Software Architecture & Design (SAD) Document v.1.2

Project Name: Udrop (Vexit)

Date: June 10, 2020

Project Manager: Hanane Elkarni

Assistant Project Manager: Francys Cunanan

Software Engineer #1: Christopher Wise

Software Engineer #2: Daniel Firestone

Software Engineer #3: Ali Randell Kasim B Hamody

Software Engineer #4: Ronnie Espinosa

Software Engineer #5: Kelsey Burgos

Table of Contents

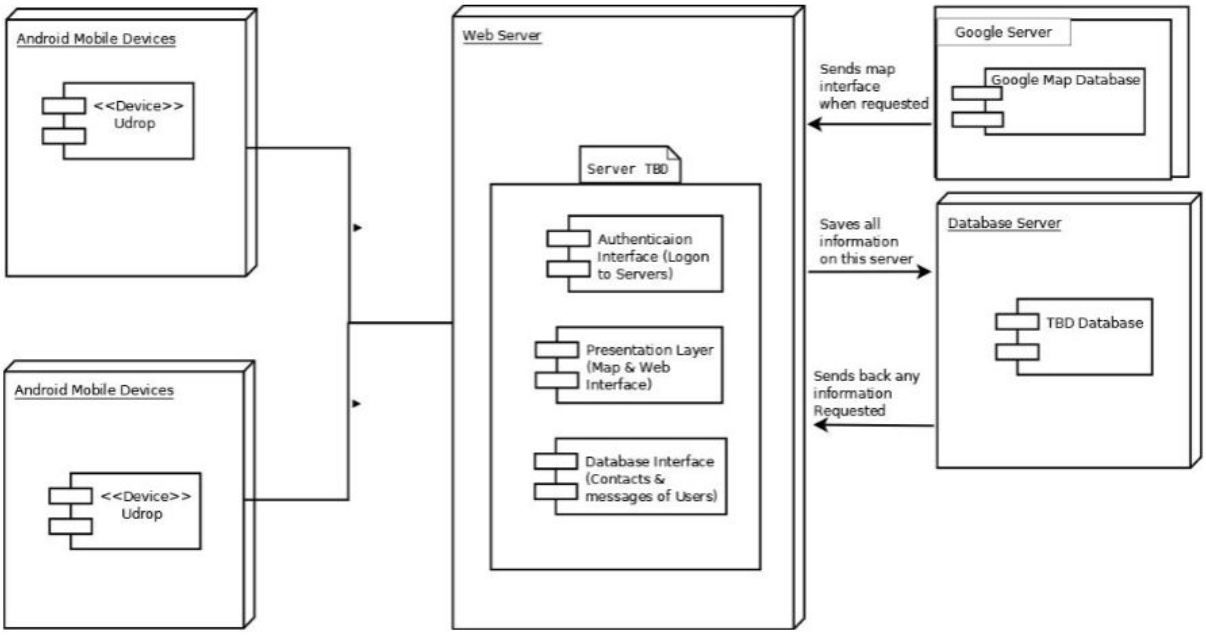
1. **Purpose**: The purpose for this document is to understand the importance and development of software architecture, design, and maintenance.

   Firstly, the development of *software architecture* is a system structure that defines behavior and conditions of the product. For instance, when activating the app the structure will check if the user's account is logged in or not. If it is, then it will go to home or return to the page where the app was last previously on. On the other hand, if it isn't then it will start with the page where you need to log in or create an account first. Software architecture is what helps use and navigate pages and classes, and without it the product will have many faults.

   Secondly, the development of *software design* is a product plan on how the product will turn out: what is its purpose, how it's going to do it, and what resources are required to make that happen. The difference between the two is that architecture focuses on the system structure like a blueprint, while design is a focus on the product's goal, risk analysis, and its software requirements.

