 20/08/2015 | Update this document | M | Update | v1.1 |
|  |  |  |  |  |
|  |  |  |  |  |

Table 1: Record of change

SIGNATURE PAGE

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

Technical Leader

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

Developer

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

Supervisor

TABLE OF CONTENTS

[1 Introduction 5](#_Toc428138945)

[1.1 Purpose 5](#_Toc428138946)

[1.2 Scope 5](#_Toc428138947)

[1.3 Definitions, Acronyms and Abbreviations 5](#_Toc428138948)

[1.4 References 5](#_Toc428138949)

[1.5 Overview 6](#_Toc428138950)

[2 Choice of Architecture design 6](#_Toc428138951)

[2.1 Layered architecture 6](#_Toc428138952)

[3 Architectural Representation 9](#_Toc428138953)

[4 Architectural Goals and Constraints 10](#_Toc428138954)

[5 Use-Case View 11](#_Toc428138955)

[5.1 BUSS User function group 11](#_Toc428138956)

[5.1.1 Account Management 11](#_Toc428138957)

[5.1.2 Common 12](#_Toc428138958)

[5.1.3 Bus 13](#_Toc428138959)

[5.1.4 Path Finder 14](#_Toc428138960)

[5.1.5 Reminder 15](#_Toc428138961)

[5.1.6 Track 16](#_Toc428138962)

[5.1.7 Favorites 16](#_Toc428138963)

[6 Logical View 17](#_Toc428138964)

[6.1 Overview 17](#_Toc428138965)

[6.2 Package diagram 18](#_Toc428138966)

[6.3 Class diagram 22](#_Toc428138967)

[7 Process View 24](#_Toc428138968)

[8 Deployment view 26](#_Toc428138969)

[9 Quality 27](#_Toc428138970)

# 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 - BusLover and web services 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 |  |
| aka | as known as |  |

Table 2: Definitions

## References

* BUSS\_Software\_Requirements\_Specification\_v1.1\_EN.docx
* BUSS\_Data\_Design\_v1.1\_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

## Layered architecture

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

#### **Layered architecture overview**

Layered architecture is a software architecture pattern that uses many layers for allocating the different responsibilities of a software product. It focuses on the grouping of related functionality within an application into distinct layers that are stacked vertically on top of each other. Functionality within each layer is related by a common role or responsibility. Communication between layers is explicit and loosely coupled. Layering your application appropriately helps to support a strong separation of concerns that, in turn, supports flexibility and maintainability.

The layers of an application may reside on the same physical computer (the same tier) or may be distributed over separate computers (n-tier), and the components in each layer communicate with components in other layers through well-defined interfaces. For example, a typical Web application design consists of a presentation layer (functionality related to the UI), a business layer (business rules processing), and a data layer (functionality related to data access, often almost entirely implemented using high-level data access frameworks).

However, the terms "tier" and "layer" are often used interchangeably. Most experts recognize a distinction between the two, where 'tier' is used when representing the physical layout of the various mechanisms in a system's infrastructure, while 'layer' is used when representing the orientation of the different physical *or conceptual* elements that make up an entire software solution. For example, a three-layer solution could easily be deployed on a single tier, such as a personal workstation. Easily, the term “tier” is for physical and the term “layer” is for logical.

Here we are going to consider about the layer architecture pattern.

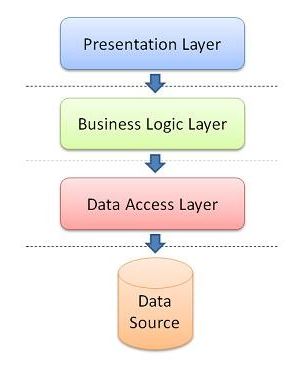


Figure 1: Layer architecture pattern

These layer are:

* **Presentation layer**: provides the application's user interface (UI).
* **Business layer**: provide the business logic and object as well as services.
* **Data layer:** provides access to external systems such as databases…

Beside it, we have **Common** package, which have common class for the whole system.

