**I3301 – I3350 – I3302**

**Freelancer Application**

**Mohammad Houssein Fleity 58688**

**Guitta Jamil Moubarak 58926**

Summary

Contents

[1- Introduction 3](#_Toc129702324)

[2- Analysis 3](#_Toc129702325)

[2-1- Business Rules 3](#_Toc129702326)

[2-2- Use case 3](#_Toc129702327)

[2-3- Class diagram 3](#_Toc129702328)

[2-4- Activity Diagram 3](#_Toc129702329)

[2-5- Sequence Diagram 3](#_Toc129702330)

[2-6- Screens description 3](#_Toc129702331)

[3- Implementation 4](#_Toc129702332)

[3-1- Technical environment used 4](#_Toc129702333)

[4- Conclusion 4](#_Toc129702334)

Introduction

The Freelancer:

An application that aims to make people's life easier by helping customers to reach among the wanted area the best service with the best rank.

this is accomplished by letting Freelancers and customers create an account with the needed information. Once the profile is created, Freelancers (Carpenter, plumbers…) can add their services and add geographical information, contact information (email, phone number), brief description and images.

While customers search for these services, by specifying the service type and the aimed area. As soon as the search begin results are displayed with the average rate for each service, plus that customers can easily reach the contact info of the freelancer as the geographical location using the link location added by the freelancer. After adding the services customers can rate them and add their feedbacks.

It's also possible for a user to report malicious accounts, making the app self-controlled by users.

In the coming parts of this report, we will be analyzing the main parts of the class diagram, the use cases, the main sequence diagrams and the activity diagram of this application which we have developed in Android and PHP implementing the MVC model

Analysis

Business Rules

User can:

* Create a profile (freelancer or a customer,),
* Search for a service (carpenter, plumber, ...) in a specific area,
* Rate a service and add feedback,
* Report a freelancer.
* Freelancer add services.

Admin can:

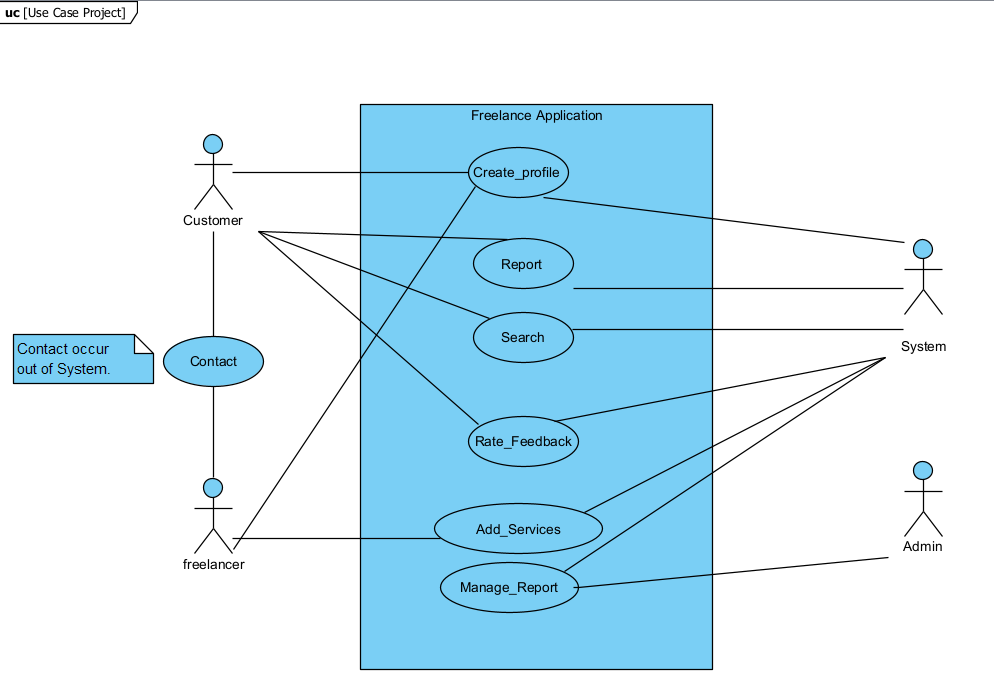
* Block a freelancer,
* Manage the system.

System:

* System controls the number of reports for customers and freelancers.

Use case

* Use case diagram.



