

# **Software Design Specification Document (CS360)**

## **Issue Logging System for Allied Bank Limited**



**Group Number: 24**

**Muhammad Raheem Zafar  
Aadam Nadeem  
Maleeha Masood  
Malik Ali Hussain  
Shahrukh Kemall**

**Course:** Software Engineering CS360

**Instructor:** Suleman Shahid

**University:** Lahore University of Management  
Sciences (LUMS)

**Version:** 1.0

**Date:** (05/04/2020)

**Number of hours spent on this document:** 128

## Contents

1	Change Log.....	3
1.1	Project Scope.....	3
1.2	Change log.....	3
	Functional Requirements .....	3
	Use Case Description:.....	14
2	Introduction.....	18
2.1	Document Purpose .....	18
2.2	Product Scope .....	18
2.3	Intended Audience and Document Overview .....	18
2.4	Definitions, Acronyms and Abbreviations.....	19
2.5	References and Acknowledgments.....	20
3	Overall Description.....	21
3.1	System overview .....	21
3.2	System constraints.....	24
3.3	Architectural strategies.....	24
4	System Architecture.....	26
4.1	System Architecture.....	26
4.2	Subsystem Architecture .....	30
4.3	Database Model .....	35
4.3.1	Database scheme and detailed description.....	35
4.3.2	Database .....	37
4.4	External Interface Requirements .....	37
4.4.1	User Interfaces .....	37
4.4.2	Hardware Interfaces.....	38
5	User Interface Design.....	39
5.1	Description of the user interface .....	39
5.2	Information architecture.....	40
5.3	Screens.....	41
5.4	User interface design rules.....	60
6	Other Non-functional Requirements.....	62
6.1	Performance Requirements.....	62
6.2	Safety and Security Requirements.....	62
6.3	Software Quality Attributes .....	63

# 1 Change Log

## 1.1 Project Scope

We don't believe that we require any changes in terms of our project scope.

## 1.2 Change log

The course staff had pointed out some serious errors with regards to our formatting of the functional requirements i.e. section 3.1 and, use case description section 3.3.3 of the SRS. Below is our attempt to rectify these errors.

### Functional Requirements

<Category: Login System >

#### RQ<1> - Customer Functional Requirement 1

- **Description**

The system should ask every user to input their usernames and password. It should be able to accomodate 5 different roles: Initiator, Support, Manager, Monitor and Admin.

- **Input**

User login details (username and password) entered.

- **Processing**

Authentication: Once the username and password has been entered, the system should compare them to the already provided credentials of the employee, as those stored in a dummy database, and figure out what kind of role is logging in by querying the database.

- **Output**

User logged in and the role specific options screen is displayed if the passwords match. Otherwise, log in screen showed again.

#### RQ<2> - Customer Functional Requirement 2

- **Description**

The system should allow a logged in user to log out.

- **Input**

Logged in user clicks on the log out button.

- **Processing**

System logs user out.

- **Output**

Log in screen that asks for user credentials is displayed.

**<Category: User's Role Specific Cases >**

**RQ<3> - Customer Functional Requirement 3**

- **Description**

Every role should be able to log one's own support related issues through the ILS app.

- **Input**

Select the option on the screen, displayed after successful login, to fill the form that describes the technical issue and gives the option to enter text to elaborate on the concern. If time allows the developers, field for attachments that support jpeg, pdf and other commonly used formats will also be present in the form.

- **Processing**

Checks if all fields are filled and form is complete.

- **Output**

If the form complete and accurately filled, the entered log is uploaded to the central database. Otherwise, an error message should be displayed.

**RQ<4> - Customer Functional Requirement 4**

- **Description**

Every role should be able to log other's support related issues through the ILS app on behalf of them. The system should be able to track original users by Name and staff ID.

- **Input**

Select the option on the screen displayed after successful login, to fill the form that describes the technical issue and gives the option to enter text to elaborate on the concern. User selects on behalf option also and enters the relevant personnel's Name and staff ID.

- **Processing**

Checks if all fields are filled and form is complete.

- **Output**

If form complete and accurate, the entered log is uploaded to the central database. Otherwise, an error message should be displayed.

**RQ<5> - Customer Functional Requirement 5**

- **Description**

Every role should be able to view the status of one's own issues through the ILS app.

- **Input**

Select the option on the screen displayed after successful login, to view status of one's own logged issues.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Retrieved logs displayed on the screen.

**RQ<6> - Customer Functional Requirement 6**

- **Description**

Every role should be able to finally close a fixed request that they have logged or open it again.

- **Input**

Select the option on the screen displayed after successful login, to view status of one's own logged issues.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Select the option to close or reopen a fixed issue from the list of retrieved issues displayed on the screen.

**RQ<7> - Customer Functional Requirement 7**

- **Description**

Allow support role users to view issues of in-city branches matching their work category.

- **Input**

Select the option on the screen displayed after successful login for support role, to view other in-city branches' issues that match their work category.

- **Processing**

System retrieves issues of in-city logged in requests and their status from the database.

- **Output**

Retrieved issues displayed on the screen.

**RQ<8> - Customer Functional Requirement 8**

- **Description**

Allow support role users to assign open issues to themselves.

- **Input**

Select the option on the screen displayed after successful login for support role, to view other people's issues that match their work category.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Retrieved issues displayed on the screen. Select the option to assign open issues to yourself.

**RQ<9> - Customer Functional Requirement 9**

- **Description**

Allow support role users to mark taken up issues as fixed.

- **Input**

Select the option on the screen displayed after successful login for support role, to view one's taken up issues.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Select the option to assign fixed status to the issues.

**RQ<10> - Customer Functional Requirement 10**

- **Description**

Allow support role users to mark taken up issues as open.

- **Input**

Select the option on the screen displayed after successful login for support role, to view one's taken up issues.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Select the option to assign open status to the issues taken up.

**RQ<11> - Customer Functional Requirement 11**

- **Description**

Allow support role users to assign their taken up issues to other team members.

- **Input**

Select the option on the screen displayed after successful login for support role, to view taken up issues by the user.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Retrieved issues displayed on the screen. Select the option to assign issues to other team members.

**RQ<12> - Customer Functional Requirement 12**

- **Description**

Allow manager role users to assign open issues to themselves.

- **Input**

Select the option on the screen displayed after successful login for manager role, to view other people's issues that match their work category.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Retrieved issues displayed on the screen. Select the option to assign open issues to yourself.

**RQ<13> - Customer Functional Requirement 13**

- **Description**

Allow manager role users to view issues taken up by their team members.

- **Input**

Select the option on the screen displayed after successful login for manager role, to view team member's taken up issues.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Retrieved issues displayed on the screen.

**RQ<14> - Customer Functional Requirement 14**

- **Description**

Allow manager role users to mark taken up issues as fixed.

- **Input**

Select the option on the screen displayed after successful login for manager role, to view one's taken up issues.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Select the option to assign fixed status to the issues.

**RQ<15> - Customer Functional Requirement 15**

- **Description**

Allow manager role users to mark taken up issues as open.

- **Input**

Select the option on the screen displayed after successful login for manager role, to view one's taken up issues.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Select the option to assign open status to the issues taken up.

**RQ<16> - Customer Functional Requirement 16**

- **Description**

Allow manager role users to assign their taken up issues to other support teams.

- **Input**

Select the option on the screen displayed after successful login for manager role, to view taken up issues by the support team.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Retrieved issues displayed on the screen. Select the option to assign issues to another support team.



**RQ<17> - Customer Functional Requirement 17**

- **Description**

Allow manager role users to reassign issues to other team members.

- **Input**

Select the option on the screen displayed after successful login for manager role, to view taken up issues by the user.

- **Processing**

System retrieves issues and their status from the database.

- **Output**

Retrieved issues displayed on the screen. Select the option to assign issues to other team members.

**RQ<18> - Customer Functional Requirement 18**

- **Description**

Allow manager role users to monitor reports based on total number of requests against Area/ Category/ Sub-category/ Location.

- **Input**

Select the option on the screen displayed after successful login for manager role, to view reports generated by the system.

- **Processing**

System generates reports.

- **Output**

Reports displayed on the screen.

**RQ<19> - Customer Functional Requirement 19**

- **Description**

Allow monitor role users to monitor reports based on total number of requests against Area/ Category/ Sub-category/ Location.

- **Input**

Select the option on the screen displayed after successful login for monitor role only, to view reports generated by the system.

- **Processing**

System generates reports.

- **Output**

Reports displayed on the screen.

**RQ<20> - Customer Functional Requirement 20**

- **Description**

Allow admin role users to change roles of users.

- **Input**

Select the option on the screen displayed after successful login for admin role, to change the role of users. Enter the name and employee ID of the user and select the new role.

- **Processing**

Query generated to update databases.

- **Output**

Database now stores the updated role if the name and employee ID are valid. Otherwise, an error message.

**RQ<21> - Customer Functional Requirement 21**

- **Description**

Allow admin role users to change team of users.

- **Input**

Select the option on the screen displayed after successful login for admin role, to change the team of users. Enter the name and employee ID of the user and select the new team of the user.

- **Processing**

Query generated to update databases.

- **Output**

Database now stores the updated team if the name and employee ID are valid. Otherwise, an error message.

**<Category: Emails and Notifications>****RQ<22> - Customer Functional Requirement 22**

- **Description**

Email and push notifications should be sent out to the initiator and CC'd to the control support team when a new issue is logged.

- **Input**

User logs an issue.

- **Processing**

Issue logged successfully to the database.

- **Output**

Automatically generated email and push notification sent to the accounts/ devices of initiator and CC'd to the control support team.

**RQ<23> - Customer Functional Requirement 23**

- **Description**

Email and push notifications should be sent out to the initiator when an issue is fixed.

- **Input**

Support team member marks an issue as fixed.

- **Processing**

Issue's status updated in the database.

- **Output**

Automatically generated email and push notification sent to the accounts/ devices of initiator.

**RQ<24> - Customer Functional Requirement 24**

- **Description**

Email and push notifications should be sent out to the initiator and CC'd to the support team member who fixed the issue when an issue is closed.

- **Input**

Issue marked as closed by a support team member or the system.

- **Processing**

Issue's status updated in the database.

- **Output**

Automatically generated email and push notification sent to the accounts/ devices of initiator and CC'd to the support team member who fixed Issue.

**RQ<25> - Customer Functional Requirement 25**

- **Description**

Email and push notifications should be sent out to the initiator and CC'd to the respective manager and concerned support team members when an issue is referred to another team.

- **Input**

Issue referred to another team.

- **Processing**

Issue's information updated in the database.

- **Output**

Automatically generated email and push notification sent to the accounts/ devices of initiator and CC'd to the respective manager and the concerned support team members.

**RQ<26> - Customer Functional Requirement 26**

- **Description**

Email and push notifications should be sent out to the initiator and CC'd to the respective manager and new resolver when an issue is referred to another team member.

- **Input**

Issue referred to another team member.

- **Processing**

Issue's information updated in the database.

- **Output**

Automatically generated email and push notification sent to the accounts/ devices of initiator and CC'd to the respective manager and the new resolver.

**RQ<27> - Customer Functional Requirement 27**

- **Description**

Email and push notifications should be sent out to the concerned support team members when an issue is re-sent to a support team.

- **Input**

Issue re-sent to a support team.

- **Processing**

Issue's information updated in the database.

- **Output**

Automatically generated email and push notification sent to the accounts/ devices of concerned support team members.

**RQ<28> - Customer Functional Requirement 28**

- **Description**

Emails and push notifications should be sent out to the concerned support team members when the initiator does not accepted fixed issue.

- **Input**

Initiator marks fixed issue as open.

- **Processing**

Issue's information updated in the database.

- **Output**

Automatically generated email and push notification sent to the accounts/ devices of concerned support team members.

**<Category: Automatic Functioning of the System>**

**RQ<29> - Customer Functional Requirement 29**

- **Description**

The system should auto close completed issues: on a daily basis, a service should be executed to close the issues automatically if they remained with fixed status for consecutive 2 days.

- **Input**

48 hour timer completed.

- **Processing**

Query generated to change the issue's status to closed.

- **Output**

Database updated to change the fixed issues status to closed.

**RQ<30> - Customer Functional Requirement 30**

- **Description**

System should be able to fetch new updates to the logs and status of issues automatically.

- **Input**

User viewing issues logged and timer completed.

- **Processing**

Logs retrieved from database.

- **Output**

Updated list of issues displayed.

**RQ<31> - Customer Functional Requirement 31**

- **Description**

The system should automatically escalate an Issue to the manager, if an issue remains unattended or unresolved for 48 hours.

- **Input**

Timer completed.

- **Processing**

Query generated to update the relevant logs in the database.

- **Output**

Database updated with issue assigned to manager.

## Use Case Description:

### Use Case 1:

Title	Login to relevant window
Actors	initiator, support, manager, monitor, admin
Preconditions	An active connection to the internet network
Postconditions	Successful login to relevant window.
Basic flow	The actor enters the credentials given to them by the bank that are already in the database. Upon successful authentication, the actor will be redirected to the role-specific window.
Alternative flow	If the actor enters incorrect credentials, a login-error message appears.

### Use Case 2:

Title	Issue Logging
Actors	initiator, support, manager, monitor, admin
Preconditions	The actor has authentic login credentials.
Postconditions	An issue complaint is registered in the system
Basic flow	The actor logs in and goes to the "Log new issue" window and selects the area, category and sub-category of the relevant issue and adds a brief description in the text box. The actor then clicks on the 'submit' button. Upon submitting, an email will be generated to the initiator and CC'd to the support team.

**Use Case 3:**

Title	View and update status of issue
Actors	initiator, support, manager, monitor, admin
Preconditions	An issue is logged in the system by the respective actor.
Postconditions	Knowledge of the issue (if it's open or closed)
Basic flow	The actor logs in and goes to the "View issue status" window where he locates his issue (if there exist multiple). The status of issue is visible next to the issue title. The status can be 'open': not resolved yet, 'fixed': resolved by the support team member, upon which an email will be generated to the users Software Requirements Specification for ILS-ABL Page 17 specified in 3.1, or 'closed': meaning the issue has been resolved by the support team and has been closed by the initiator or auto-closed by the system after 2 days of 'fixed' status. If the actor is satisfied with the resolution of the issue, they can close the issue. After the issue is 'closed', an email will be sent to the initiator and the support team member who resolved it
Alternative flow	The actor proceeds with the same steps in basic flow. The status is 'fixed' but if the user is not satisfied with the resolution, then the issue will be re-sent to the support team and an email will be sent to the support team.

**Use Case 4:**

Title	Assign Open Issues
Actors	Support Team, Manager
Preconditions	There exist open issues in the system.
Postconditions	The actor has assigned open issues to relevant personnel.
Basic flow	The actor locates the list of issues and identifies the open ones. For both actors, they can assign issues to themselves or relevant support team members.

**Use Case 5:**

Title	Changing Roles
Actors	Admin
Preconditions	At least one employee exists in the system
Postconditions	The employee's role has been changed
Basic flow	The actor chooses an employee whose role has to be changed. The roles can be switched to any of the other roles that the employee isn't currently.