#### **Advantages and disadvantages of layer model**

* Advantages:
  + Layered architecture increases flexibility, maintainability, and scalability.
  + Multiple applications can reuse the components.
  + Layered architecture enables develop loosely coupled systems.
  + Different components of the application can be independently deployed, maintained, and updated, on different time schedules.
  + Layered architecture also helps you to test the components independently of each other.
* Disadvantages:
  + There might be a negative impact on the performance as we have the extra overhead of passing through layers instead of calling a component directly.
  + Development of user-intensive applications can sometime take longer if the layering prevents the use of user interface components that directly interact with the database.
  + The use of layers helps to control and encapsulate the complexity of large applications, but adds complexity to simple applications.
  + Changes to lower level interfaces tend to percolate to higher levels, especially if the relaxed layered approach is used.

#### **The reason of choosing 3 layers architecture pattern**

* Easily in develop and maintain system due to high flexibility, scalability and maintainability.
* Better support for test-driven development.
* It is good support for application built by project team has not too much developers thanks to the ease of reusability.
* BUSS system is not complete system, now. We built the system that towards extensibility and maintainability in the future.

# Architectural Representation

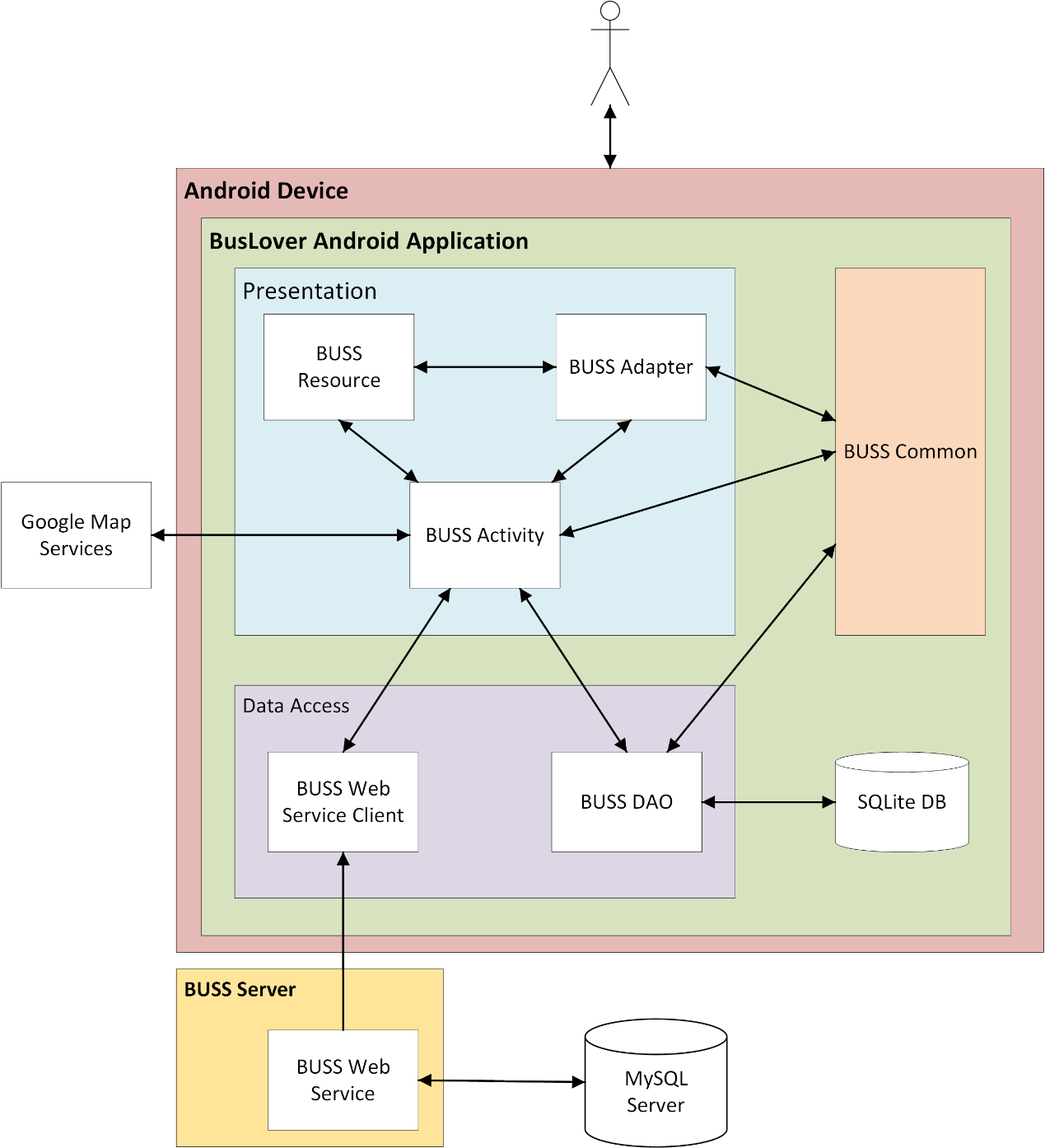


Figure 2: System overview

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

Presentation layer

* Contains activity, adapter and resources packages.
* Responsible for creating and displaying the UI and handling user interactions.
* Data shown is fetched from the Domain Model (Common package).