* Scenarios:

**Use Case:** Create Profile**.**

**Actors:** Customer, freelancer and system**.**

**Goal:** to Create a profile as a customer or a freelancer.

**Pre-condition:** wants to offer a service or needs a service.

**Description:** The user chooses if**:**

* + - he’s a customer then, inputs his name, birthdate, gender, his email as a username, a strong password with confirmation,
    - he’s a freelancer then, inputs his name, phone number, birthdate, gender, location, email as a username, a strong password with confirmation,

While the user is inputting his info the System checks if his email is free to be occupied, his password is strong or weak, if it’s confirmed correctly and if his age>18 else the profile would not be created.

**Post-condition:** Profile created and inserted in the database plus a message informing the user that his profile has been successfully created.

**Use Case:** Rate\_Feedback.

**Actors**: Customer, freelancer and system

**Goal:** Rate the service of a freelancer and leave feedback so that other customers can see it.

**Pre-condition:** the customer is already signed in.

**Description:**

* + Customer hits freelancer profile clicks the 3-points option,
  + From the menu customer chooses rate,
  + System displays a 5-stars rate to choose from and an optional textbox,
  + After rating the freelancer, customer either adds feedback then submits or submits directly,
  + System assigns the rate to the freelancer profile.

**Post-condition:** The feedback / rate is stored by the system in the database and is shown on the profile of the freelancer.

**Use case:** Search

**Actors:** customer, system.

**Goal:** customer find the best service offer for his needs.

**Pre-condition:** the customer signs in successfully.

**Description:**

* Customer chooses the service needed from a drop-down list,
* Customer chooses the area in order: governorate then the village/street near him,
* System displays corresponding service providers by descending order of rates with latest feedbacks,
* Customer chooses the freelancer whose service fits him the best,
* Clicks on the profile; the system opens the freelancer’s profile that includes contact info and location.

**Post-condition:** the customer finds his target.

**Use Case:** Report

**Actors:** Customer, freelancer and system.

**Goal:** report an inappropriate user.

**Pre-condition:**

* + The customer is already signed in.
  + The customer hasn’t passed his limit of reports.
  + He hasn’t reported this freelancer account before.

**Description:**

* + The customer has attested a bad behavior from a freelancer or an inappropriate content,
  + Goes to the freelancer profile, clicks on more/3-points,
  + System displays options including report option,
  + Customer clicks report,
  + System displays Required predefined dropdown list of reports,
  + Customer chooses a reason for the report and hits submit,
  + System adds the report to the database.

**Post-condition:** the freelancer is reported to the system.

**Use Case:** Expose\_Service\_By\_Photos**.**

**Actors:** freelancer and system.

**Goal:** toexpose his work.

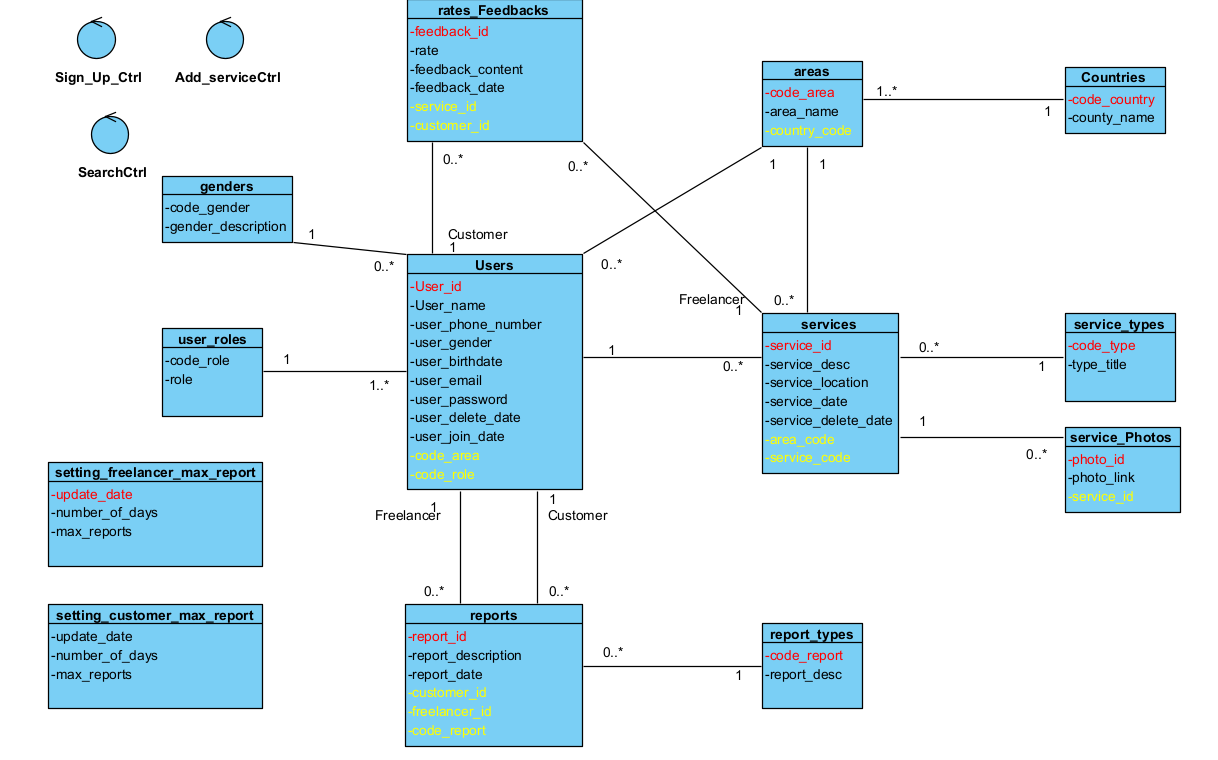
**Pre-condition:** the freelancer is already signed in.