**Use Case Table:**

Primary Actor	Associated Use cases
Initiator	<ol style="list-style-type: none"><li>1. Login using credentials</li><li>2. Log support related issues</li><li>3. View status of own issue</li><li>4. Mark own issue as closed or open it again</li><li>5. Logout from system</li></ol>
Support	<ol style="list-style-type: none"><li>1. Login using credentials</li><li>2. Log support related issues</li><li>3. View status of own issue</li><li>4. Mark own issue as closed or open it again</li><li>5. View issues of in-city branches matching their work category.</li><li>6. Assign open issues to themselves.</li><li>7. Mark resolved issues as fixed.</li><li>8. Reopen resolved issues.</li><li>9. Reassign issues to other team members.</li><li>10. Logout from system</li></ol>
Manager	<ol style="list-style-type: none"><li>1. Login using credentials</li><li>2. Log support related issues</li><li>3. View status of own issue</li><li>4. Mark own issue as closed or open it again</li><li>5. Assign open issues to themselves.</li><li>6. View Status of Issues pending with their team.</li><li>7. Mark resolved issues as fixed.</li><li>8. Refer Issue to other Support Team</li><li>9. Assign/Re-assign Issue to other team members.</li><li>10. Monitor reports</li><li>11. Logout from system</li></ol>
Monitor	<ol style="list-style-type: none"><li>1. Login using credentials</li><li>2. Log support related issues</li><li>3. View status of own issue</li><li>4. Mark own issue as closed or open it again</li><li>5. Monitor reports</li><li>6. Logout from system</li></ol>



Admin	<ol style="list-style-type: none"><li>1. Login using credentials</li><li>2. Log support related issues</li><li>3. View status of own issue</li><li>4. Mark own issue as closed or open it again</li><li>5. Change Role</li><li>6. Change Team</li><li>7. Logout from system</li></ol>
-------	---

## **2 Introduction**

### **2.1 Document Purpose**

The purpose of this document is to describe in depth the design specifications and the architecture of the first prototype of the Issue Logging System Mobile Application that we are developing for our client, Allied Bank Limited. It is the first version of a Software Designs Specification Document for the product.

This document aims to help its reader understand the application being developed from a design perspective. It splits the complete product (ILS app) into its subsystems (front end and its components and the backend working system) and describes each so that the reader can have an informative as well as a pictorial representation of the system being developed. The document also shows how the use cases described in the SRS will be mapped in the system using the suggested design plan.

### **2.2 Product Scope**

The Issue Logging System Application acts as a platform for all the employees of Allied Bank Limited (ABL). Using the ILS app, employees at ABL can report technical concerns, and the relevant support team, technologically educated employees at ABL, can choose to resolve these concerns.

This application will help in more efficient resolution of technical problems since it aims to be an application that is easy to use on the go: the ILS app aims to be a cross platform application that can easily be used on both iOS and Android Mobile Devices by all the relevant personnel involved. This application also facilitates the swift communication between the people involved, aiming to remove any communication related hindrances from the process of resolving a technical issue. This is done by ensuring that any communication between the employees is done to the point - there is not much room for extra conversation.

### **2.3 Intended Audience and Document Overview**

The intended user group of this document is the IT department of our client, as well as Lahore University of Management Science course CS360's course staff.

The beginning of this document consists of the changes made to some information as specified in the Software Requirements Specification Document, followed by a short guide about the document. The rest of this document starts with an overview of the system designs and constraints encountered, followed by three sections that in depth explain the product design. Section 5 prototypes the user interface of the application in the form of a series of images and diagrams. Section 4 describes the system and the database designs. Section 6 focuses on mapping non-functional requirements onto the design proposed.

For all types of reader, please ensure that section 2 is well read.

If you are from ABL, sections 1, 3 and the appendices can be skipped since the former is information already discussed with you and the latter is just an overview, and the contents are described in depth later on. All other sections should be read thoroughly in the order of sections 5, 4 and 6.

If you are CS360's course staff, you should read all the sections starting from section 1, followed by 5, 3, 4 and 6. The appendices can be read at your convenience.

Going through section 5 first might assist in understanding the document since people generally understand pictorial references better than textual.

## **2.4 Definitions, Acronyms and Abbreviations**

- ☐ ABL: Allied Bank Limited.
- ☐ Admin: Designated employees that have the administration role in the ILS system.
- ☐ Android Studio: The official integrated development environment for Google's Android operating system.
- ☐ API: Application Program Interface
- ☐ Area: The broader category of a problem. For example: IT is the area of the Hardware category.
- ☐ Attack: an attempt by an individual with a malicious intent to either steal sensitive information or cause mischief to the victim.
- ☐ Category: The category to which an issue belongs to. For example: Hardware is the category of all issues relevant to monitor or a computer mouse etcetera.
- ☐ Caching: a technique used to store a file or any data locally for further use to avoid fetching from a server or a file at a location away from the host.
- ☐ CC: A carbon copy, or "Cc'd" message is an e-mail that is copied to one or more recipients. Both the main recipient (whose address is in the "To:" field) and the Cc'd recipients can see all the addresses the message was sent to.
- ☐ Close issue: The final confirmation of resolution of the issue, done by the initiator, after the support role employee resolves the technical problem.
- ☐ Dart: Programming Language used in Flutter.
- ☐ Employee: An educated individual hired by ABL.
- ☐ Fix issue: When a support team member resolves an issue.
- ☐ Flutter: One of the front end development tools.
- ☐ GPS: Global Positioning System.
- ☐ GUI: Graphical User Interface.
- ☐ IDE: Integrated Development Environment.
- ☐ ILS: Issue Logging System.
- ☐ Initiator: Any employee that logs in an issue on the app.
- ☐ Issue: Refers to a technical issue.

- ☐ Location: City of the branch where the issue was logged in.
- ☐ Log an issue: Fill a form with the required information about the technical issue and upload it.
- ☐ Manager: Manager of a support team.
- ☐ MOM: Minutes of Meeting.
- ☐ MVC: Model-View-Controller.
- ☐ RQ: Requirement.
- ☐ SDS: Software Designs Specification
- ☐ SQL: Structured Query Language
- ☐ Sub-category: The more specific category of an issue. For example: any issues related to keyboard will have the sub-category as keyboard.
- ☐ Sub-location: Area of the branch where the issue was logged in.
- ☐ Sublime text: A cross-platform source code editor.
- ☐ Support: Any employee part of a support team.
- ☐ Support Team: The employees at ABL that can resolve issues in a specific category from the support team of that category.
- ☐ TCP: TCP (Transmission Control Protocol) is a standard that defines how to establish and maintain a network conversation through which application programs can exchange data.
- ☐ Technical Issue: A hardware or software problem related to electronic devices that have been issued to an employee or a branch by the bank and is property of the bank.
- ☐ UI: User Interface
- ☐ Work Category: the expertise area of a support team member.
- ☐ Xcode: Integrated development environment for macOS and iOS.
- ☐ XSS Attack: a web security attack caused by inserting html or script files into placeholders that may get executed if preventions are not in place.

## 2.5 References and Acknowledgments

Geographic Locations of Allied Bank's branches in Pakistan: <https://www.abl.com/business-banking/home-remittances/branch-and-atm-locator/>

## 3 Overall Description

### 3.1 System overview

The purpose of the system we are aiming to develop is to act as the initial prototype of a mobile based platform that can be utilized by the current employees at Allied Bank Limited to primarily act as communication bridge between any employee that can report technical errors of electronics in use and a special team of technicians, also employees at ABL, that can resolve these repairs. A secondary use of this system is to monitor the total number of requests against Area/ Category/ Sub-category/ Location.

Our system aims to develop a more mobile system than the website version of the ILS that is currently being used in ABL branches. The system is being built to cater to five different classes of employees at ABL: Initiator, Support, Manager, Monitor and Admin. However, all these five classes of people are employees at ABL and have their own credentials for their use across company wide software's, including this one. For the scope of this course, dummy usernames and passwords will be pre-installed into a temporary database for use with this application. Predefined credentials are essentials for any user to use the system.

Once a user has logged in using their credentials, displays relevant to their roles will be accessible to them. All types of users will be able to log in their issues, view the status of their issues as well as close or reopen a resolved request. Users will also be able to perform other services through the app as per their category description below:

Primary Actor	Associated Use cases
Initiator	<ul style="list-style-type: none"><li>6. Login using credentials</li><li>7. Log support related issues</li><li>8. View status of own issue</li><li>9. Mark own issue as closed or open it again</li><li>10. Logout from system</li></ul>
Support	<ul style="list-style-type: none"><li>11. Login using credentials</li><li>12. Log support related issues</li><li>13. View status of own issue</li><li>14. Mark own issue as closed or open it again</li><li>15. Logout from system</li><li>16. View issues of in-city branches matching their work category.</li><li>17. Assign open issues to themselves.</li><li>18. Mark resolved issues as fixed.</li><li>19. Reopen resolved issues.</li><li>20. Reassign issues to other team members.</li></ul>

Manager	12. Login using credentials 13. Log support related issues 14. View status of own issue 15. Mark own issue as closed or open it again 16. Logout from system 17. Assign open issues to themselves. 18. View Status of Issues pending with their team. 19. Mark resolved issues as fixed. 20. Mark resolved issues as open. 21. Refer Issue to other Support Team 22. Assign/Re-assign Issue to other team members. 23. Monitor reports
Monitor	7. Login using credentials 8. Log support related issues 9. View status of own issue 10. Mark own issue as closed or open it again 11. Logout from system 12. Monitor reports
Admin	8. Login using credentials 9. Log support related issues 10. View status of own issue 11. Mark own issue as closed or open it again 12. Logout from system 13. Change Role 14. Change Team

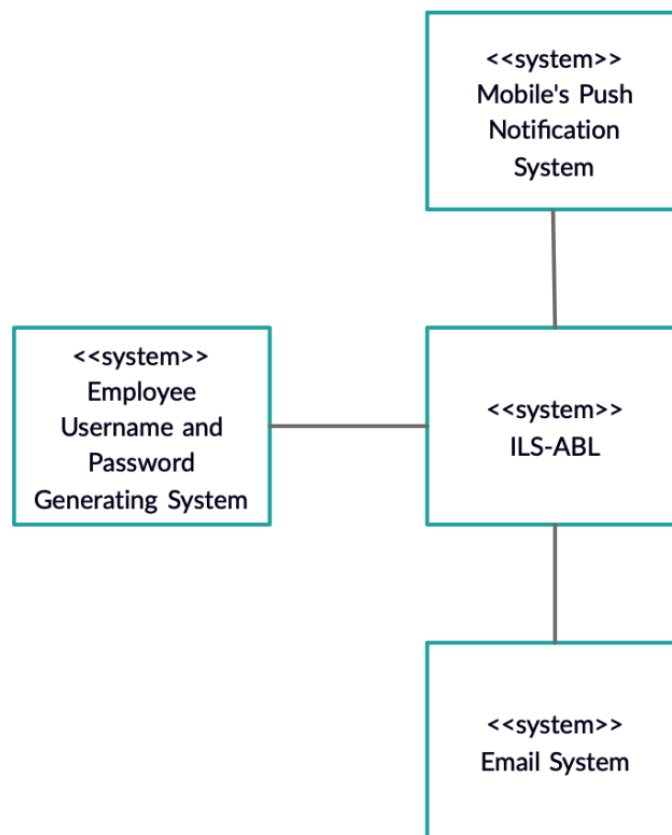
The methodology of resolving issues through the mobile application (as per client's requirements) is:

1. A user logs in either their own issue or someone else's on their behalf using their account. The status of the issue is Open.
2. Any support role user will be able to view Open issues in their city through the system and can either take up or get assigned the job, by certain roles, of resolving the issues through their account in the system. The status of this issue will be marked by the support role member as Job Taken using their account in the system.
3. Once the support role user resolves the issue, they mark the status of the issue as Fixed using their account in the system. If they are unable to resolve the issue, they can mark it as Open again or assign it to someone else.
4. If the initiator of the issue is satisfied, they mark the status of the issue as Closed using their account in the system. If they are not, they can Open the issue again.
5. If any Fixed issue is not marked as closed for a period of 48 hours, then it should automatically be marked as Closed.
6. If nobody takes the job within 48 hours, then the issue is automatically assigned to the manager of the relevant support team.
7. Specific emails and push notifications are associated with the method as per the functional requirements.

The design approach we are going for to build our system is distributed in nature, following very closely the client server architecture. The system is being constructed to run on mobile devices (Android and iOS both) all linked to a centralized cloud database that stores the uploaded logs and their status as well as employee information like login credentials.

Since the app is mobile based, we aim to make it more friendly for use by avoiding too much text as text can be hard to read on a smaller smartphone screen. We further aim to design the app screen by using already available widgets of buttons, placeholders, drop down menus, check boxes etcetera. The physical appearance of the application design will be centered around the colour of our clients logo: orange, blue and white. More elaborated and additional details on design have been given in 3.3.

Attached is the context diagram that illustrates the ILS and the other systems in its environment.



## 3.2 System constraints

1. The full implementation of an ILS application for ABL requires scalability that can accommodate users across the whole country. However, due to financial constraints and limited resources, our prototype system might not be safely scalable on this level since testing as well as deployment at this level is not financially feasible at this point.
2. The system we are trying to develop is almost a replica of the website version of ILS. Thus, we have to face the constraint of sticking to functionalities of the website.
3. Because our application is being made for a bank that has strict security requirements, we have had the trouble of not getting very detailed information about their existing system. This might make our application unable to be immediately deployed for use since certain changes might have to be made to make our application exactly like the website by the website developers.
4. We face a constraint of time since all the required functionalities must be developed within the dedicated month of development of the application.
5. An added constraint is the fact that all five of us group members are coding a mobile application for the first time and so have limited development skills.
6. Because of the nature of its use, an added constraint to every functionality is the fact that every feature must be secure and safe against potential attacks.
7. There is also the constraint of developing an application that is compatible for both Android and iOS devices, having the same functionalities for both types.

## 3.3 Architectural strategies

The design approach we are going for to build our system is distributed in nature, following very closely the client server architecture. The system is being constructed to run on mobile devices (Android and iOS both) all linked to a centralized cloud database that stores the uploaded logs and their status as well as employee information like login credentials. Communication will occur via the internet using WiFi and/or 3G/4G using TCP to ensure reliable delivery of packets, which will help with the scalability of the system too.

To develop our application, we have decided to use the Flutter framework based on the Dart language. We've decided to go ahead with flutter because firstly, we're aiming to build an application for Android and iOS devices. Flutter would allow us to save time in order to develop an application for both these platforms as compared to a Native i.e. operating system specific framework. Flutter also abstracts away a lot of complexity that comes with developing native Android and iOS apps. We also believe that widgets of Flutter will help us in developing an easy and aesthetic UI for our application. Further, flutter allows greater reusability of code since widgets allow abstractions that promote repeated use of code. Additionally, flutter also has existing documentation which can assist our journey of exploring it for the development phase.

For our backend database, we need a relational cloud database that can support querying with SQL. Thus, we have decided to use Heroku that has extensions available for easy SQL querying. To deploy code to Heroku, we will be using GitHub that can very easily be integrated with Heroku. Additionally, using an external database will also be in compliance with the bank's standard since the safety of the data will not be in our hands alone.

To support authentication, we are planning to use already coded login system parts modified to our use since we want to ensure that our system is not breakable.



Since the app is mobile based, the mode of communication between a user and the app will almost always be using the touchscreen. We thus aim to make the ILS mobile app more friendly for use by avoiding too much text as text can be hard to read on a smaller smartphone screen. Thus, we aim to design the app screen by using widgets of buttons, placeholders, drop down menus, check boxes etcetera. The physical appearance of the application design will be centered around the colour of our clients logo: orange, blue and white. Additionally, we plan to reuse as many widgets and segments of code we can find relevant to our application from online sources to ease our testing process since code already in use has been rigorously tested previously.