Data access layer

* Consists of two package: WebserviceClient and DAO.
* Responsible for querying database, calling a web service or send email…

Common package: which have common class for the whole system.

Webserver: deploy webservice and have connection to MySQL Server DB.

SQLite DB: local DB on device

Google Map Services: services of Google which were called by implement APIs.

# 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 function group

### **5.1.1 Account Management**



Figure 3: Account management

|  |  |  |  |
| --- | --- | --- | --- |
| 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**



Figure 4: Common

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

### **Bus**



Figure 5: Bus

|  |  |  |  |
| --- | --- | --- | --- |
| 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**



Figure 6: Path finder

|  |  |  |  |
| --- | --- | --- | --- |
| 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**



Figure 7: Reminder

|  |  |  |  |
| --- | --- | --- | --- |
| 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**



Figure 8: Track

|  |  |  |  |
| --- | --- | --- | --- |
| 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 Favorites**

Figure 9: Favorites

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 |  | User |  |
| 2 |  | User |  |
| 3 |  | User |  |
| 4 |  | User |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |

# Implementation View

## Overview

Implementation View includes Package diagram. Package diagram describes the organization of packages and elements.

## Package diagram

### **6.2.1 Android**

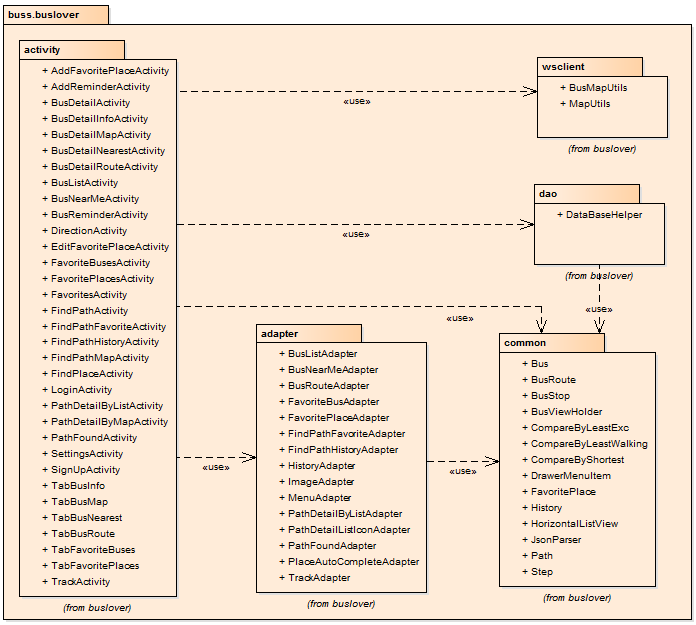


Figure 17: Package Diagram - Android

* app.java.buss.buslover.activity

|  |  |  |
| --- | --- | --- |
| No. | activity class | Role |
| 1 | AddFavoritePlaceActivity | Provide activity add favorite place |
| 2 | AddReminderActivity | Provide activity add reminder |
| 3 | BusDetailActivity | Provide activity bus detail |
| 4 | BusDetailInfoActivity | Provide activity bus detail tab info |
| 5 | BusDetailMapActivity | Provide activity bus detail tab map |
| 6 | BusDetailNearestActivity | Provide activity bus detail tab nearest |
| 7 | BusDetailRouteActivity | Provide activity bus detail tab route |
| 8 | BusListActivity | Provide activity bus list |
| 9 | BusNearMeActivity | Provide activity bus near me |
| 10 | DirectionActivity | Provide activity direction – navigation |
| 11 | EditFavoritePlaceActivity | Provide activity edit favorite place |
| 12 | FavoritesActivity | Provide activity favorites |
| 13 | FavoriteBusesActivity | Provide activity favorite tab bus |
| 14 | FavoritePlacesActivity | Provide activity favorite tab place |
| 15 | FindPathActivity | Provide activity find path |
| 16 | FindPathFavoriteActivity | Provide activity find path tab favorite |
| 17 | FindPathHistoryActivity | Provide activity find path tab history |
| 18 | FindPathMapActivity | Provide activity find path tab map |
| 19 | FindPlaceActivity | Provide activity find place |
| 20 | LoginActivity | Provide activity login |
| 21 | PathDetailByListActivity | Provide activity path detail by list |
| 22 | PathDetailByMapActivity | Provide activity path detail by map |
| 23 | PathFoundActivity | Provide activity path found |
| 24 | SettingsActivity | Provide activity settings |
| 25 | SignUpActivity | Provide activity sign up |
| 26 | TrackActivity | Provide activity track |

