# CS 458
# Software Validation and Verification

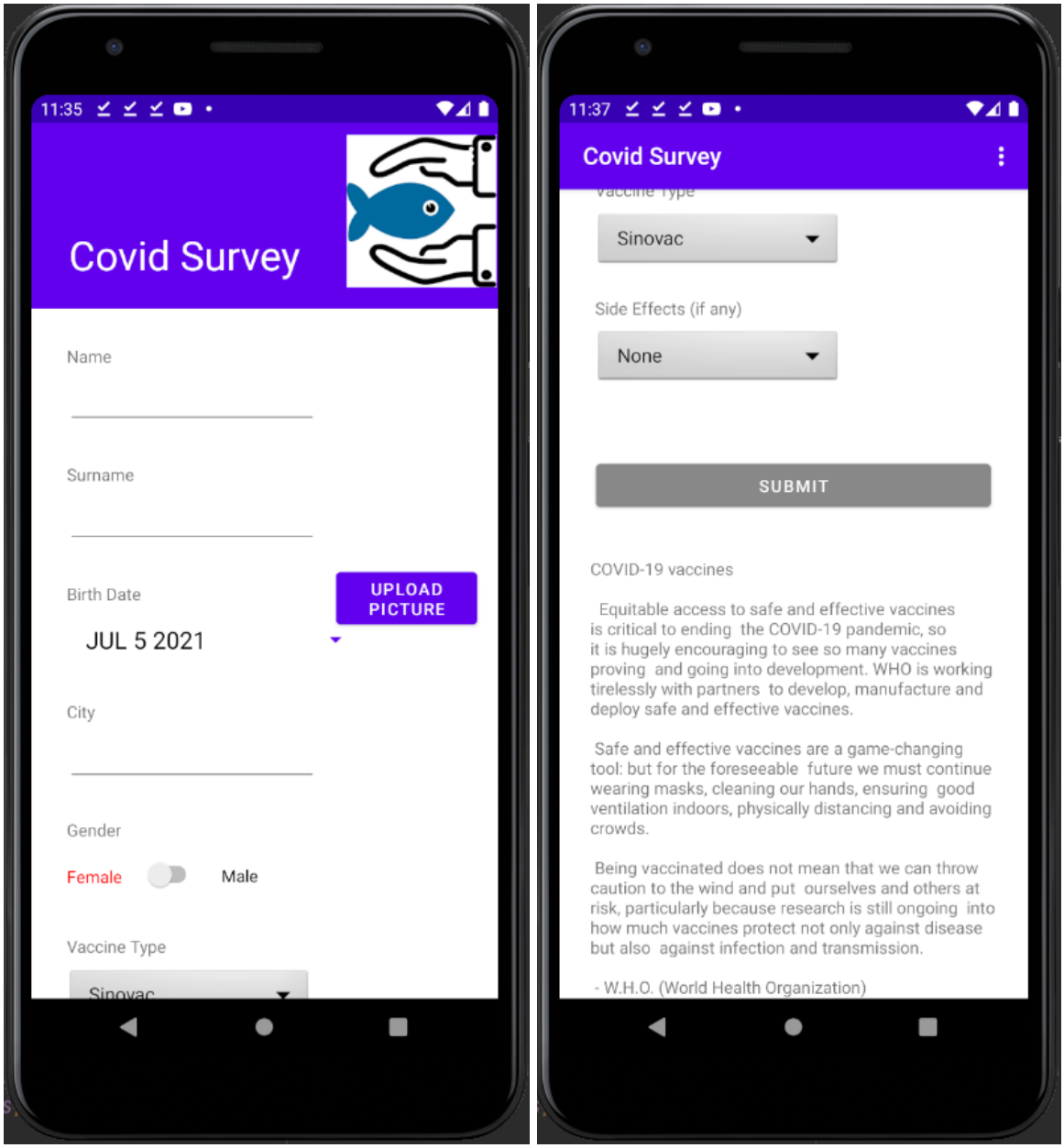

## Introduction to Test Automation (MOBILE)

Project #2


| | |
|---|---|
| Okan Şen | 21202377 |
| Hassam Abdullah | 21701610 |
| Mustafa Tuna Acar | 21703639 |

# 1 Code Details/Information

## 1.1 Screenshot of Mobile Application




Figures 1, 2: Survey Page of the app

This page includes the necessary information that we want the user to provide us. Additionally, there is an informative text about Covid-19 vaccines quoted from the World Health Organization's official website. If any text fields are empty( Name, surname, or city), and if they have not uploaded a photo of themselves, the submit button below will be grayed out and will

not be clickable. Once they are filled, it becomes available and green. However, the user will be prompted to edit their entries if those fields include, white spaces, special characters, or integers. We only accept letters.