Our application doesn't aim to use much additional storage on mobile devices other than the application size: issue storage and retrieval will occur directly from the cloud database. To tackle the synchronization problem, we plan to automatically refresh a page, and hence query again to the database, after a fixed amount of time. Carrying out communication using TCP also helps the cause. We also aim to make the application run swiftly on low end mobile phones i.e. those with less RAM or other older generation phones. This is achieved by not incorporating any memory eating processes in the application and ensuring that the app does not consume a lot of unnecessary resources.

As mentioned, our application right now is a prototype and if approved by the client, in future, we plan to make it deployable to such a level that every employee and at every ABL branch across Pakistan can be connected without any sort of scalability failures. For this larger objective, scalability has been kept in mind when designing the application.

## 4 System Architecture

### 4.1 System Architecture

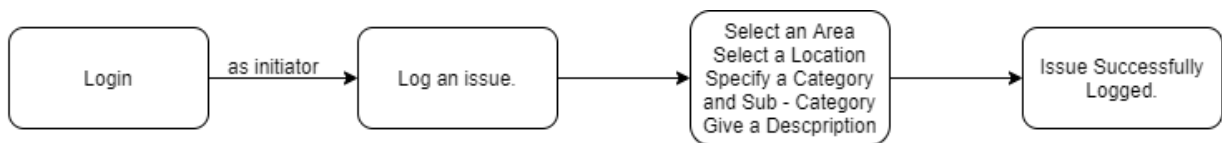
The system is overall decomposed into seven major parts. Firstly, **Login** the login page will be the first component of the system that the user will interact with. The Login system will interact with another major component of the system, the **"Database"**. This component of the system contains the Tables that will be used throughout the application. The login system interacts with the Employee Table only, when a user enters the credentials to login the system validates them from the employee database and grants access to the user according to their position in the hierarchy, the role/position of the user is acquired from the roles table in the database which is linked to the employee table itself.

The **"Database"** component of the system includes five different tables, Employee table, Branch Location table, Issue table, Roles table and Support Team table. The employee table will carry all the details about a user/employee needed for the application to function. The Branch Location table will contain details about all the branches in Pakistan. All the details and data for these two tables will be provided by the client itself subject to approval and deployment. For testing purposes we will be making dummy schemas and will add dummy data. The third table will be the Issue table which will contain all the issues logged by the application along with all the necessary details provided in the next sections. The role table will carry the information about each employee's role in the hierarchy. The Support team table will carry details necessary about every support staff member who will already be present in the employee table as a normal employee.

After the Database component the next major components of the system are stated below:

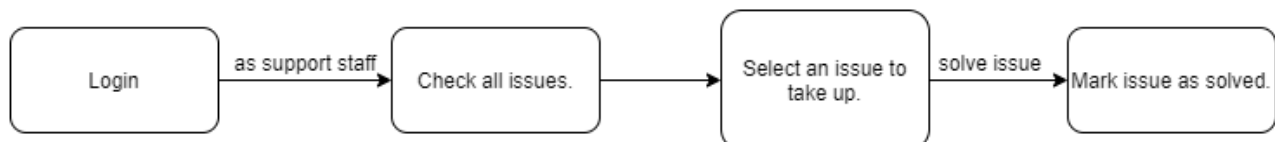
#### 1) **Log an issue:**

This component of the system will be responsible for the logging of an issue that is the main function of the application. It will be interacting with the branch location database of the bank and will be storing the issues being logged with all information in another database.



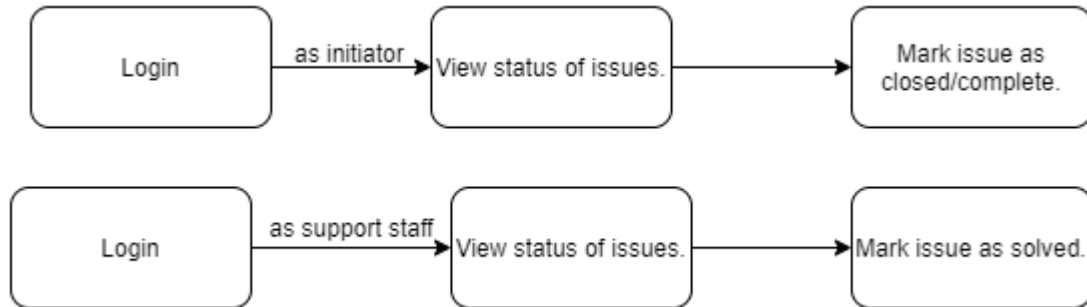
#### 2) **Check all issues:**

This is where a support staff member will enter and will have an option to take up an issue to resolve. Administrators will be able to use this component to check the progress of issues and observe performance of staff members and departments etc.



### 3) View status of issues:

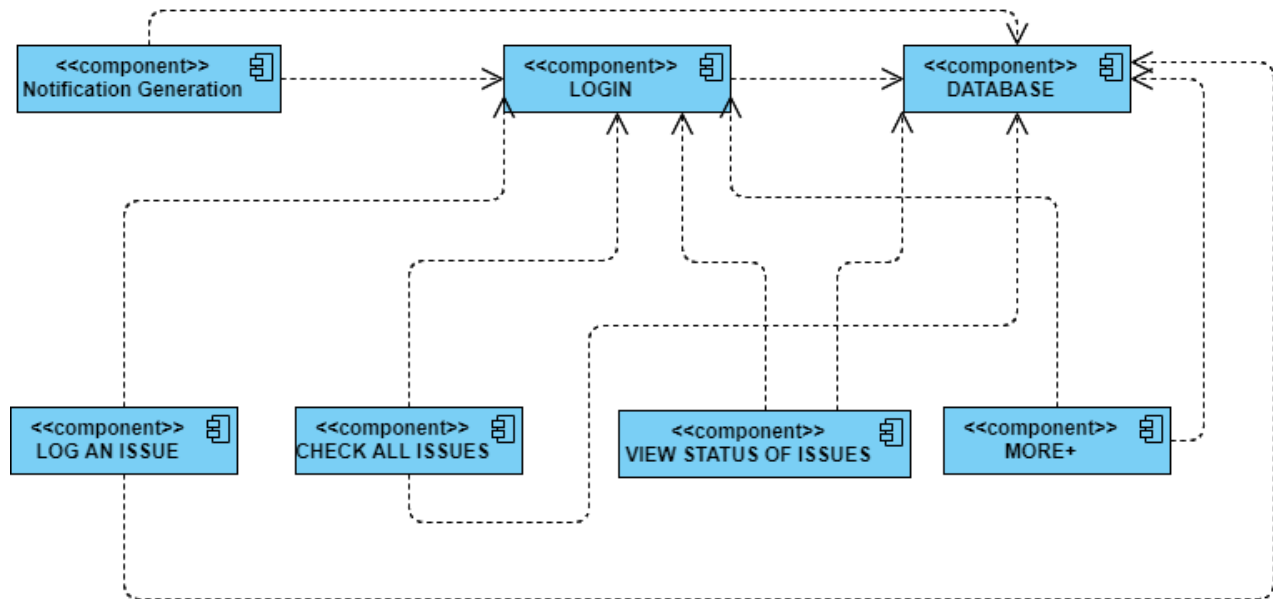
The issues taken-up or logged by a user will be visible in this tab. The user as a support staff member will be able to mark the issue as completed and as a user who logged in an issue will be able to check the status of the issue or mark it completed. In case of not being satisfied with the resolution of the issue the initiator can re-log the issue.

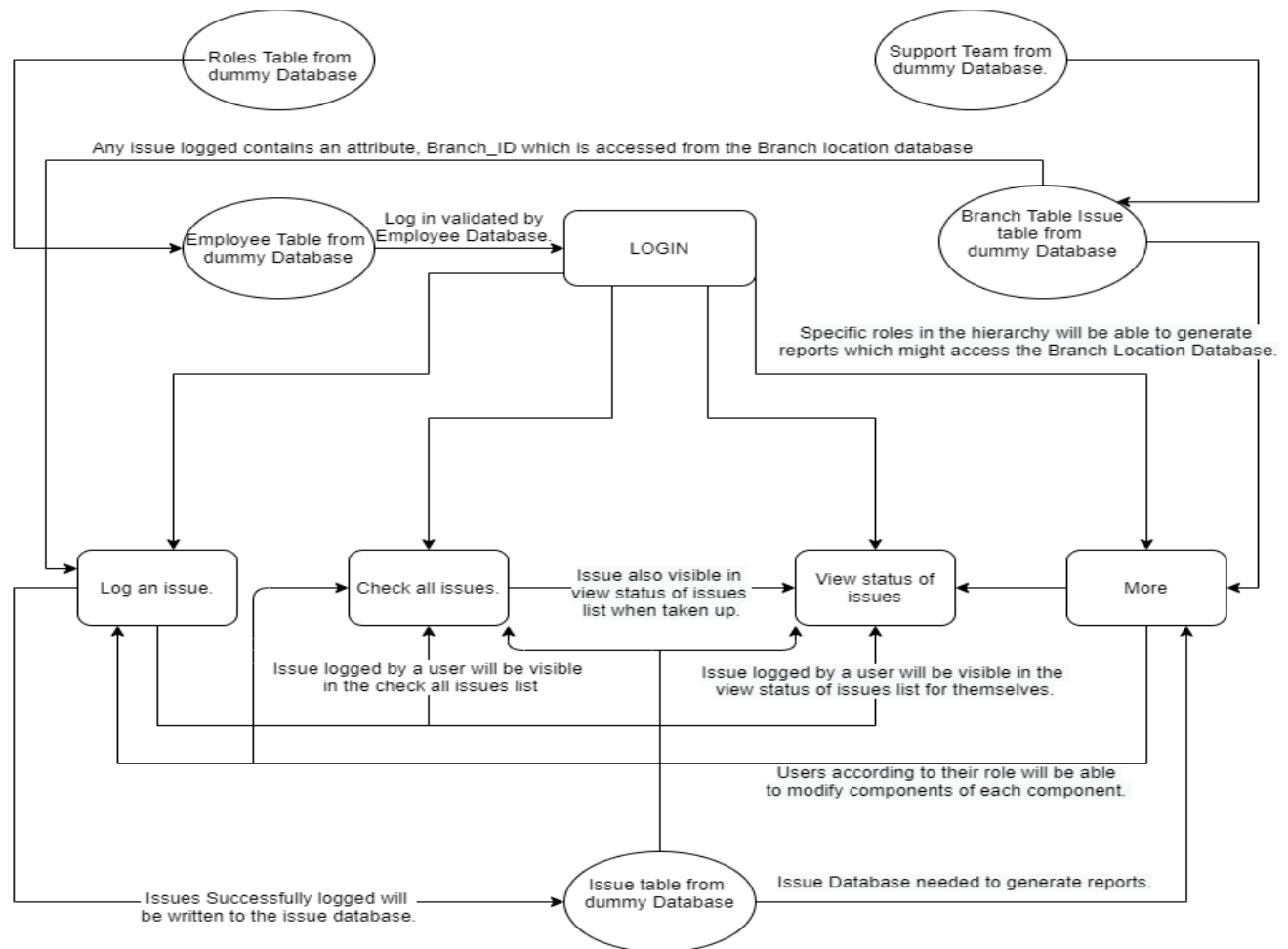


### 4) More options:

This component of the system is totally dependent on the role of the person who is accessing the system at the moment. As mentioned before there are 5 roles assigned which have the following hierarchy, initiator, support team member, manager, monitor and administrator. This component will enable them to use the attributes/options that are particularly available to their respective role only. For example, only the administrator has the power to change the team or to change the role of an employee so the **More component** will direct the administrator to a screen where all such actions will be listed.

Another very important component of the system is the **notification generation** component which will contain the mechanism to generate notification in the form of emails as well as push notifications on the user's device. This component of the system will not be directly interacting with the user, it will be stimulated to act in cases when the user is interacting with the other components such as an email notification for a successfully logged issue.

**Component Diagram:**

**Activity Diagram:**

The above diagram shows a general flow of the components. It tells how a user interacts with these three components. The fourth component “**More options**” component will be explained in the next section in detail as it has a different functionality according to each role.

This decomposition of the system was adopted because it divides the major components of the system that will be used the most into different parts in terms of functionality. It divides the software in a way that makes the user experience smooth and helps us avoid clutter and mixed up functionality. The login component is kept secluded as it has to be used only once at the start and any of the functionality in that component does not have to be used in any processes after login. The four major components stated above in bullet points are divided as they had been divided in the same way by the client itself for the existing Web ILS that is in place already. The More component is kept aside to provide functionality subject to the role of the user logged into the application and lastly the notification generation component is kept different as it will be interacting to the Database component on its own for example to fetch an email address to send an email and also because the user will not be directly interacting with the notification generation functions for example the user cannot send an email through the ILS but the actions of the user can result in an email.

## 4.2 Subsystem Architecture

The More options component of our system, specifically the “**More**” button carries all the functionality which is subject to the role of the user. Below is described what the “**More**” button contains for all the roles.

### **Initiator:**

The initiator will have no options than the ones already provided on the main page.

### **Support Team Member:**

The support team member will have an additional functionality from which he/she will be able to assign any issue from the issue list to any other support team member. This change will be written to the database in the issues table specifically.

### **Manager:**

The manager will be able to refer the issue to members of other support teams as well as to other team members. These changes will be written to the database and will initiate email and push notifications. Another additional functionality for the manager will be to generate and monitor reports on issues logged in the past, this will involve fetching of information from the database to provide it to the user.

### **Monitor:**

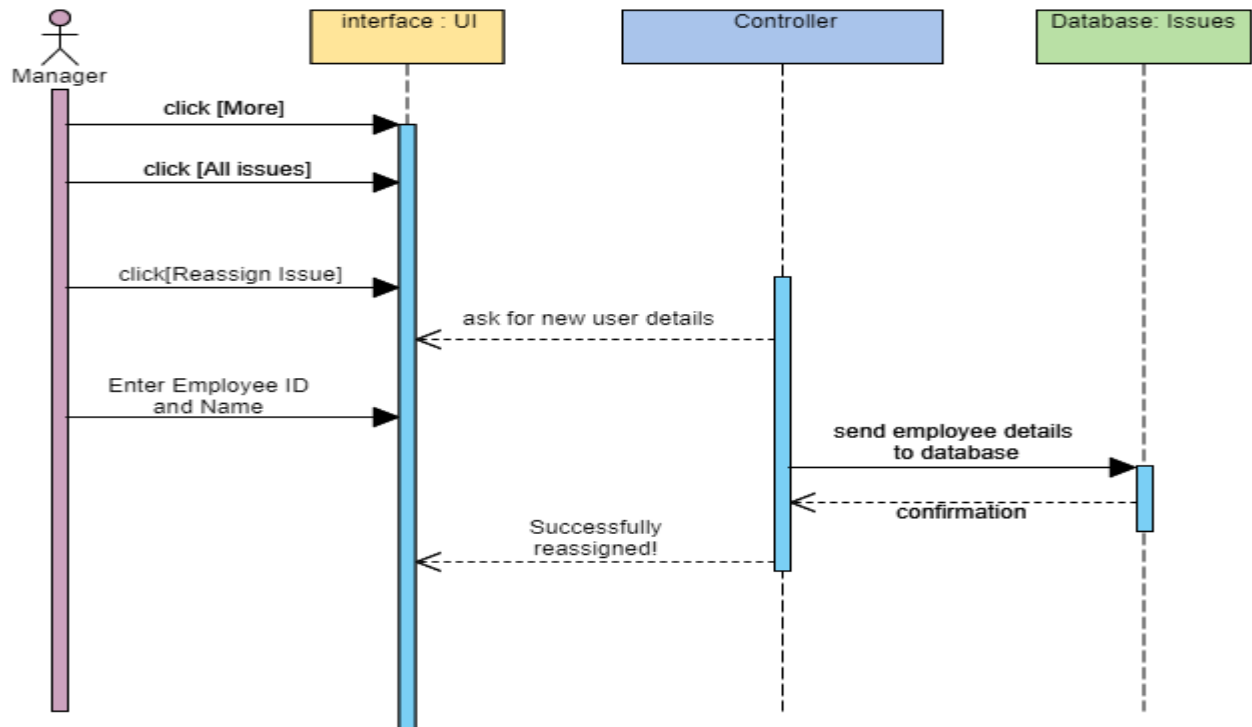
The monitor will be able to generate and monitor reports on issues logged in the past, this will involve fetching of information from the database to provide it to the user.