**Description:**

* + in his profile the freelancer clicks on albums and adds from his gallery the photos of his latest work.

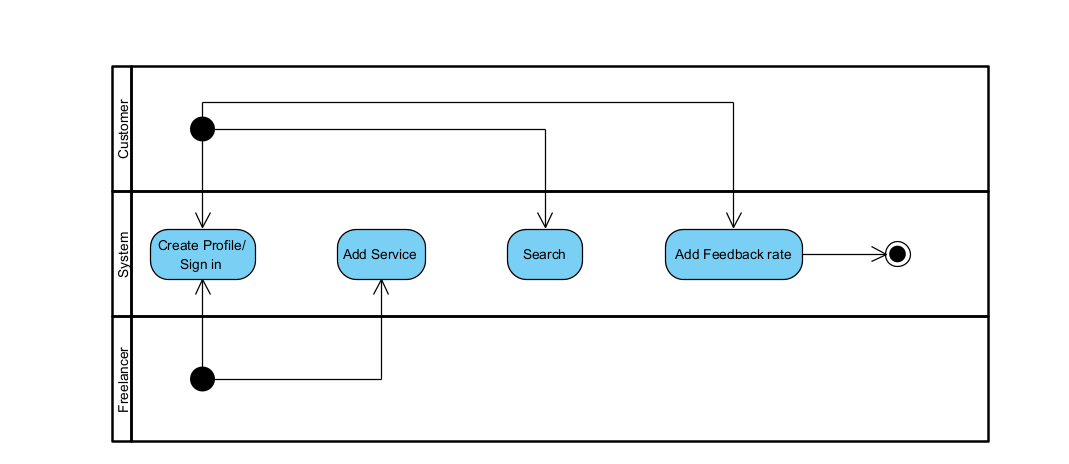
**Post-condition:** Photos are added to profile and database, customers can see them.