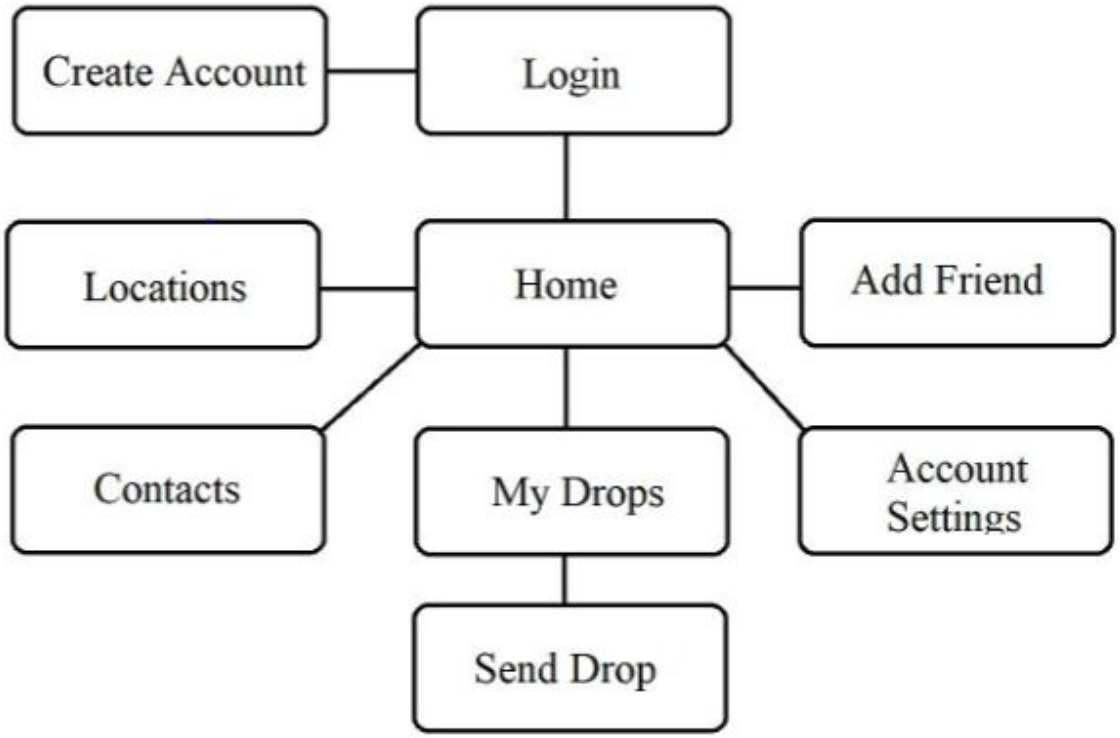
   Lastly, *software maintenance* is important because it focuses on the service provided with the product. This is important to note since it needs to take the responsibility of modifying the product even after its purchase; this is needed to correct faults after release and ensure consumers continue to purchase and use the company's products. Thus, maintenance is required for a SAD so we can check with the product's current architecture and design, so that it may be modified in the future.

2. **High-Level Design**:

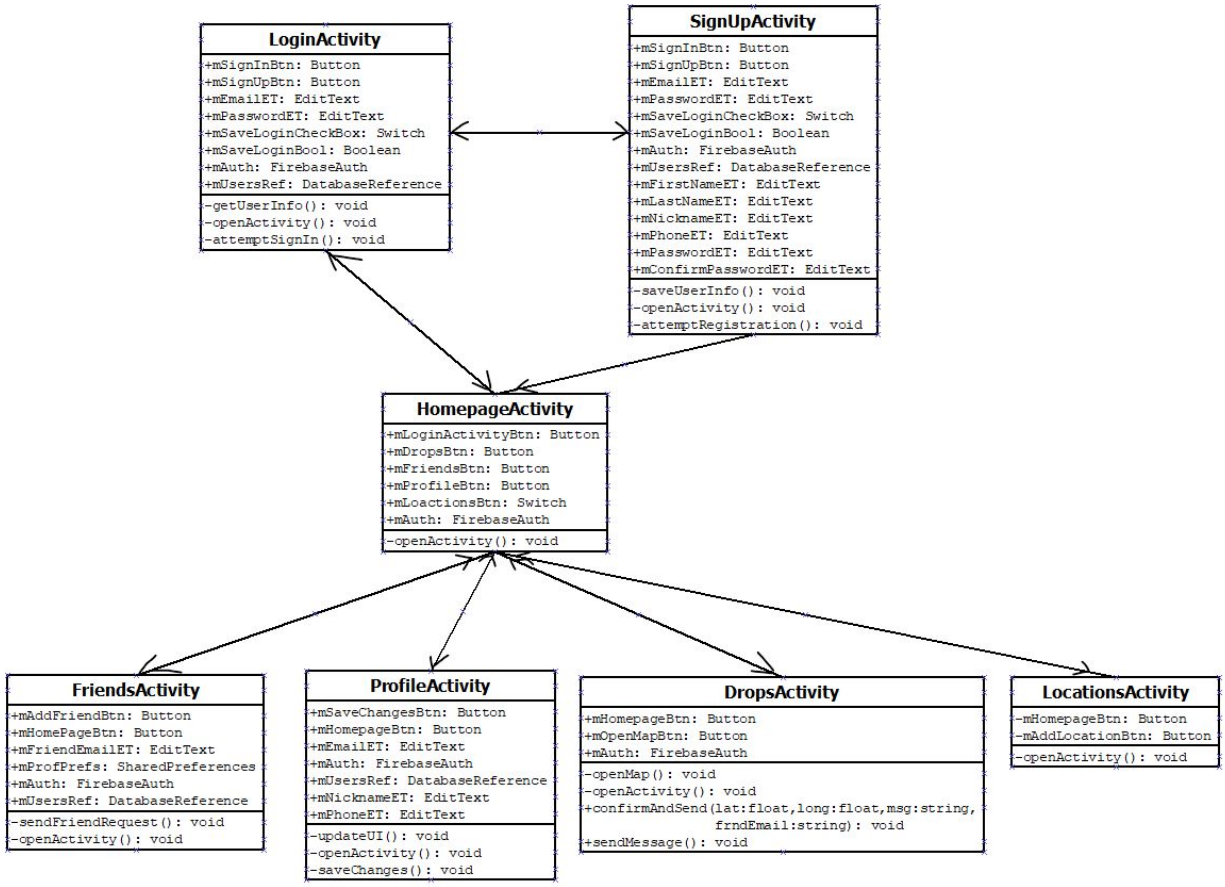


This diagram will be used to show how the design of the app works on both the map and webbing of the High-Level Design.

3. **Low-Level Design**:



This low-level design shows the outline of the classes, its variables, and member functions.



4. **Pseudo-Code Algorithms of Critical Functions in Class Diagram**:

>>Start in Homepage Activity
>>Homepage consists of buttons with following options: Login, Sign-Up, Friends, Profile, Locations, and Drops Activity.
>>If the user presses Login, they'll login into their profile where their contact info is collected and put into Firebase Data Storage.
>>If the user doesn't have an account for the app, they'll go to Sign-In and place their contact info into Firebase Data Storage.
>>The registered user can then look up or add someone on his/her friends list. This algorithm will showcase shared preferences between each other just like the model of other social media platforms.
>>If the user goes to Profile, he/she'll be able to change their profile such as info, avatar, and nickname.
>>The Location Activity allows the user to keep track and add locations from Google Maps.
>>The user can then use these pieces of info to drop messages onto people in certain locations, as its intended design.

```
void attemptSignIn(){
        if(user filled fields incorrectly ){
                show error message;
                return;
        }

        if( email and password match database email and password){
                getUserInfo();
                move to homepage activity;
        }
        else {
                show error message;
        }
}

Void getUserInfo(){
        retrieve user info from database;
        store info in Profile_Prefs;
}




Void attemptRegistration(){
        if(user entered sign up fields correctly and user with same email does not exist){
                create new user class with user entered information;
                send new user class to database to store;
                saveUserData();
                move to homepage activity;
        }
}


Void saveUserInfo(){
        store user entered data in ProfilePrefs;
}

Void sendFriendRequest(){
        if(receiver email belongs to registered user){
                find receiver email associated with user in database;
```

```
                add sender email to friendRequests;
        }
        else{
                showError();
        }
}

Void updateUI(){
        if( currentPassword == mPasswordET){
                saveChanges();
        }
        else{
                “Error, please enter your current password
                before permitting any new information!”
        }
}
```

```
Void sendMessage(){
        If (a friend is selected){
                friendSelected = true;
                friendEmail = email
        }
        Else
                friendEmail = “”
                friendSelected = false;

        If (clicked next && friendsSelected == true){
                openMap(string message, string friendEmail );
        }
        If (clicked back)
                Return to homepage;

Void openMap(){
        Opens google map with scroll feature
```

```
        if( clicked on map == true){
                confirmAndSend(float clickLatitude, float clickLongitude, string message,
                string friendEmail)
        }
        if(clicked back button)
                Send user back to edit message;
}

Void confirmAndSend(float lat, float long, string msg, string frndEmail){
        Sends information to the database to be sent to the friend;
}
```