### **Administrator:**

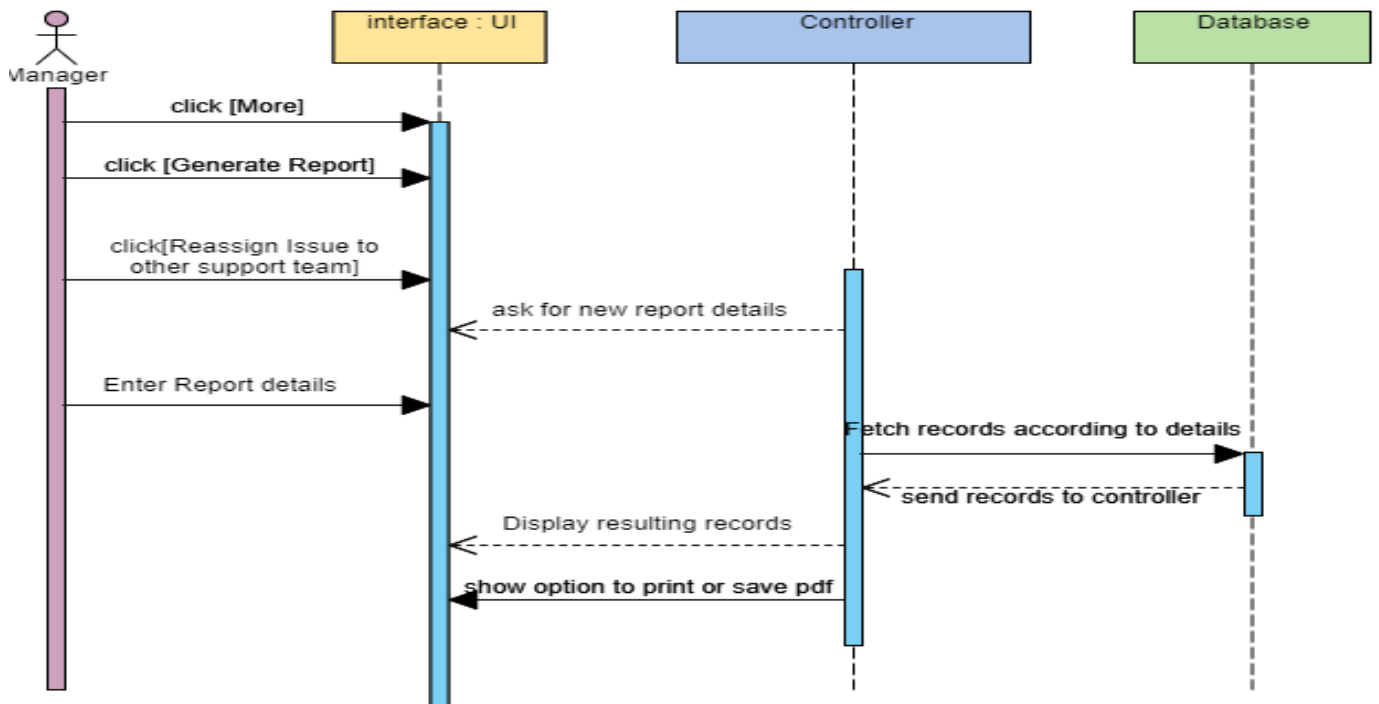
The administrator being the topmost role in the hierarchy will have options to change roles of employees already present in the employee database and to change the team of a specific employee. These changes will be written to the database in the issues table specifically.

These cases are described in detail in the sequence diagrams shown below:

- 1) The following sequence diagram shows how a Manager, or a Support Team Member will be able to re-assign an issue to another team member.

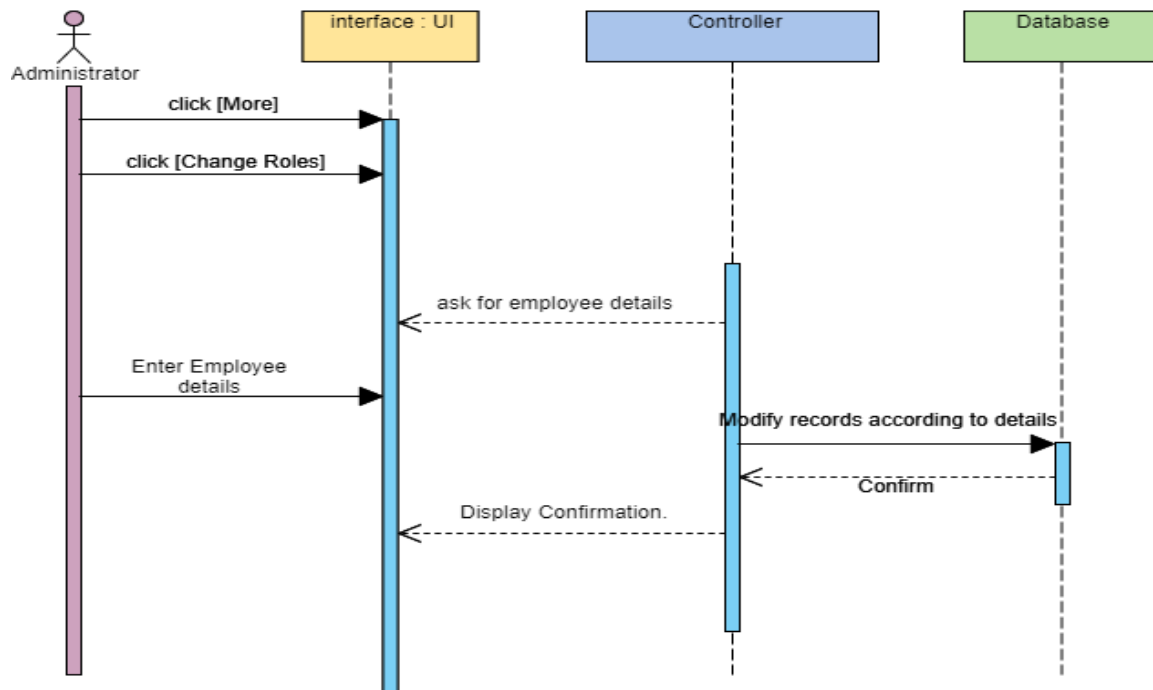


- 2) The following use case diagram shows how a Manager or a Monitor will be able to generate reports.

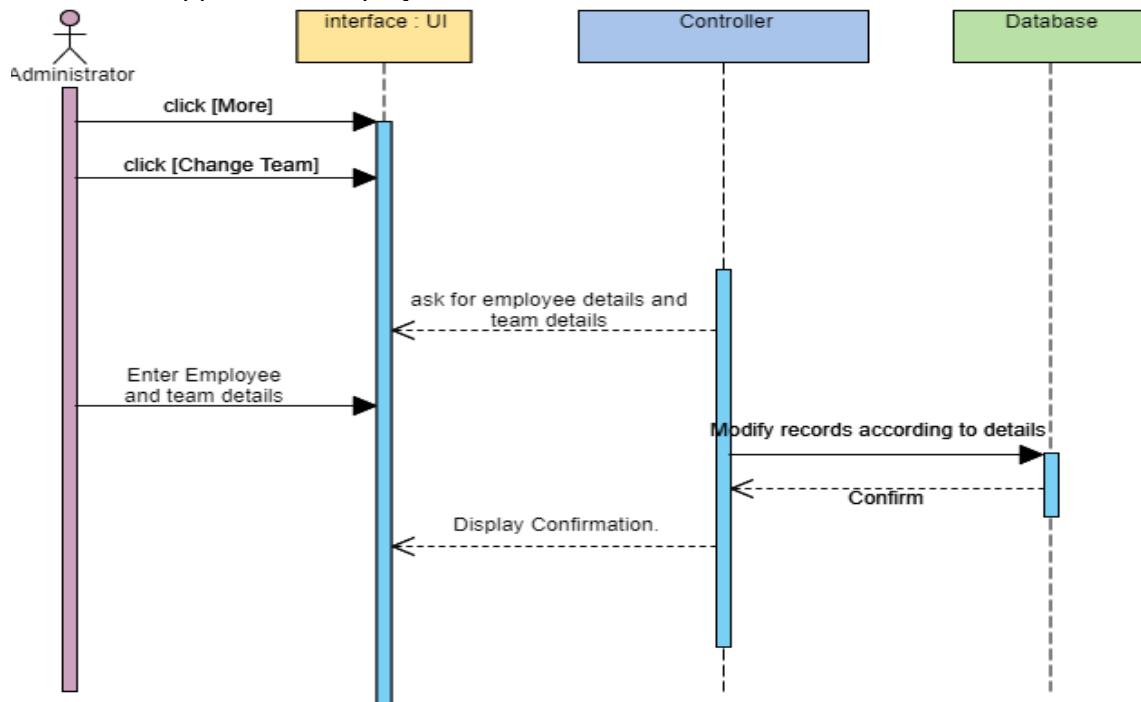




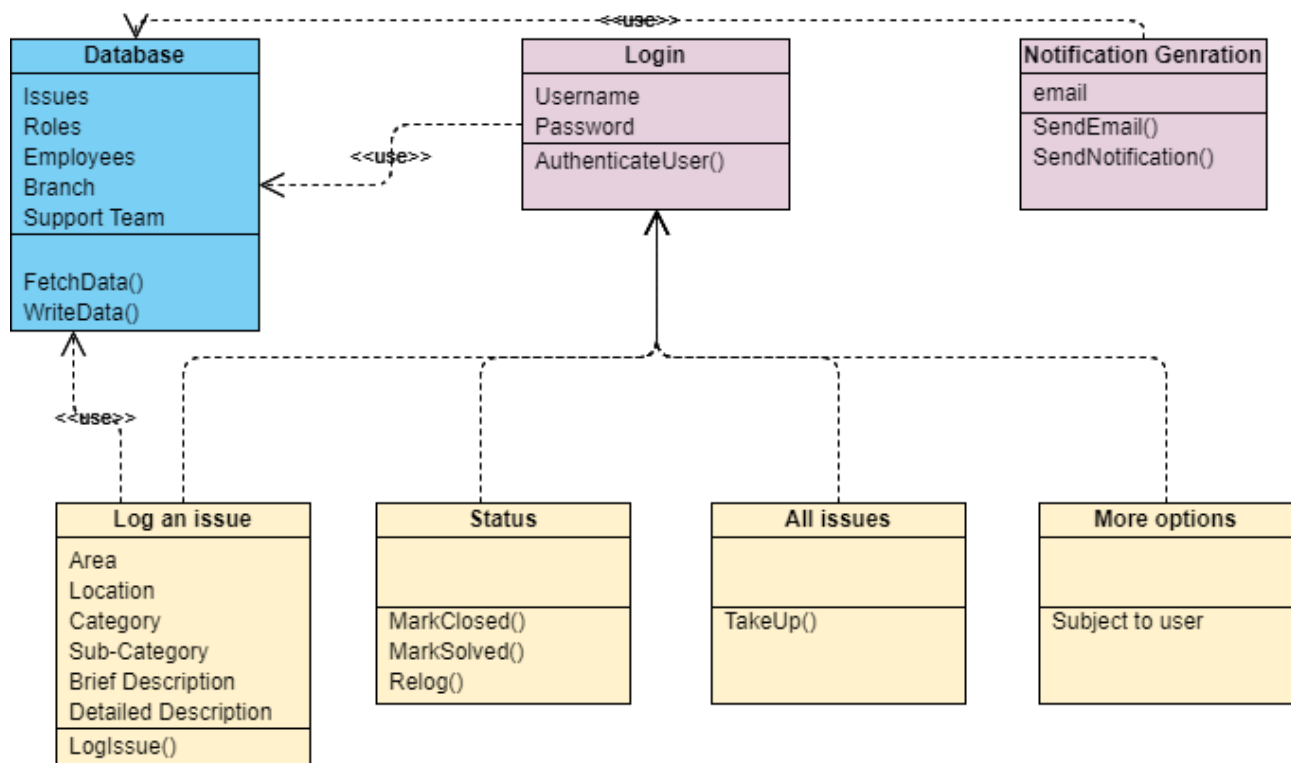
- 3) The following sequence diagram shows how an Administrator will be able to change the role for an employee.



- 4) The following sequence diagram shows how an Administrator will be able to change the team for a support staff employee.



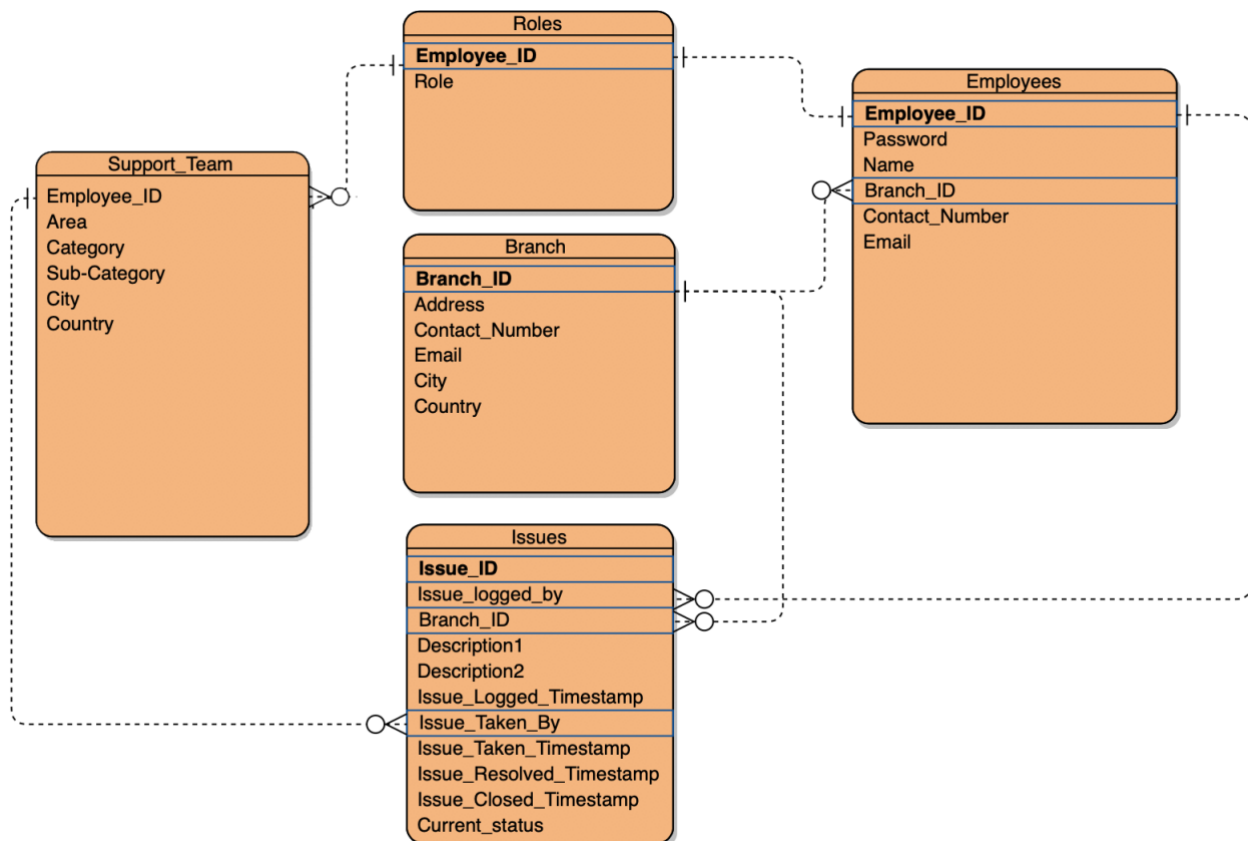
Class diagram for the system:



### 4.3 Database Model

We are restricted to using a relational database management system for our ILS application because the client's current web ILS is working with a relational database and to mimic that into our dummy database, we are required to use the similar database. Relational databases are categorized by a set of tables where data gets fit into a pre-defined category. The tables consist of rows and columns where the column has an entry for data for a specific category and rows contain instances for that data defined according to the category. The standard user and application programming interface (API) of a relational database is the Structured Query Language (SQL) which we will be using throughout our application for creating, reading, updating or deleting the data.