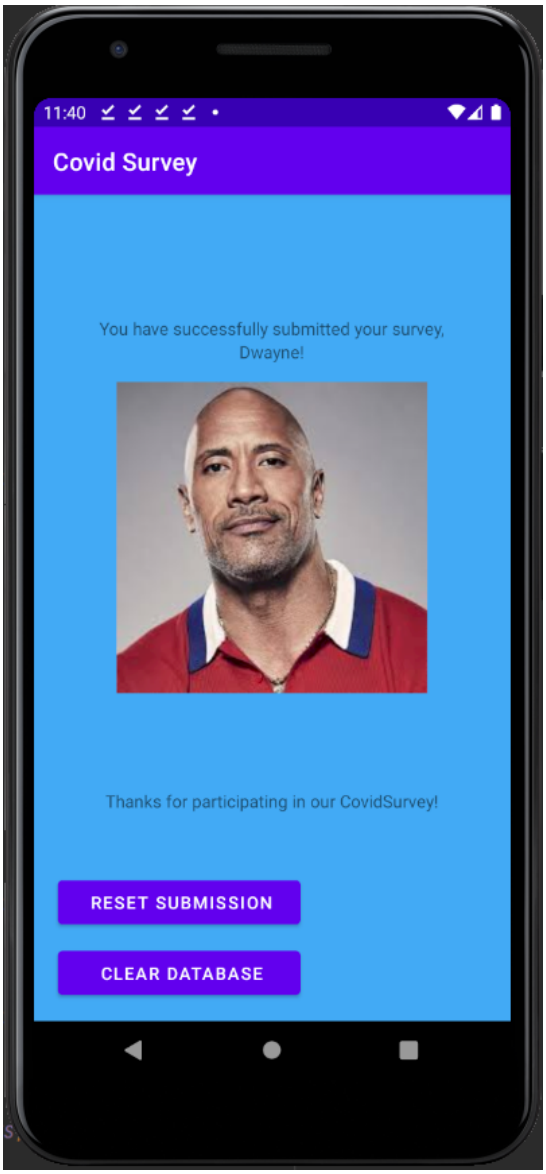



Figure 3: Page View after a successful survey submission

Once a user successfully submits their survey, their information will be stored and they will be directed to this page. Here they will be notified that they have submitted their survey with a personalized message called by their names, and an image showing themselves.

There are two buttons at the bottom of this page, “reset submission” and “clear database”. These are included only for testing purposes, and would not be there if this app was to be released. “Reset submission”, resets the info of this user who has submitted a survey on this

phone and lets them submit another survey. The previous entries are kept. The purpose of this button is to populate the database with a variety of entries when testing quickly. It speeds up both testing and development times. “Clear database” on the other hand, keeps that information of submission while deleting every entry on this particular phone. In order to submit a new survey, the user/tester would have to press the “Reset” button again. Normally, the database would only keep one entry per variable per phone.