* app.java.buss.buslover.adapter

|  |  |  |
| --- | --- | --- |
| No. | adapter class | Role |
| 1 | BusListAdapter | Adapter for BusListActivity |
| 2 | BusNearMeAdapter | Adapter for BusNearMeActivity |
| 3 | BusRouteAdapter | Adapter for BusRouteActivity |
| 4 | FavoriteBusAdapter | Adapter for FavoriteBusesActivity |
| 5 | FavoritePlaceAdapter | Adapter for FavoritePlacesActivity |
| 6 | FindPathFavoriteAdapter | Adapter for FindPathFavoriteActivity |
| 7 | FindPathHistoryAdapter | Adapter for FindPathHistoryActivity |
| 8 | ImageAdapter | Adapter for ImageChooserDialog |
| 9 | MenuAdapter | Adapter for Menu |
| 10 | PlaceDetailByListAdapter | Adapter for PlaceDetailByList |
| 11 | PathFoundAdapter | Adapter for PathFoundActivity |
| 12 | PlaceAutoCompleteAdapter | Adapter for PlaceAutoCompleteTextView |
| 13 | TrackAdapter | Adapter for TrackActivity |

* app.java.buss.buslover.common

|  |  |  |
| --- | --- | --- |
| No. | common class | Role |
| 1 | Bus | Bus class |
| 2 | BusRoute | Bus route class |
| 3 | BusStop | Bus stop class |
| 4 | BusViewHolder | View holder class for bus |
| 5 | CompareByLeastExc | Compare by least execution class for path found |
| 6 | CompareByLeastWalking | Compare by least walking class for path found |
| 7 | CompareByShortest | Compare by shortest class for path found |
| 8 | DrawerMenuItem | Drawer class for menu item |
| 9 | FavoritePlace | Favorite place class |
| 10 | History | History class |
| 11 | HorizontalListView | Horizontal list view class |
| 12 | JsonParser | Json parser class |
| 13 | Path | Path class |
| 14 | Step | Step class |

* app.java.buss.buslover.dao

|  |  |  |
| --- | --- | --- |
| No. | common class | Role |
| 1 | DataBaseHelper | Database helper class |

* app.java.buss.buslover.wsclient

|  |  |  |
| --- | --- | --- |
| No. | common class | Role |
| 1 | BusMapServiceClient | Bus map service class |
| 2 | FindPathServiceClient | Find path service client |