#### 4.3.1 Database scheme and detailed description



Our application will be working with one database consisting of 5 tables. The names of these tables and their details are as followed:

**1) Support\_Team:** This table stores details of all support teams for the respective issues. Employee\_ID will be the foreign key because every employee\_ID can be member of more than one support teams, it is then followed by 'Area' of expertise of the support team member e.g. IT-Support, the 'Category' e.g. hardware, the 'Sub-category' e.g. keyboard or printer, followed by the team's location.

**2) Employees:** This table stores details of all employees, consisting of their unique 'Employee\_ID', employee's password, their name, the 'Branch\_ID' they work in, their contact number and email.

**3) Roles:** This table stores information regarding what role has been assigned to what 'Employee\_ID' which is also the primary key in this table.

**4) Branch:** This table stores branch details i.e the unique 'Branch\_ID', it's address, location, and contact details like contact number and email.

**5) Issues:** This table contains details of the issues that are logged. It contains unique 'Issue\_ID', the 'Branch\_ID' and 'Employer\_ID' of individual who logged the issue, followed by the description of issue logged and it's timestamp, who the issue has been taken or assigned to and its timestamp and also the timestamps of when the issue of marked as resolved and later marked as closed. The table will also include 'current\_status' for every issue showing its progress.

Each table has a common relation with another table. All these relations are described below:

Field1	Field2	Relationship type
Employees.Employee_ID	Support_Team.Employee_ID	one-to-one
Roles.Employee_ID	Support_Team.Employee_ID	one-to-many
Support_Team.Employee_ID	Issues.Issue_Taken_By	one-to-many
Employee.Employee_ID	Issues.Issue_logged_by	one-to-many
Branch.Branch_ID	Issues.Branch_ID	one-to-many
Branch.Branch_ID	Employees.Branch_ID	one-to-many

### **4.3.2 Database**

We have chosen a relational database because the client has refused to authorize access to the database where the Bank's existing web ILS is running and also our ILS mobile application will eventually run to assure safety and security of employee's personal information. Our client has however agreed to provide us with the structure of their existing database and using that structure we have designed this dummy database which our application will use for running and testing, Once client approves the application, the bank's IT department will themselves merge tables we have in our dummy database that do not exist in their current running database i.e. 'Issues' table, and they will be easily able to run the application with their own database. This is why the structure of our database tries to overlap with their provided information and so is chosen this way.

To implement our relational database, we will use ClearDB as an add-on by Heroku. ClearDB is a cloud, hybrid, and on-premise database-as-a-service for MySQL powered applications. It stores and manages your MySQL database for you so that you don't have to deal with things like database servers, replication, advanced storage, IT support, and especially things like database failures. ClearDB is designed around the idea that systems, networks, and storage fails from time to time, which under ordinary circumstances can leave your database offline. ClearDB solves this problem by running a geo-separated, mirrored database cluster utilizing a combination of MySQL master-master replication technology as well as ClearDB's own custom SQL routing technology which detects failures and automatically re-routes connections to the servers which are still online. Once the system that has failed is back online and is back in sync, CloudDB's routing technology will automatically switch back to the newly recovered server instance so that you get the lowest latency and highest performance which will be ideal for the ILS application.

## **4.4 External Interface Requirements**

### **4.4.1 User Interfaces**

The ILS mobile application front-end is being implemented on Flutter, which will make it possible to develop a very user-friendly GUI for the users. By using Flutter for our cross-platform application, our GUI will optimize itself according to any mobile device's screen size as well as give similar user experience to all users in terms of speed and efficiency. The client explained that the ILS application will be used by all employees belonging to all age groups. Therefore to make it a user-friendly experience for all users, we have used

- large and easy to read fonts,
- large dark coloured buttons with prominent button text,
- light and consistent colour scheme with distinguishable colourful icons and
- easy to fill forms by mainly having to choose options from dropdown menus to assure minimum text fields to fill by user.

#### **4.4.2 Hardware Interfaces**

- The ILS application will be available for all IOS and Android devices.
- The device's free storage space required for application will be mediocre and devices will be able to run the application with no excessive load on the processors.
- The device with an ILS application installed has to be connected to a stable internet connection via Wi-Fi router/mobile data to use the application. This is to assure stable communication between the host and server.
- The application's print reports feature will require a wireless printer to print the reports.
- The application will need access to the device's camera if the user chooses to insert a supporting picture while logging an issue.

## 5 User Interface Design

### 5.1 Description of the user interface

#### **Flutter**

We will be using Flutter as the front-end development tool. We had many options to choose from and the decision of using Flutter was taken after considering the following advantages of Flutter over other front-end development tools. Firstly, flutter has a very less code development time. This basically means that the new code can be shipped to the testing device in a lesser time as compared to its competitors. Secondly, Flutter does not need any platform specific UI components to render its UI. As we intend to make an application for both android and iOS, then in our opinion using flutter will reduce a significant amount of hassle involved in making it a cross platform application. Moreover, in Flutter, we can perform functions like fetching GPS coordinates, login authentication from our dummy database through ready to use plugins supported by Google. Lastly, Flutter provides us a greater extent of ability to customize the UI as compared to the other available tools.

#### **Android Studio**

Android Studio will be used for the development of the Android version of our mobile application. It features an intelligent Code Editor equipped with IntelliJ IDEA interface, which makes code writing and analysis faster, easier and more accurate.

#### **Xcode**

Xcode will be used for the development of the iOS version of our mobile application. Xcode IDE aids users in developing apps for Apple devices by providing them with an Assistant Editor. This tool automatically shows resources that it deems useful in the current coding process while not interrupting the development and editing in the main window. This allows developers to quickly find data that could be of assistance to them in completing the project faster.

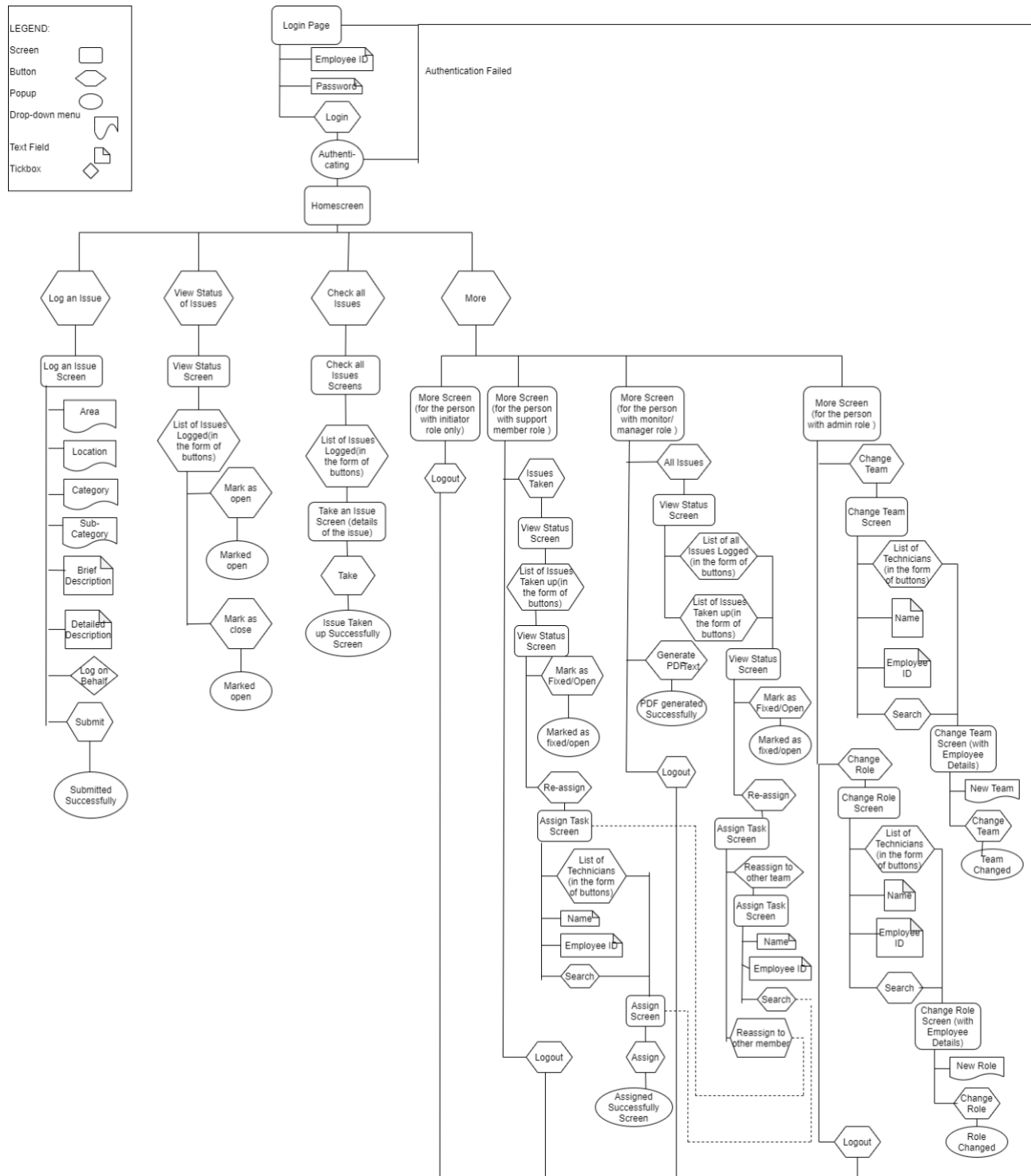
#### **Sublime Text**

Sublime text will be used as the text editor. It is a fast, stable and mature text editor which also has an updated Dart plugin. The updated plugin supports syntax highlighting for modern Dart code and includes several common Dart snippets to help code faster. Moreover, it is also user-friendly and easier to use for our team as we have been using it for quite a while now.

#### **MVC Presentation Layer**

The MVC presentation layer consists of two components: view and controller. The UI of our application basically forms the view component of the MVC presentation layer, and the controller component will be the code of our application which will decide whether to access another layer or return a view right away.

## 5.2 Information architecture





## 5.3 Screens

### Initiator Use Cases

#### Use Case 1: Login using credentials

The figure 1.1, will be the first screen the user will see upon opening the application. The screen will have two text fields, one for the Employee ID and the second one for the password. After entering the credentials, the user will click on the 'Login' button and the data entered will be sent to the server for authentication. If the credentials are right, the user will be directed to the home screen, i.e. figure 1.2, otherwise he will be prompted to enter the correct details again

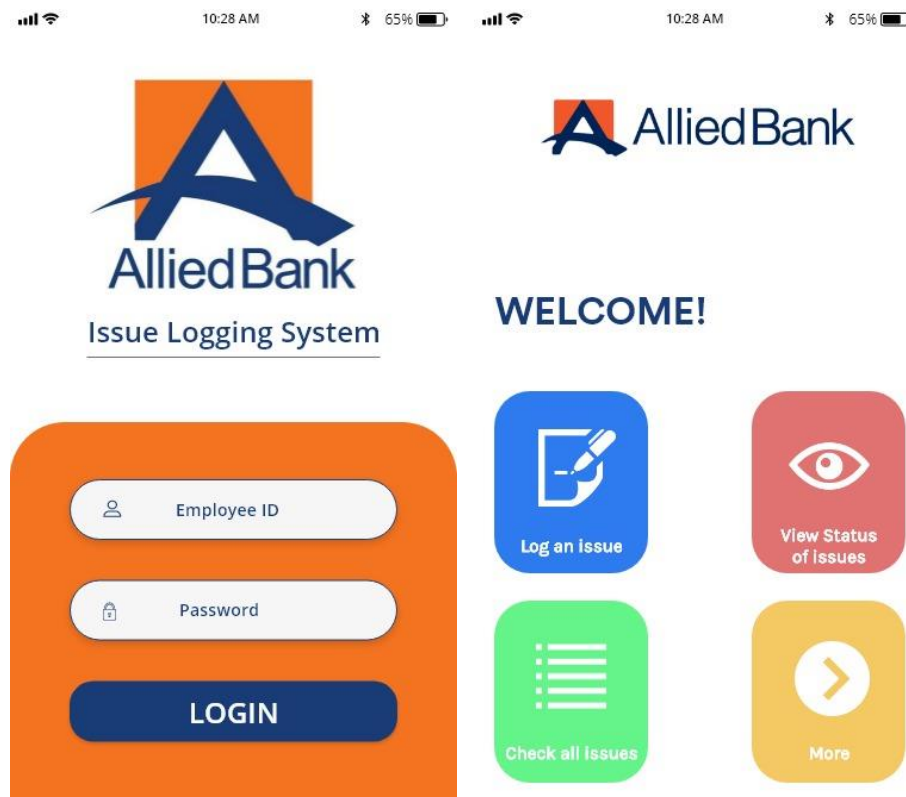


Figure 1.1

Figure 1.2

**Use Case 2: Log support related issues**

To log an issue, the user will click the 'Log an Issue' button from the homescreen, i.e. figure 1.2. Upon clicking that button the user will be directed to the 'Log an Issue' screen, i.e. figure 2.1. The area and location of the branch fields along with the category and sub-category fields will get input from the dropdown lists, the buttons of which have been placed adjacent to the respective fields. Then there are also text fields for brief and detailed description of the issue to be logged. Lastly, there is a checkbox to click on, in case someone is logging an issue on someone else's behalf. Once all the fields, except the non-mandatory checkbox, are filled the user can proceed by clicking the 'SUBMIT' button and he will be able to see the pop-up 'Submitted Successfully!' as shown in figure 2.2.