## 1.2 Excerpts of Mobile Application Code

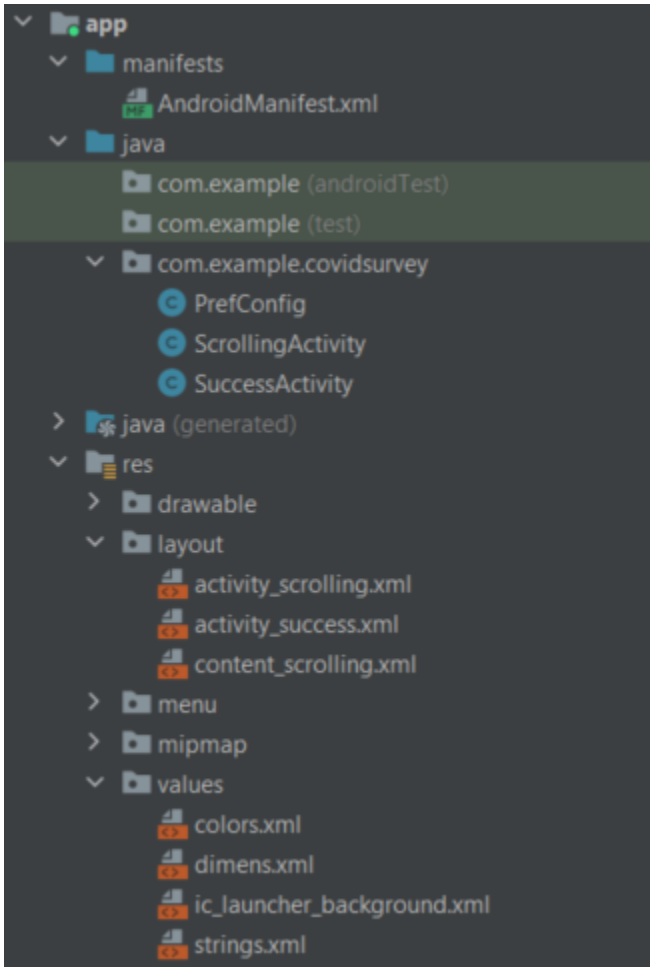
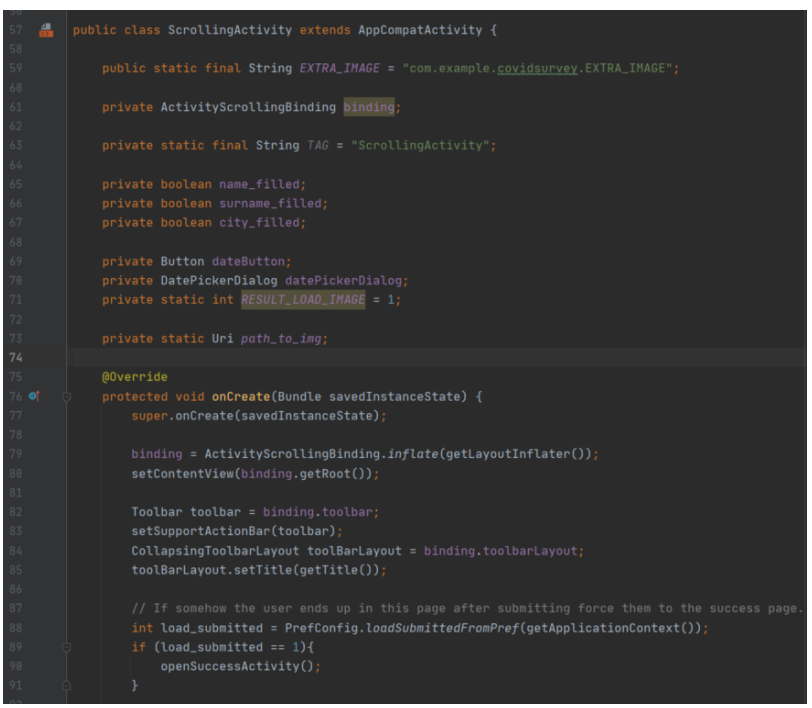
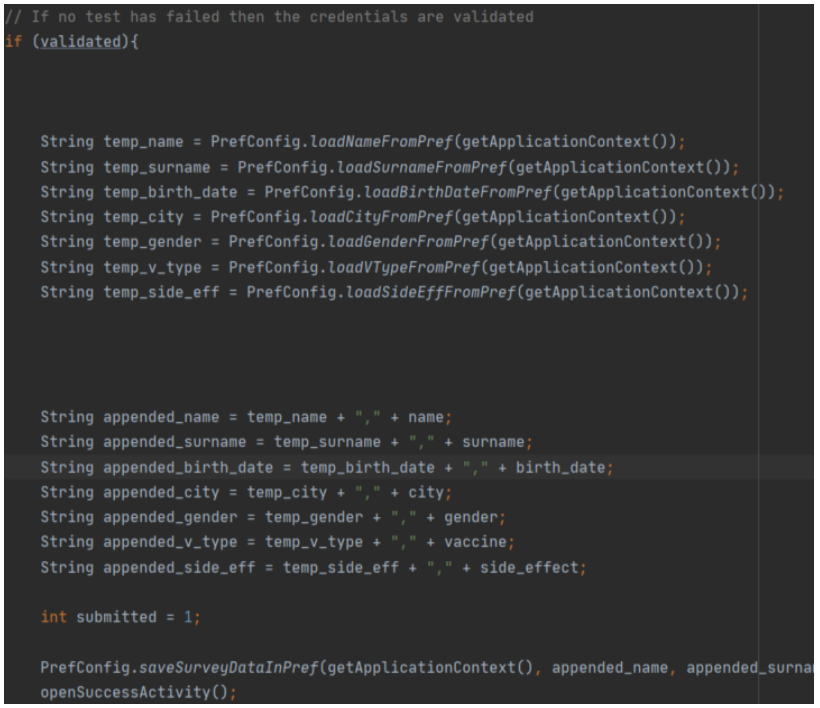