### **6.2.2 Webservice**



Figure 18: Package Diagram – Web service

# Process View

## Login Activity diagram

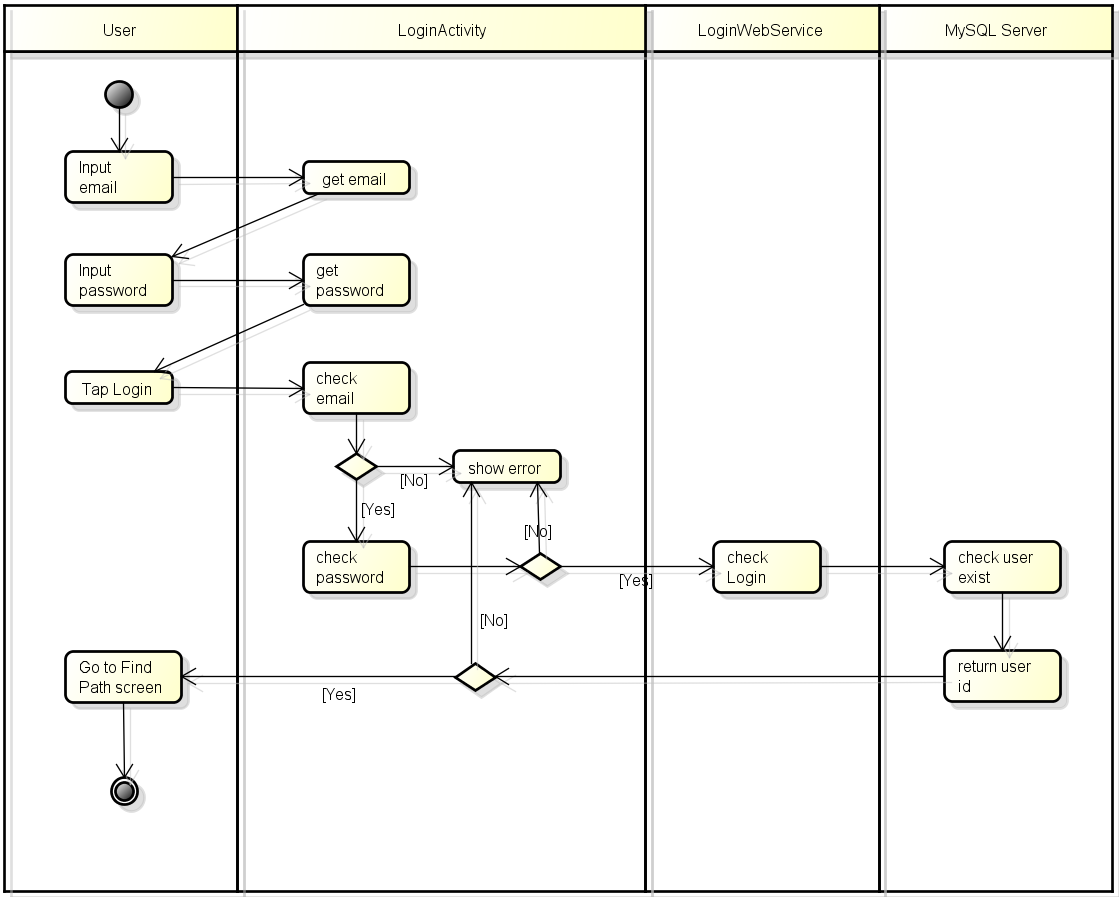


Figure 18: Activity diagram – Login

## Signup Activity diagram

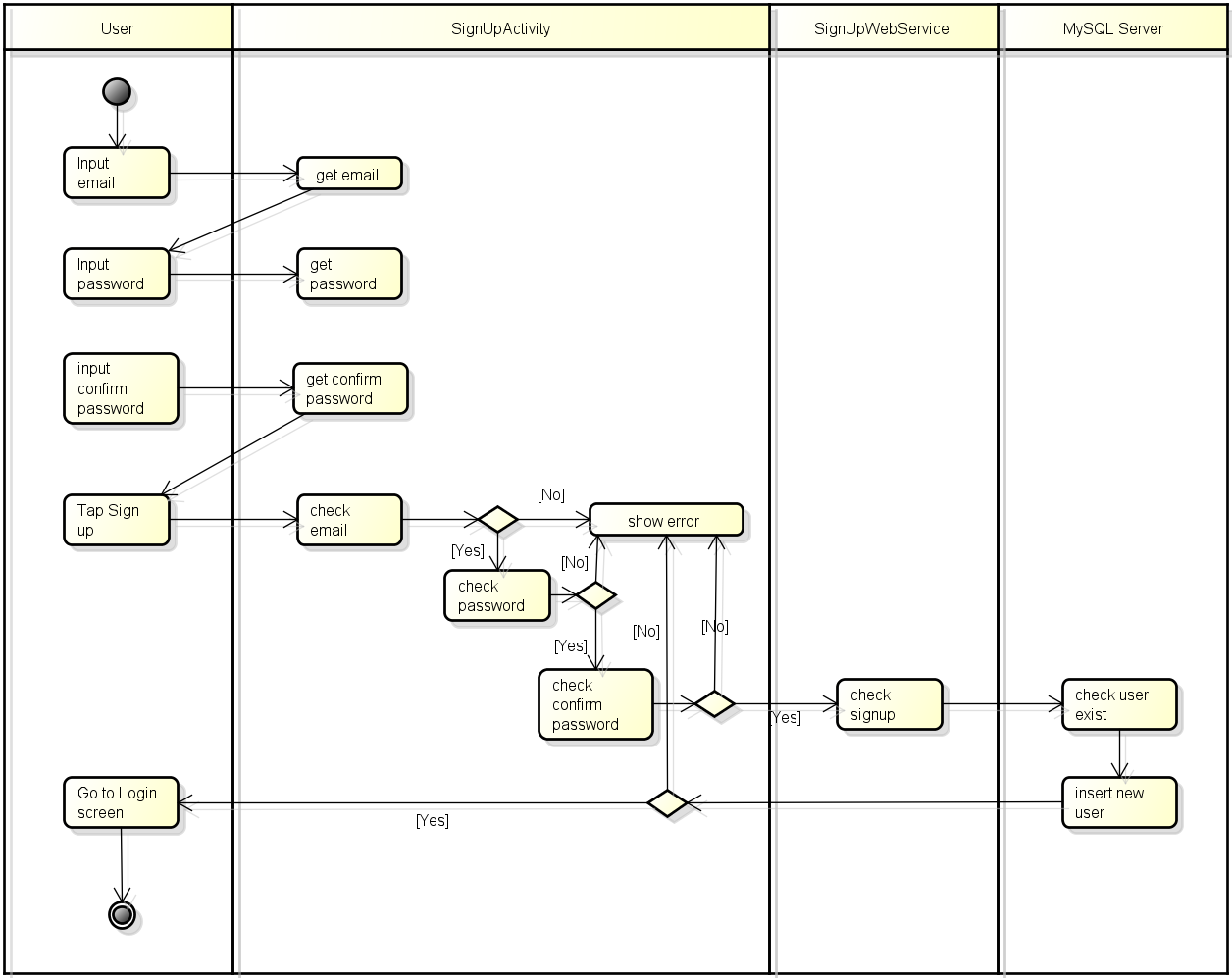


Figure 19: Activity diagram – Signup

**Log in:** User input data (email, password) and click “Sign in” button, LoginActivity class call function checkEmail(email) and checkPassword(password).

If data entered validate result is false, onClick() will show error. If data entered is true, onClick() will call Login async class to request webservice to log in.

Login webservice will request MySQL to select if any user with email and password which equal to entered data is exist.

If not any user exist, Login webservice will return 0, Login async class will display error message to User.