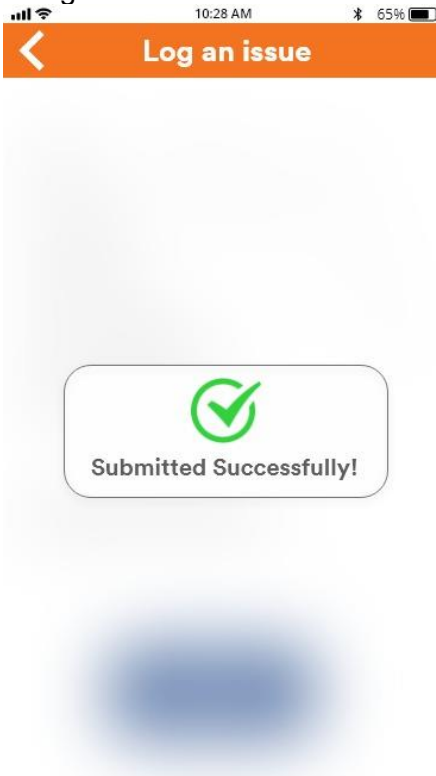
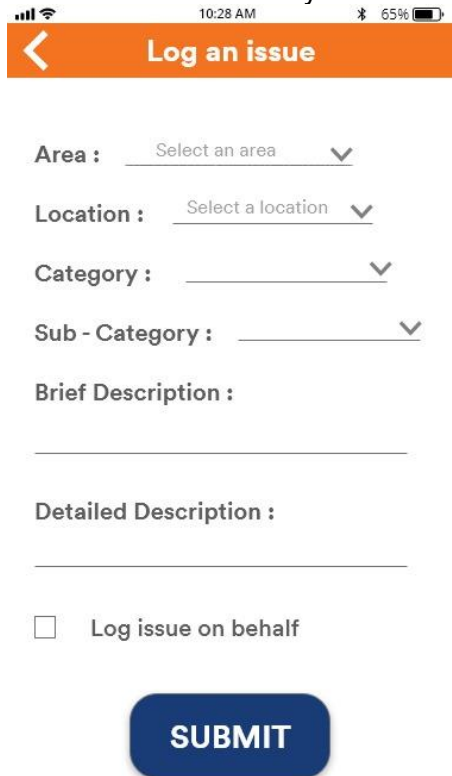


Figure 2.1

Figure 2.2

**Use Case 3: View status of own issue**

To view the status of the issues logged by the user himself, he will click on the 'View Status of Issues' button from the homescreen, i.e. figure 1.2. The user will be directed to the 'View Status' screen, i.e. figure 3.1, where he will be able to see the issues logged by him and their statuses in the form of red, yellow and green icons. A Red icon represents the issue which has been closed, green icon represents the issue which is open, and the yellow icon represents the issue which has been fixed and waiting for the user to mark it as close or open it again.

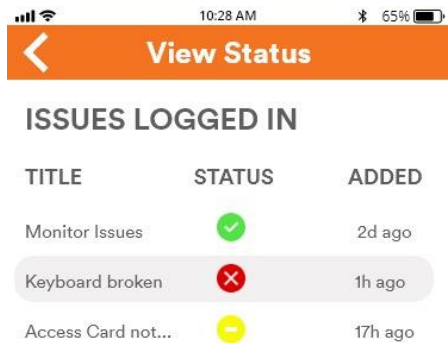


Figure 3.1

**Use Case 4: Mark own issue as closed or open it again**

Once the Issue logged by the initiator has been fixed. He will be able to mark that issue as closed or open it again, in case the initiator is not satisfied by the job, by navigating to the 'View Status' screen, i.e. figure 3.1. The user will reach this screen by clicking on the 'View Status of Issues' button from the homescreen, i.e. figure 1.2. Once there, he can click on the issue, with the yellow icon as its status, and he will be directed to a new 'View Status' screen, i.e. figure 4.1. Here the user can click 'Mark as close' button, in case he is satisfied by the job, or he can click on 'Mark as open' button if he is not satisfied by the job and wants to open it again. The user will see the pop-up notification corresponding the button which he pressed, i.e. figure 4.2,4.3.

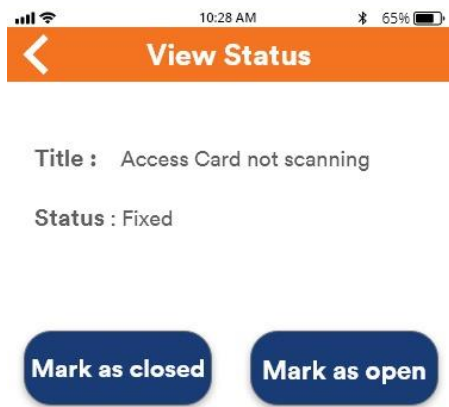


Figure 4.1

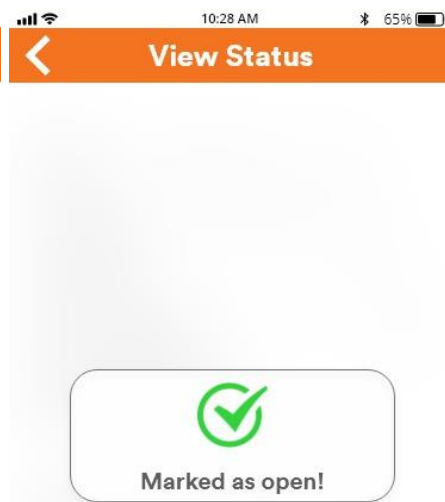


Figure 4.2

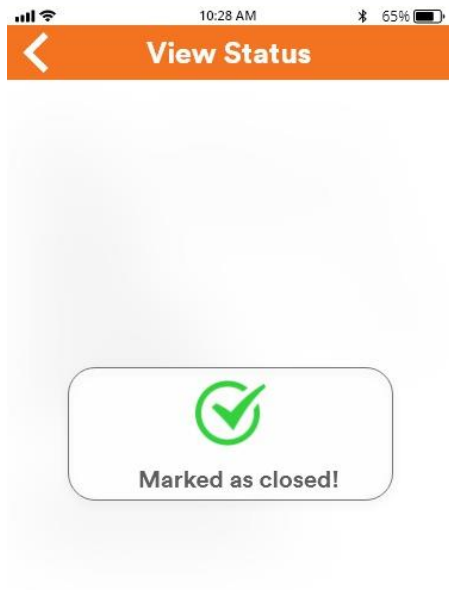


Figure 4.3

**Use Case 5: Logout from the application**

To Logout from the application, the user should click on the 'More' button from the homescreen, i.e. figure 1.2. The user will be directed to a screen with a 'Logout' button, i.e. figure 5.1. Upon clicking that button, the user will be logged out and redirected to the Login screen, i.e. figure 1.1.

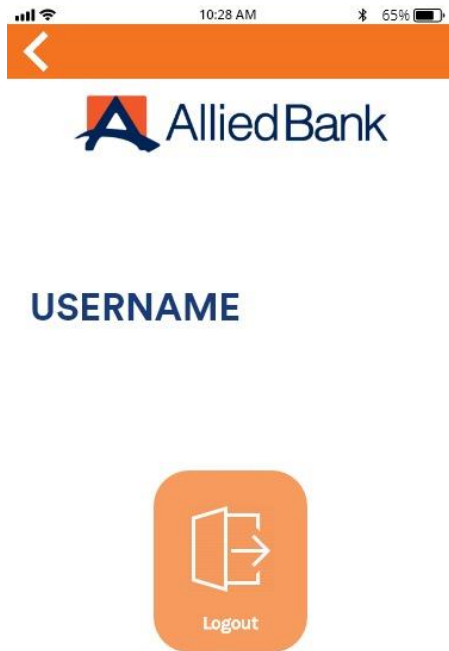


Figure 5.1

**Support Team Member Use Cases****Use Case 1: Login using credentials**

Same as the Use Case 1 for Initiator

**Use Case 2: Log support related issues**

Same as the Use Case 2 for Initiator

**Use Case 3: View status of own issue**

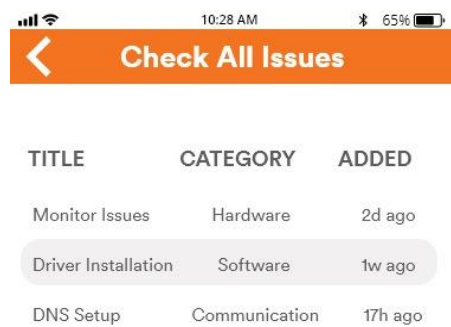
Same as the Use Case 3 for Initiator

**Use Case 4: Mark own issue as closed or open it again**

Same as the Use Case 4 for Initiator

**Use Case 5: View issues of in-city branches matching their work category**

To see the issues logged in the branches, located in the city assigned to the technician, matching to his work category, he will click the 'Check all Issues' button from the homescreen, i.e. figure 1.2. The user will be directed to the 'Check all Issues' screen, i.e. figure 5(a).1, and he will be able to see all the relevant open issues in the form of a list.



The screenshot shows a mobile application interface. At the top, there is a status bar with signal strength, time (10:28 AM), and battery level (65%). Below the status bar is an orange header bar with a white back arrow icon and the text 'Check All Issues'. The main content area displays a list of issues in a table format. The table has three columns: 'TITLE', 'CATEGORY', and 'ADDED'. The first row shows 'Monitor Issues' under 'Hardware' added '2d ago'. The second row, which is highlighted with a light blue background, shows 'Driver Installation' under 'Software' added '1w ago'. The third row shows 'DNS Setup' under 'Communication' added '17h ago'.

TITLE	CATEGORY	ADDED
Monitor Issues	Hardware	2d ago
Driver Installation	Software	1w ago
DNS Setup	Communication	17h ago

Figure 5(a).1

**Use Case 6: Assign open issues to themselves**

To assign an open issue to himself, the technician will navigate to the 'Check all Issues' screen, i.e. figure 5(a).1, by clicking on the 'Check all Issues' button on the homescreen, i.e. figure 1.2. Once he reaches there, he can click on any issue from the list of issues logged and he will be directed to the 'Take an Issue' screen, i.e. figure 6.1. This screen will contain all the details of the issue and if a technician decides to take up that issue then he can click on the 'Take' button at the bottom of the screen. The task will be assigned to him and he will see a pop-up notification of 'Taken up Successfully!', i.e. figure 6.2.

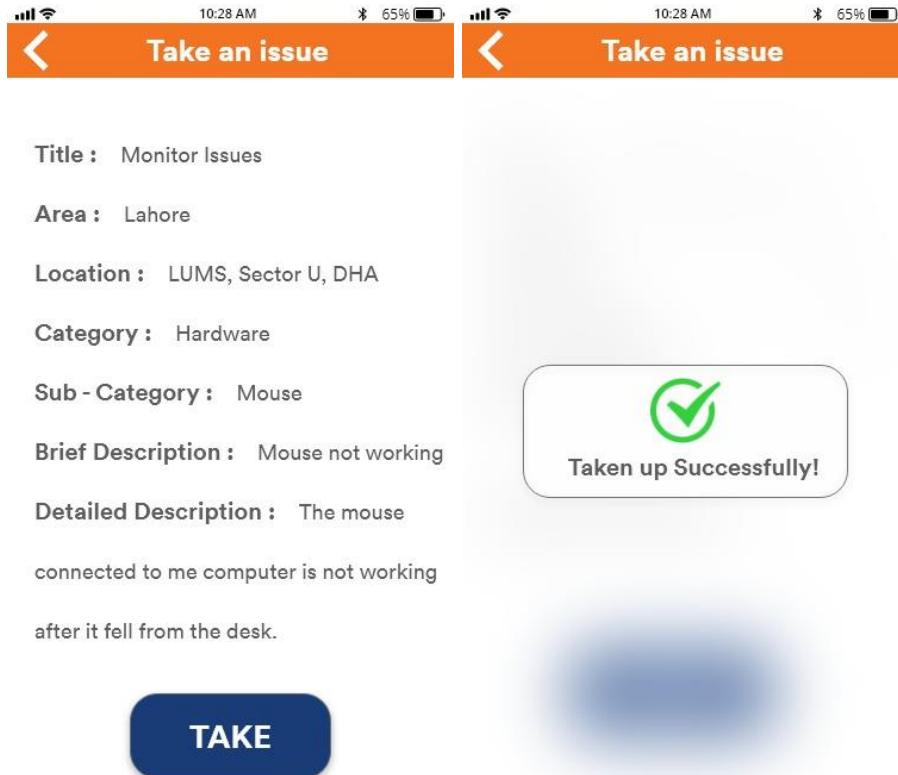


Figure 6.1

Figure 6.2



**Use Case 7: Mark resolved issues as fixed**

To mark a resolved issue as 'fixed', the technician will click on the 'More' button from the homescreen, i.e, figure 1.2. He will be then directed to a new screen with 'Issues Taken' and 'Logout' button, i.e. figure 7.1. Upon clicking the 'Issues Taken' button, 'View Status' screen, i.e. figure 7.2, will open and he will be able to see all the issues he has taken up. He will then click on the issue, he wants to close. A new 'View Status' screen, i.e. figure 7.3, will open and the technician can click on the 'Mark as Fixed' button to mark the issue as fixed. He will also then receive a 'Mark as fixed!' pop-up notification, i.e. figure 7.4.