Figure 4: Project View

There are several important files for the app to work as expected. The majority of the work is handled in main classes ScrollingActivity, SuccessActivity, PrefConfig. The others are supporting files or page layout files.



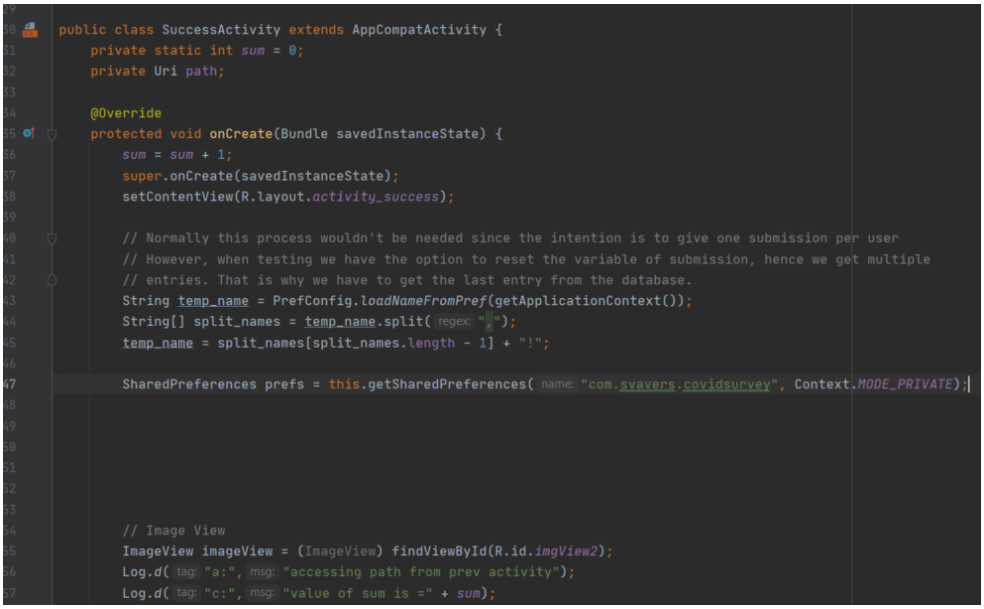
```
public class ScrollingActivity extends AppCompatActivity {

    public static final String EXTRA_IMAGE = "com.example.covidsurvey.EXTRA_IMAGE";

    private ActivityScrollingBinding binding;

    private static final String TAG = "ScrollingActivity";

    private boolean name_filled;
    private boolean surname_filled;
    private boolean city_filled;

    private Button dateButton;
    private DatePickerDialog datePickerDialog;
    private static int RESULT_LOAD_IMAGE = 1;

    private static Uri path_to_img;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityScrollingBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        Toolbar toolbar = binding.toolbar;
        setSupportActionBar(toolbar);
        CollapsingToolbarLayout toolBarLayout = binding.toolbarLayout;
        toolBarLayout.setTitle(getTitle());

        // If somehow the user ends up in this page after submitting force them to the success page.
        int load_submitted = PrefConfig.loadSubmittedFromPref(getApplicationContext());
        if (load_submitted == 1){
            openSuccessActivity();
        }
```



```
// If no test has failed then the credentials are validated
if (validated){



    String temp_name = PrefConfig.loadNameFromPref(getApplicationContext());
    String temp_surname = PrefConfig.loadSurnameFromPref(getApplicationContext());
    String temp_birth_date = PrefConfig.loadBirthDateFromPref(getApplicationContext());
    String temp_city = PrefConfig.loadCityFromPref(getApplicationContext());
    String temp_gender = PrefConfig.loadGenderFromPref(getApplicationContext());
    String temp_v_type = PrefConfig.loadVTypeFromPref(getApplicationContext());
    String temp_side_eff = PrefConfig.loadSideEffFromPref(getApplicationContext());




    String appended_name = temp_name + "," + name;
    String appended_surname = temp_surname + "," + surname;
    String appended_birth_date = temp_birth_date + "," + birth_date;
    String appended_city = temp_city + "," + city;
    String appended_gender = temp_gender + "," + gender;
    String appended_v_type = temp_v_type + "," + vaccine;
    String appended_side_eff = temp_side_eff + "," + side_effect;

    int submitted = 1;

    PrefConfig.saveSurveyDataInPref(getApplicationContext(), appended_name, appended_surna
    openSuccessActivity();
```