If user exist, Login webservice will return userID, Login async class will log in and display Find Path screen to User.

# Deployment view

Deployment view of BUSS

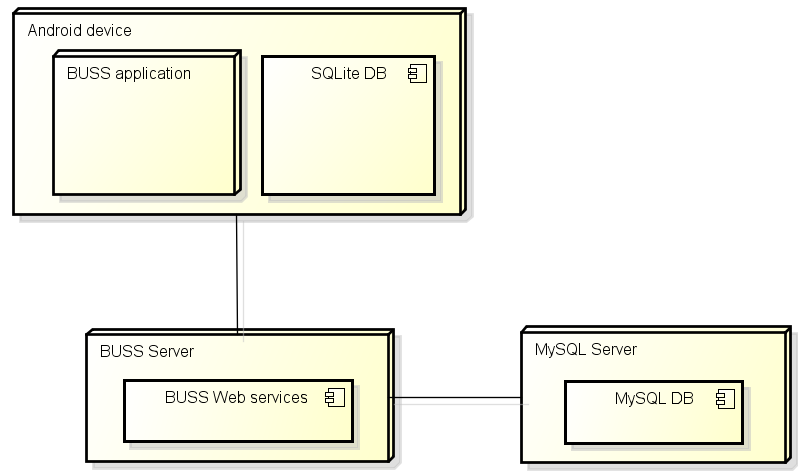


Figure 20: Deployment view

|  |  |  |
| --- | --- | --- |
| No. | Name | Description |
| 1 | Android device | Android device with Android 5.0+ |
| 3 | BUSS application | The BUSS android application – BL |
| 3 | SQLite DB | Local SQLite database on Android device |
| 5 | BUSS Web Server | Server for BUSS |
| 6 | BUSS Web Service | Web service for BUSS |
| 7 | MySQL sever | MySQL server use to store system’s data. MySQL5.6.16 or higher |
| 8 | MySQL DB | MySQL database which act as Server DB |

# Quality

Reference to: “BUSS\_Software\_Requirement\_Specification\_v1.1\_EN”