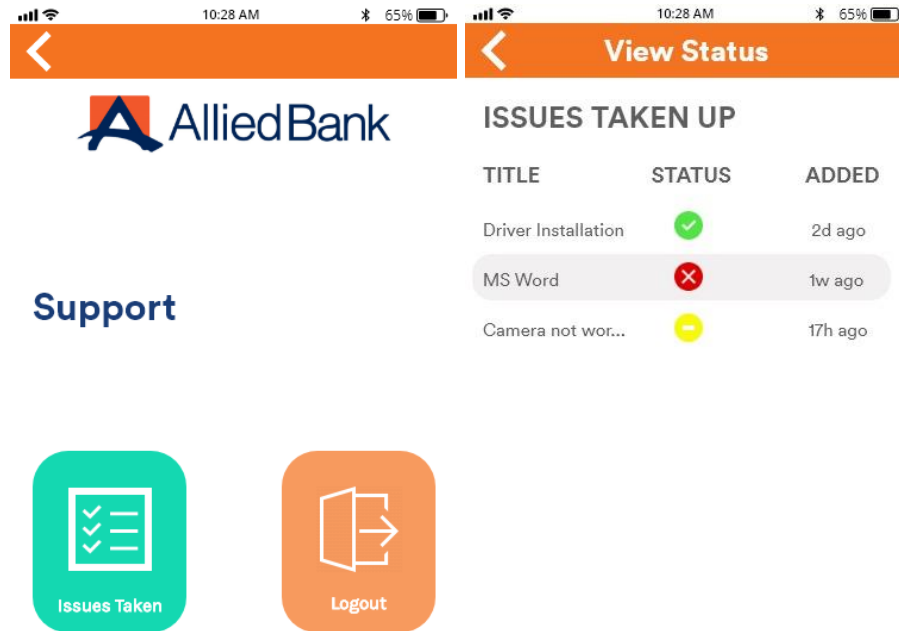


Figure 7.1

Figure 7.2

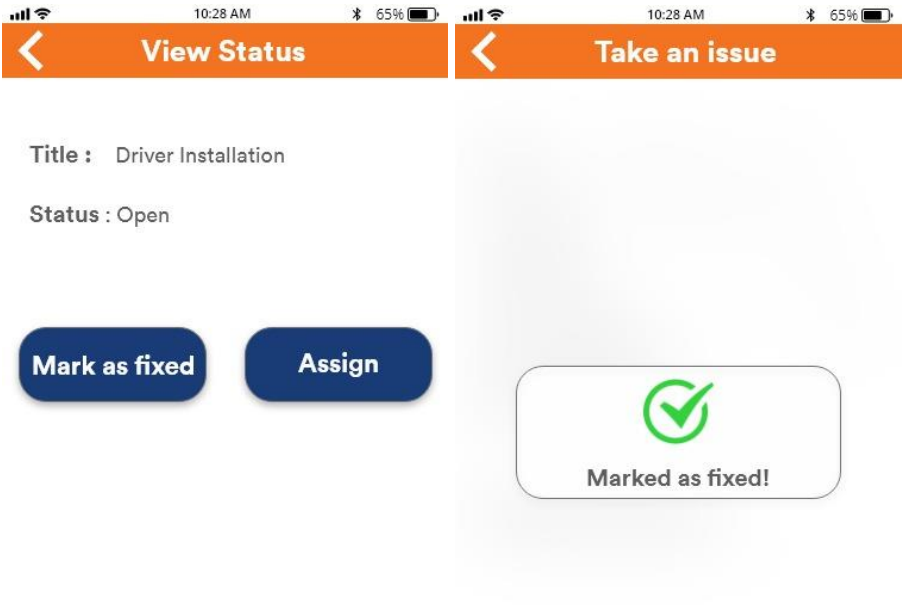


Figure 7.3

Figure 7.4

**Use Case 8: Reopen resolved issues**

To reopen a resolved issue, the user will navigate to the 'View Status' screen by first clicking on the 'More' button on the homescreen, i.e. figure 1.2 and then on the 'Issues Taken' button from the screen corresponding to figure 7.1. Once there, the user can select a closed issue by clicking on it and he will be directed to a new 'View Status' screen, i.e. figure 8.1. There he will click on the 'Mark as open' button to reopen the issue. He will receive the 'Marked as open!' pop-up notification afterwards, i.e. figure 4.2.

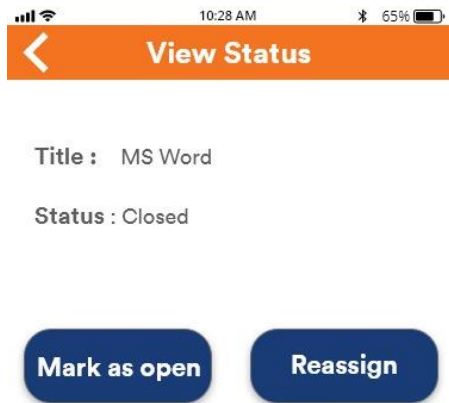


Figure 8.1

**Use Case 9: Reassign issues to other team members**

To reassign issues, previously taken up by himself, to other team members, the user will navigate to the 'View Status' screen by first clicking on the 'More' button on the homescreen, i.e. figure 1.2 and then on the 'Issues Taken' button from the screen corresponding to figure 7.1. Once there, the user can select an issue by clicking on it and he will be directed to a new 'View Status' screen, i.e. figure 8.1. There he will then click on the 'Reassign' button and he will be directed to the 'Assign' screen, i.e: figure 9.1. This screen will contain a list of technicians. The user can select the new technician by either selecting a person from the list or by typing his name and employee ID in the 'Name' and 'Employee ID' fields and searching for him by clicking the 'Search' button. Once the user selects the new technician, a new 'Assign' screen, i.e. figure 9.2, will open with the details of the technician. He can then click on the 'Assign' button to reassign the issue to that technician. In the end he will receive a 'Successfully Assigned!' pop-up notification, i.e. figure 9.3.

The image shows a mobile application interface for assigning roles. At the top, there is a status bar with signal strength, time (10:28 AM), and battery level (65%). Below the status bar is an orange header bar with a back arrow and the word "Assign". The main content area is divided into two columns. The left column has two headers: "Name" and "Role". Below these headers is a list of employees: Aadam Nadeem (Support), Maleeha Masood (Support), Malik Ali Hussain (Support), Raheem Zafar (Support), and Shahrukh Kamaal (Manager). The right column has three labels: "Name : Aadam Nadeem", "Role : Support", and "Employee ID : 123456". Below these labels is a blue button labeled "Assign". At the bottom of the screen, there are two input fields: "Name : \_\_\_\_\_" and "Employee ID : \_\_\_\_\_", followed by a blue button labeled "Search".

Name	Role
Aadam Nadeem	Support
Maleeha Masood	Support
Malik Ali Hussain	Support
Raheem Zafar	Support
Shahrukh Kamaal	Manager

Name : Aadam Nadeem  
Role : Support  
Employee ID : 123456

Assign

Name : \_\_\_\_\_  
Employee ID : \_\_\_\_\_

Search

Figure 9.1

Figure 9.2

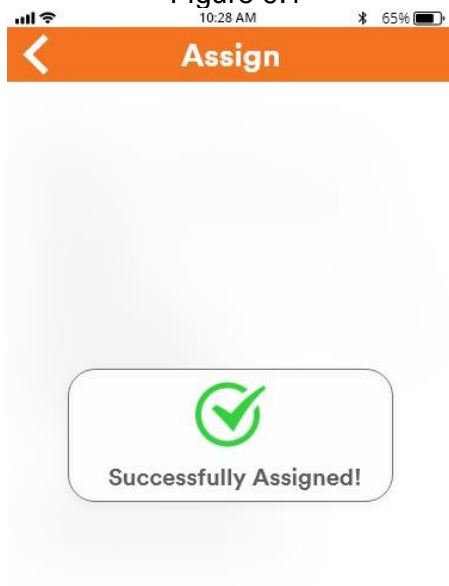


Figure 9.3

**Use Case 10: Logout from the application**

To Logout from the application, the user should click on the 'More' button from the homescreen, i.e. figure 1.2. The user will be directed to a screen with 'Issues Taken' and 'Logout' button, i.e. figure 7.1. Upon clicking the 'Logout' button, the user will be logged out and redirected to the Login screen, i.e. figure 1.1.

**Manager Use Cases****Use Case 1: Login using credentials**

Same as the Use Case 1 for Initiator

**Use Case 2: Log support related issues**

Same as the Use Case 2 for Initiator

**Use Case 3: View status of own issue**

Same as the Use Case 3 for Initiator

**Use Case 4: Mark own issue as closed or open it again**

Same as the Use Case 4 for Initiator

**Use Case 5: Assign open issues to themselves**

Same as the Use Case 6 for the Support Team Member

**Use Case 6: View Status of Issues pending with their team**

To view the status of the issues taken up by his team, the manager will click on the 'More' button from the homescreen, i.e. figure 1.2. He will then be directed to a new screen with 'All Issues', 'Print PDF' and 'Logout' button, i.e. figure 6(a).1. Upon clicking the 'All Issues' button, 'View Status' screen, i.e. figure 6(a).2, will open and he will be able to see all the issues he and his team has taken up and their status.

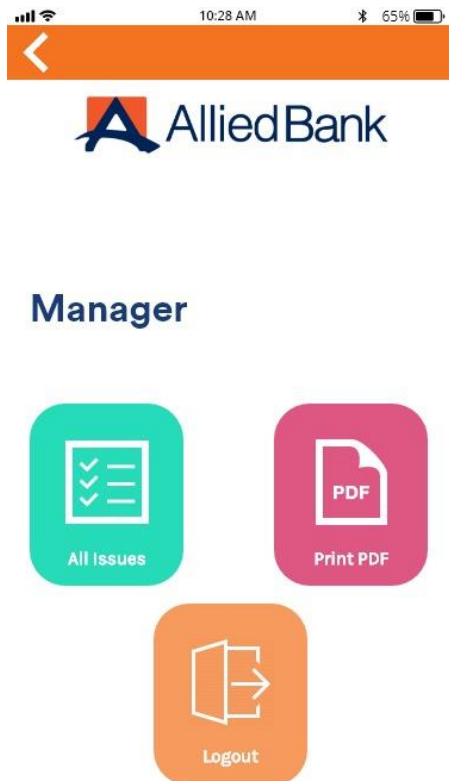


Figure 6(a).1

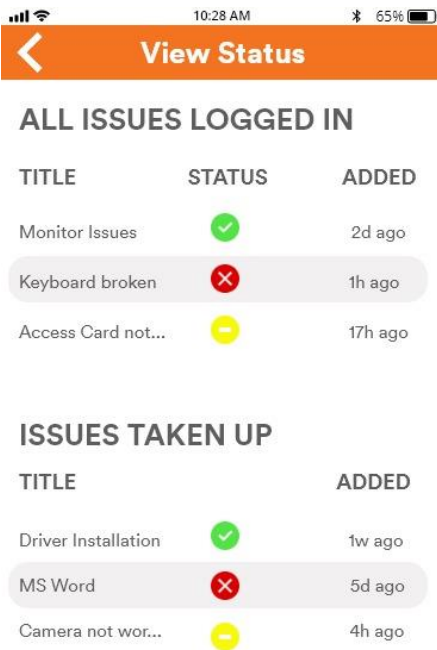


Figure 6(a).2

**Use Case 7: Mark resolved issues as fixed**

Same as the Use Case 7 for the Support Team Member

**Use Case 8: Refer Issue to other Support Team**

To refer an issue to other support team, the manager will click on the 'More' button from the homescreen, i.e. figure 1.2. He will then be directed to a new screen with 'All Issues', 'Print PDF' and 'Logout' button, i.e. figure 6(a).1. Upon clicking the 'All Issues' button, 'View Status' screen, i.e. figure 6(a).2, will open and he will be able to see all the issues his team and he himself has taken up and their status. He will then click on the issue to be referred to the other team and he will be directed to a new 'View Status' screen, i.e. figure 7.3. He will then click on the 'Reassign' button and the user will be directed to the screen, i.e. figure 8(a).1, where he can choose whether he wants to assign the issue to another team or his other team member. The user will then click on the 'Reassign to other team' button. Upon clicking this button, a 'Assign' screen will open, i.e. figure 8(b).2, with the 'Name' and 'Employee ID' fields. After typing in the the name and employee ID in the respective field, he will then click on the 'Search' button for searching the employee with the entered credentials in the database. If such employee exists then, a new 'Assign' screen, i.e. figure 9.2, will open with the details of the technician. He can then click on the 'Assign' button to reassign the issue to that technician. In the end he will receive a 'Successfully Assigned!' pop-up notification, i.e. figure 9.3.

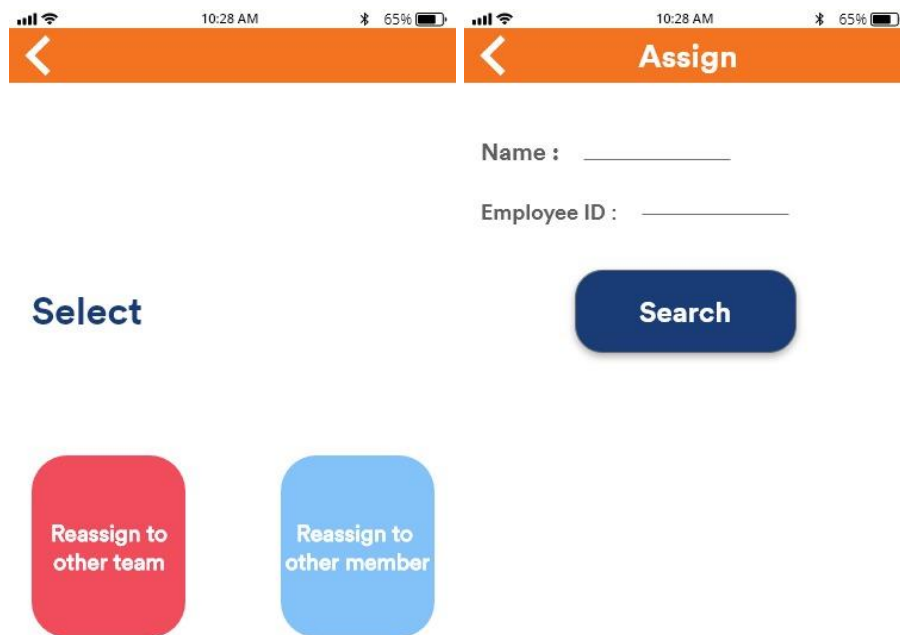


Figure 8(a).1

Figure 8(a).2

**Use Case 9: Assign/Re-assign Issue to other team members**

To assign/re-assign an issue to other team members, the manager will click on the 'More' button from the homescreen, i.e. figure 1.2. He will then be directed to a new screen with 'All Issues', 'Print PDF' and 'Logout' button, i.e. figure 6(a).1. Upon clicking the 'All Issues' button, 'View Status' screen, i.e. figure 6(a).2, will open and he will be able to see all the issues his team and he himself has taken up and their status. He will then click on the issue to be referred to the other team and he will be directed to a new 'View Status' screen, i.e. figure 7.3. He will then click on the 'Reassign' button and the user will be directed to the screen, i.e. figure 8(a).1, where he can choose whether he wants to assign the issue to another team or his other team member. The user will then click on the 'Reassign to other member' button. He will be directed to the 'Assign' screen, i.e. figure 9.1. This screen will contain a list of technicians. The user can select the new technician by either selecting a person from the list or by typing his name and employee ID in the 'Name' and 'Employee ID' fields and searching for it by clicking on the 'Search' button. Once the user selects the new technician, a new 'Assign' screen, i.e. figure 9.2, will open with the details of the technician. He can then click on the 'Assign' button to reassign the issue to that technician. In the end he will receive a 'Successfully Assigned!' pop-up notification, i.e. figure 9.3.

**Use Case 10: Monitor Reports**

To generate the report of the issues taken up by his team, the manager will click on the 'More' button from the homescreen, i.e. figure 1.2. He will be then directed to a new screen with 'All Issues', 'Print Pdf' and 'Logout' button, i.e. figure 6(a).1. Upon clicking the 'Print PDF' button, a pdf file will be downloaded on the device containing all the issues and their details.

**Use Case 11: Logout from the application**

To Logout from the application, the user should click on the 'More' button from the homescreen, i.e. figure 1.2. The user will be directed to a screen with 'All Issues', 'Print PDF' and 'Logout' button, i.e. figure 6(a).1. Upon clicking the 'Logout' button, the user will be logged out and redirected to the Login screen, i.e. figure 1.1.