Class diagram



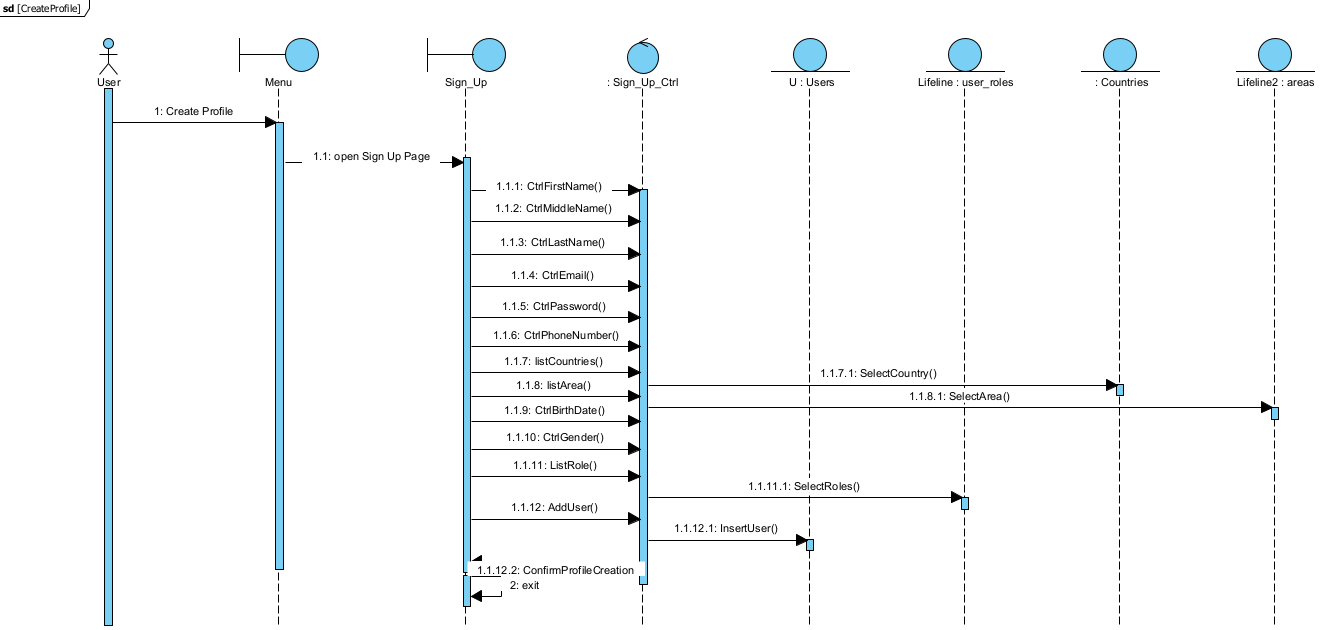
* Relation Users-areas: when singing up, the user chooses his location (in our case the area) that will be saved in the database and shown in his profile hence, we have a relation many to one between Users-areas.
* Relation areas-services: for each service a location is specified since a freelancer can have a personal location (in his profile) and another location for his services hence, we have a relation one to many between areas-services.
* Relation Users-rates\_Feedbacks: Users add feedbacks and rates to services offered by freelancers, each user can add many feedbacks, feedback is added by one and only one user.
* Relation rates\_Feedbacks-services: each service can have none or many feedbacks/rates and each feedback is concerned with one and only one service,

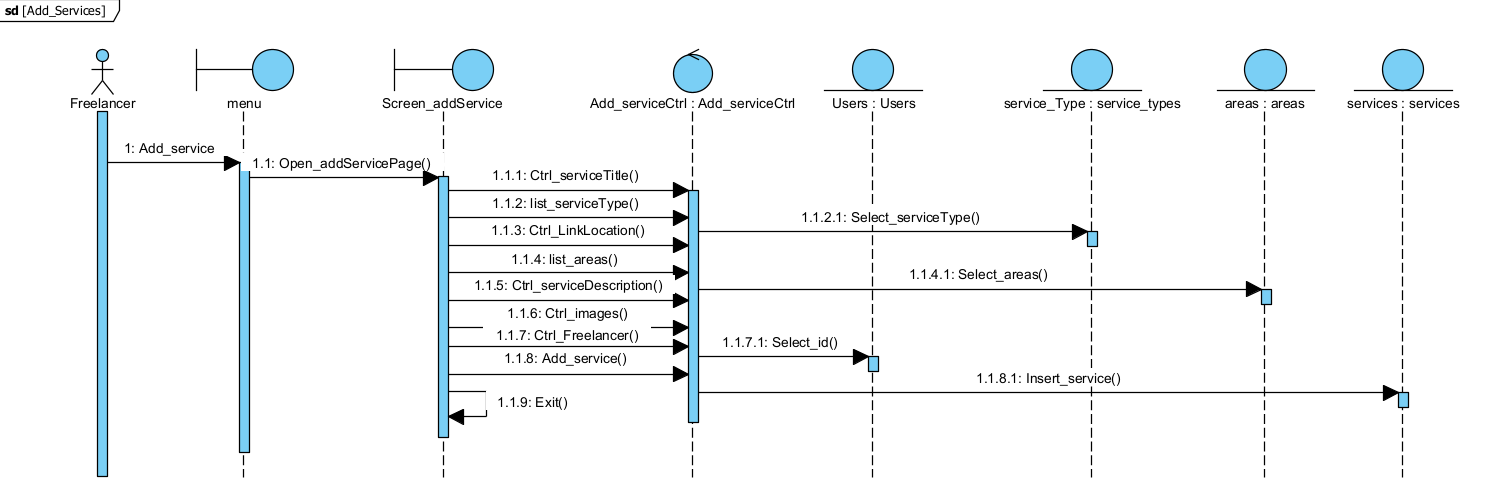
Activity Diagram



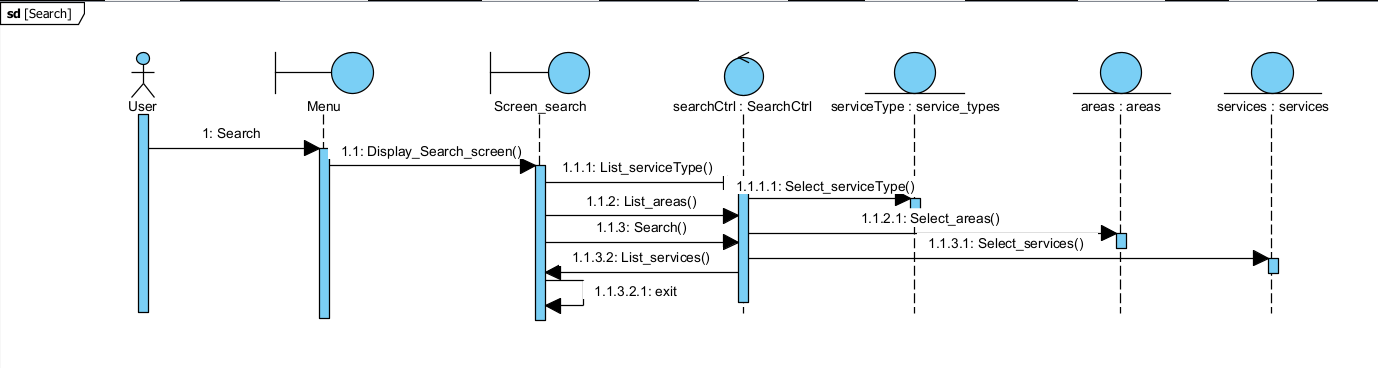
Sequence Diagram

Sequence Diagram for Create\_Profile:



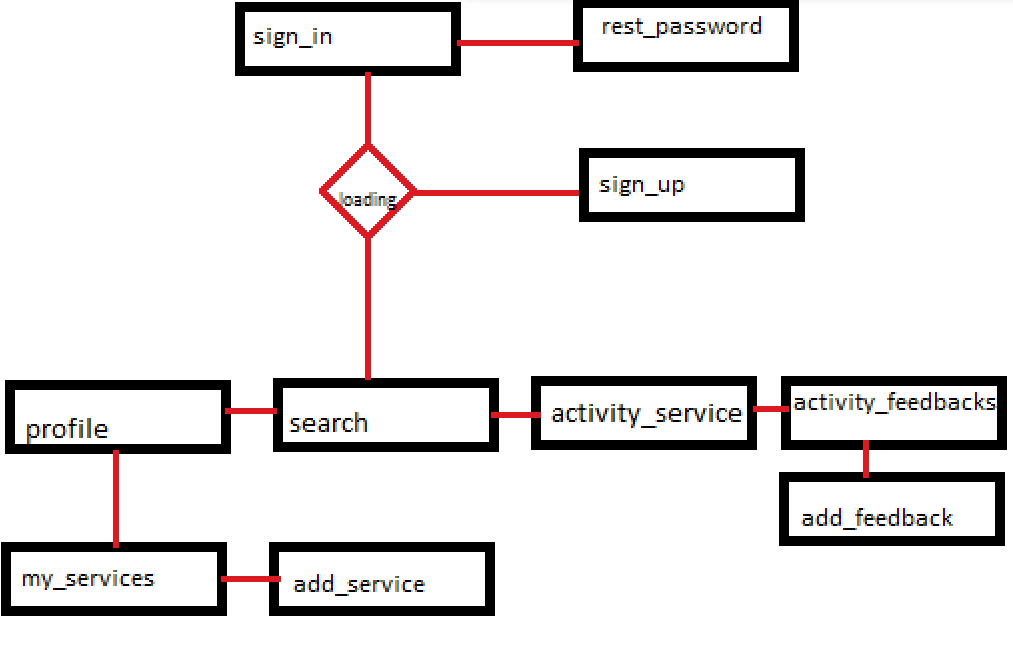
Sequence Diagram for Add\_Services:

Sequence Diagram for Search:



Screens description

* Tree of the screens/report and for each screen, the **controls** and the entity **classes** used.