**Monitor Use Cases****Use Case 1: Login using credentials**

Same as the Use Case 1 for Initiator

**Use Case 2: Log support related issues**

Same as the Use Case 2 for Initiator

**Use Case 3: View status of own issue**

Same as the Use Case 3 for Initiator

**Use Case 4: Mark own issue as closed or open it again**

Same as the Use Case 4 for Initiator



**Use Case 5: Monitor Reports**

Same as the Use Case 10 for Manager

**Use Case 6: Logout from the application**

Same as the Use Case 11 for Manager

**Admin Use Cases****Use Case 1: Login using credentials**

Same as the Use Case 1 for Initiator

**Use Case 2: Log support related issues**

Same as the Use Case 2 for Initiator

**Use Case 3: View status of own issue**

Same as the Use Case 3 for Initiator

**Use Case 4: Mark own issue as closed or open it again**

Same as the Use Case 4 for Initiator

**Use Case 5: Change Role**

To change the role of a technician, the admin will click on the 'More' button from the homescreen, i.e. figure 1.2. He will be then directed to a new screen with 'Change Team', 'Change Roles' and 'Logout' button, i.e. figure 5(b).1. Once the user clicks on the 'Change Roles' button, he will be directed to the 'Change Roles' screen, i.e. figure 5(b).2. This screen will contain a list of technicians. The user can select the new technician by either selecting a person from the list or by typing his name and employee ID in the 'Name' and 'Employee ID' fields and searching for it by clicking on the 'Search' button. Once the user selects the technician, a new 'Change Role' screen, i.e. figure 5(b).3, will open with the previous details of the technician and a 'New Role' field. He will then select the new role from the dropdown list by clicking on the button placed adjacent to the 'New Role' field. After selecting the new role, he will then click on the 'Change Role' button and receive a pop-up notification 'Role Changed!', i.e. figure 5(b).4, as the role of the technician is changed.

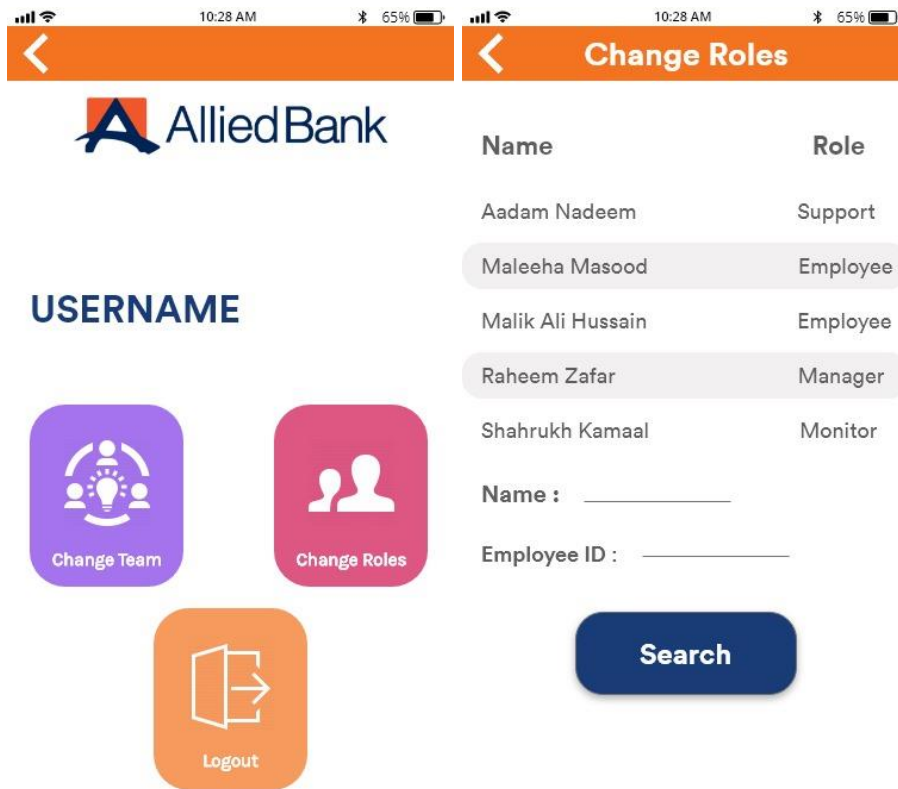


Figure 5(b).1

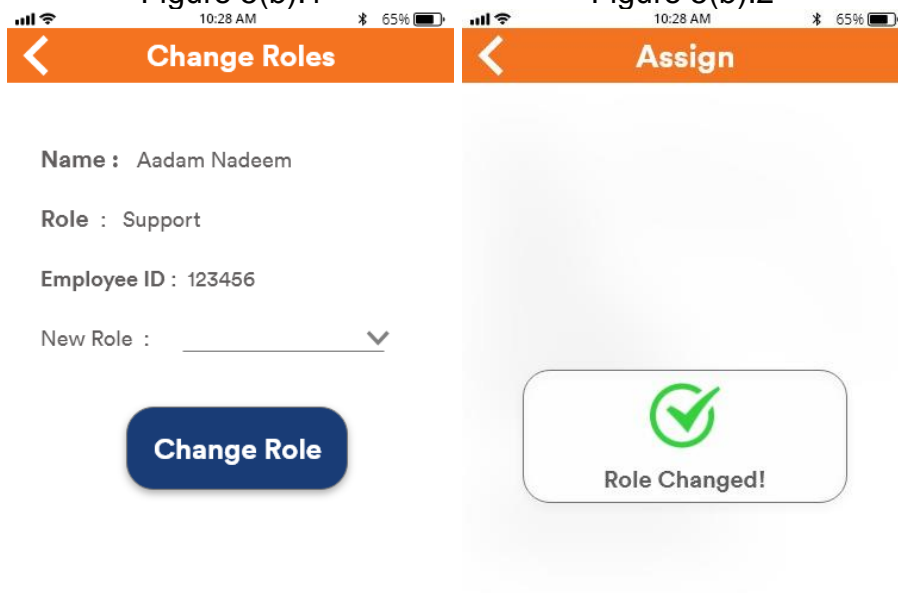


Figure 5(b).3

Figure 5(b).4

**Use Case 6: Change Team**

To change the team of a technician, the admin will click on the 'More' button from the homescreen, i.e. figure 1.2. He will be then directed to a new screen with 'Change Team', 'Change Roles' and 'Logout' button, i.e. figure 5(b).1. Once the user clicks on the 'Change Team' button, he will be directed to the 'Change Teams' screen, i.e. figure 6(b).1. This screen will contain a list of technicians. The user can select the new technician by either selecting a person from the list or by typing his name and employee ID in the 'Name' and 'Employee ID' fields and searching for it by clicking on the 'Search' button. Once the user selects the technician, a new 'Change Teams' screen, i.e. figure 6(b).2, will open with the previous details of the technician and a 'New Team' field. He will then select the new team for the technician from the dropdown list by clicking on the button placed adjacent to the 'New Team' field. After selecting the new team, he will then click on the 'Change Team' button and receive a pop-up notification 'Team Changed!', i.e. figure 6(b).3, as the team of the technician is changed.

The figure consists of two side-by-side screenshots of a mobile application interface titled 'Change Teams'.

**Figure 6(b).1 (Left Screenshot):** The screen displays a list of technicians with their names and roles. The 'Maleeha Masood' row is highlighted. Below the list are input fields for 'Name' and 'Employee ID', and a blue 'Search' button.

Name	Role
Aadam Nadeem	Hardware
Maleeha Masood	Software
Malik Ali Hussain	Communication
Raheem Zafar	General
Shahrukh Kamaal	Software

Name : \_\_\_\_\_  
Employee ID : \_\_\_\_\_  
**Search**

**Change Teams**

**Change Team**

Figure 6(b).1

Figure 6(b).2

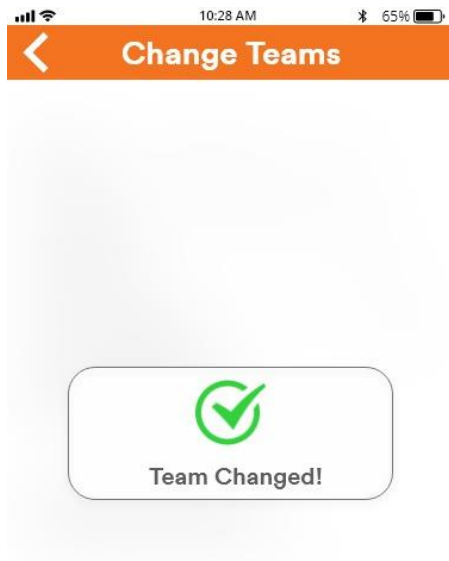


Figure 6(b).3

### **Use Case 7: Logout from the application**

To Logout from the application, the user should click on the 'More' button from the homescreen, i.e. figure 1.2. The user will be directed to a screen with 'Change Team', 'Change Role' and 'Logout' button, i.e. figure 5(b).1. Upon clicking the 'Logout' button, the user will be logged out and redirected to the Login screen, i.e. figure 1.1.

## **5.4 User interface design rules**

Following UI principles were used in designing the UI:

- **Structure:** The UI was designed in such a way as to keep the relevant functions of a component separate from the irrelevant functions of other components. A user will only be concerned with functions related to his/her role.
- **Clarity:** The overall visibility of the interface is simplified such that every component and its function is clear to the user before using it. Every path and exit is easily understandable and accessible to the average user.
- **Response:** The system is responsive to user actions and provides immediate feedback to the user in case of function completion or occurrence of errors. Design dialogs are used at every task completion to yield closure for the user.
- **Consistent:** The UI of the system is designed in a way that it reuses some of the screens for similar functions for the component in another role. Almost all of the screens follow the same theme and basic design. Identical terminology is used throughout the interface and consistent commands are used throughout.

- **Simplicity:** The UI is very easy to follow for an average user, with error messages in case of a misdirection. Components are clear to a user irrespective of the fact that he/she has technical knowledge or not.
- **Error Handling:** The system displays issues in clear words in case of an error to help the user understand the mistake.

## 6 Other Non-functional Requirements

### 6.1 Performance Requirements

- The application should be compatible with both android and IOS to ensure that it is accessible to all users. Flutter toolkit is being used to develop the application to make it easier for developments in both platforms, ios and android.
- If the issue is not addressed within 48 hours, it should be escalated automatically to the manager to ensure timely resolution of the problem.
- The issue reported must be stored in the database and should be visible to the respected support team within 120 seconds after logging an issue. Which means that the time to store data in the database + loading time for a new user + time to fetch data from the database for the new user, should not be more than 120 seconds. To reduce query and processing times we will be experimenting with efficient indexing techniques and caching results where possible to reduce latency and avoid bottlenecks.
- The issue resolved must be stored in the database and visible to the initiator within 120 seconds after the issue is marked and fixed by the technician. The implementation and techniques to achieve this are also the same as the requirement stated before.
- The fixed issue must be notified to the initiator within 120 seconds after the issue is marked fixed by the support team. The implementation and techniques to achieve this are also the same as the requirement stated before and an additional functionality of a notification generation will be added in this part.
- Any updates to the system should take no more than 120 seconds. Any changes made to the issue or employee database must be visible on the application in 120 seconds which will include time to write the database and to fetch data as well. We will be attempting to achieve this by caching and rails again to improve indexing and traversal through the database.
- Emails should be sent with a latency of no greater than 12 hours after each activity.

Time constraints have been applied to ensure consistency throughout all users using the application

### 6.2 Safety and Security Requirements

- **Use of dummy server:**  
The client has refused to authorize access to the server where their existing web ILS is running to ensure safety and security of the bank's server and other confidential details they can not afford to outsource. We are required to set up a dummy server for testing and running the application. Once the client approves the application, the Bank's IT department will themselves replace the dummy server with their own server.
- **Use of dummy database:**

The client has refused to authorize access to the database where the Bank's existing web ILS is running, this is to assure safety and security of employees personal information. Client has agreed to provide us with the structure of their existing database and using that structure we will design a dummy database which our application will use for running and testing, once the client approves the application, the bank's IT department will themselves replace the dummy database and run the application with their own database.

- **Use of universal login by client:**

To ensure that only the authorized personnel i.e. the bank's employees use the application and no other person gets authorized to login, the client wants the ILS application to login only via the employee's universal login credentials that every ABL Bank's employee is assigned for all logins. To cater this requirement, our application will not have a sign up option, only a sign in option will be given that will use only the employees universal login credentials stored in banks database for a successful sign in into the ILS application. Our system will only be able to fetch data from the database but will not be able to write to the database to add a new record. The user will only be able to use the application once he is permitted after entering right credentials that match the employee database.

## 6.3 Software Quality Attributes

- **6.3.1 Correctness:**

The ILS application for ABL needs to be free from any error considering the users of the application. We aim to assure correctness because users are not expecting any errors. We aim to achieve that by multiple accuracy tests being conducted at 4 phases.

- **Phase 1:** During the development phase, after adding each feature to the application, the developer will assure that the desired output is achieved.
- **Phase 2:** After completing the front-end development, our developers will test the application for any errors.
- **Phase 3:** After application is complete, our entire team will try and test the application while catering any unexpected output issue that any team member observes.
- **Phase 4:** A beta version will be provided to the client to test if users are facing any undesired output.

- **6.3.2 Portability:**

Our system is an attempt to add portability to the bank's ILS and we ensure this by designing an application which can be used on the go. Our goal is to ensure maximum portability as well as reliability by reducing latency and processing time so that the application can be used anywhere anytime. Developing a multi platform application using Flutter will also ensure portability of the application.

- **6.3.3 Adaptability:**

We aim to develop an application that performs well and gives the same user experience to users of all IOS and android platforms. We will assure this by using a platform modifiable code base through flutter for the development phase of the application which means that the application can easily be adapted to adjust to any platform.

- **6.3.4 Usability:**

Once the beta version of our application is ready, the users will be asked for feedback and improvements will be made if needed to assure the best user friendly experience for the users. We ensure usability by using a GUI filled with graphics and colors to make our application more usable. For the final version we will be providing a document with instructions to make the user experience easy and the shift smooth.

- **6.3.5 Availability:**

To ensure availability, we are using cloud database i.e. ClearDB as an add-on by Heroku. This is because ClearDB is designed around the idea that systems, networks, and storage fails from time to time, which under ordinary circumstances can leave your database offline. If the system that has failed is back online and is back in sync, CloudDB routing technology will automatically switch back to the newly recovered server instance so that you get the lowest latency and highest performance which will be ideal for the ILS Application

- **6.3.5 Scalability:**

We will achieve scalability when we use a cloud database for our ILS Application. There will be no limitation on the amount of data that can be stored in the database, no matter how many issues have been logged or how many new employees are added, our database performance will not slow down and our application will run smoothly.



## Appendix A - Group Log

- Before the spring break, we attended all our regular meetings with our primary TA, Taimoor. Below are the minutes of the few meetings we had after SRS and before the spring break:
- 2nd March 2020 MoM: We had a few questions about the environment to work in. Discussed general confusions about the SDS document.
- 9th March 2020 MoM: We discussed our plan on dividing work.
- 23rd March 2020 MoM: We discussed SRS marks and how to rectify our mistakes for SDS.
- Additionally, we have been in continuous touch with the course staff regarding confusions that we had.
- All of us have also been in constant touch with each other most of the spring break to keep track of each other's progress, provide feedback, discuss parts, division of work and for the final compilation of this document. We had conference calls every other day for most of the spring break where we discussed updates and work left to be done.

**Appendix B – Contribution Statement**

<i>Name</i>	<i>Contributions in this phase</i>	<i>Approx. Number of hours</i>	<i>Remarks</i>
<i>Aadam Nadeem</i>	<i>Section 5</i>	<i>28</i>	<i>Outstanding</i>
<i>Maleeha Masood</i>	<i>Sections 1, 2, 3 and the Appendixes</i>	<i>25</i>	<i>Outstanding</i>
<i>Malik Ali Hussain</i>	<i>Section 5</i>	<i>25</i>	<i>Outstanding</i>
<i>Muhammad Raheem Zafar</i>	<i>Sections 4.1, 4.2, 6</i>	<i>25</i>	<i>Outstanding</i>
<i>Shahrukh Kemall</i>	<i>Sections 4.3 - 4.5, 6</i>	<i>25</i>	<i>Outstanding</i>