* Loading: is a page created to import modification table from database (areas, service\_types, genders, user\_roles), to be able to use them lately and to avoid having repeted access to the database from the sign\_in page.
* Each interface has a control class, with four more classes three of them are for the entities (Feedbacks, Services, Users), and the last one concerned with the connection PHP Android containing the URL for PHP files.
* In PHP (website) the same tree is applied without the loading and services interfaces.
* In both the website and the application the navigation bar allow user to jump between the screens.

Implementation

Technical environment used

* For **PHP** technical environment:
  + Front-end:
    - Bootstrap 5.2.3 (CSS, JS).
    - HTML5.
  + Back-end:
    - PHP.
  + Servers and compilers: Visual Studio Code, XAMPP, WAMPP
* **Front-end**: For design most commonly and easy used libraries is Bootstrap especially after version NO.5 with the added JS scripts combined with HTML5 build our website and we used SASS and JSON in order to customize Bootstrap and to use icons offline. All mentioned before was used with the tension to save the most important criteria of the design simplicity and purple teem.
* **Back-end**: Connection between PHP and the local MySQL server is direct and no need for adapters as soon as we create a file called connection.php we implement within this file the method mysqli\_connect to get a MySQL object $conn used to connect to the database within the user called freelancer and the localhost domain name. We include once this connection whenever connection to the database is needed in other pages and we use the MySQL object created within this page to apply queries using mysqli\_query function and then fetching the result within mysqli\_fetch methods. once connection with database is no more needed, we close connection using the mysqli\_close function. User name and id are saved within a session than used directly in other pages and allow user to access other pages (user shouldn’t reach search page without logging in and having a $\_SESSION[‘id’] value).
* For **Android** technical environment:
  + Design:
    - XML
  + Back-end:
    - Java,
    - Volley 1.2.1,
    - PHP,
    - JSON.
  + Compilers: Android Studio, Visual Studio Code.
  + Emulator: Android Pixel 7, Google Chrome.
  + Testing: Postman Agent.
* **Design (**Front-end**)**: using XML tags and attributes the design of the mobile is lying under the topic of the simplicity and purple teem. As we tried to use unified sizes by predefining them in the *main/res/values/dimensions.xml* for padding, margin, and text size, as for colors which is defined in *main/res/values/colors.xml*. While for icons, backgrounds and borders it’s has been created within the folder *main/res/drawable* and accessed within the *@drawable/fil.xml* method. The bottom navigation bar was created using buttons with fixed ids and order as for functions (profile, search, back, Log-Out). But it’s not found in three pages (*sign-in.xml, sign-up.xml, reset-password.xml*)in order to prevent unwanted access to the app. Screens are detailed in the part **1.6 Screens description.**
* **Back-end:** the back-end was done using JAVA with minimum SDK 24 which allow 99% of android user to use the app. As Java handled the back-end of the app a library was needed to access the PHP files in order to access the database in the MySQL server, those we used VOLLEY 1.2.1 which act as an adapter between PHP output and java input and vis-versa. In java we initiate a request to PHP using the *URL* of the PHP file combined with the IP address, and the corresponding method (POST) the request send PHP data from android and use a response listener to get data back from PHP while there will be an error listener for handling error within the connection. On the other hand the PHP manage connection with database and output a JSON object containing the needed info for Android.
* **Implementing MVC:** consist of separating control screens from entities classes and for each entity creating a class, however in PHP it wasn’t as efficient and worthless to apply MVC as in Android, where we have created for the three main entities (Users, Services, Feedbacks) three Java classes with the corresponding fields and methods while for the rest of the entities, modification tables, since it was unnecessary to create for each one a class and it doesn’t affect the performance of the app we’ve created a hash map Mapping each entry in these tables to their corresponding id, as we’ve added a class for links which facilitate changing the links whenever the hosting changes, and we’ve created a limited connection user specified to the application.
* During implementation changing hosting may cause some problems which need a developer to fix the links. We can solve these problems without the need of the developer by adding an exterior file containing the link that can be changed by normal text editor and admins but still we risk crashing the links since it’s a sensitive operation.

Conclusion

In conclusion, we have developed the main screens and functions of this application respecting the previous analysis in Android and PHP implementing MVC and all use cases. leaving the doors open for upcoming updates which will include adding countries to the development part more control on the report part and developing a well back end to handle service photos as other updates should be applied to assure security and inhense performance of the application/web.