Figures 5, 6: Survey Page Main Class Excerpts

These excerpts are from the survey page's main class. The code checks for the validity of the inputs and if everything is good to go, the code lets the user press the submit button and all the info is stored in a private file that can only be accessed by the developers. The only way to access this file for non-developer users is to root their Android phones, however, we are unable to prevent this from happening since any app's security is compromised once a phone is rooted. That is within Android Software's scope.



```
public class SuccessActivity extends AppCompatActivity {
    private static int sum = 0;
    private Uri path;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        sum = sum + 1;
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_success);

        // Normally this process wouldn't be needed since the intention is to give one submission per user
        // However, when testing we have the option to reset the variable of submission, hence we get multiple
        // entries. That is why we have to get the last entry from the database.
        String temp_name = PrefConfig.loadNameFromPref(getApplicationContext());
        String[] split_names = temp_name.split( regex: ",");
        temp_name = split_names[split_names.length - 1] + "!";

        SharedPreferences prefs = this.getSharedPreferences( name: "com.svavers.covidsurvey", Context.MODE_PRIVATE);




        // Image View
        ImageView imageView = (ImageView) findViewById(R.id.imgView2);
        Log.d( tag: "a:", msg: "accessing path from prev activity");
        Log.d( tag: "c:", msg: "value of sum is =" + sum);
```

Figure 7: After submitting a page view main class (SuccessActivity)

This class is to let the user know they have successfully submitted their survey and are prevented from submitting a new entry to the app, and a few fields are loaded from the stored private file to show personalized messages to the user.

```
public class PrefConfig {

    private static final String SURVEY_PREFS = "com.svavers.covidsurvey";
    private static final String PREF_NAME = "pref_name";
    private static final String PREF_SURNAME = "pref_surname";
    private static final String PREF_BIRTH_DATE = "pref_birth_date";
    private static final String PREF_CITY = "pref_city";
    private static final String PREF_GENDER = "pref_gender";
    private static final String PREF_V_TYPE = "pref_v_type";
    private static final String PREF_SIDE_EFF = "pref_side_eff";
    private static final String PREF_SUBMITTED = "pref_submitted";

    public static void saveSurveyDataInPref(Context context, String name, String surname, String birth_da
        SharedPreferences pref = context.getSharedPreferences(SURVEY_PREFS, Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = pref.edit();
        editor.putString(PREF_NAME, name);
        editor.putString(PREF_SURNAME, surname);
        editor.putString(PREF_BIRTH_DATE, birth_date);
        editor.putString(PREF_CITY, city);
        editor.putString(PREF_GENDER, gender);
        editor.putString(PREF_V_TYPE, vaccine_type);
        editor.putString(PREF_SIDE_EFF, side_eff);
        editor.putInt(PREF_SUBMITTED, submitted);
        //editor.putString(PREF_IMAGE, image);
        editor.apply();
    }

    public static String loadNameFromPref(Context context){
        SharedPreferences pref = context.getSharedPreferences(SURVEY_PREFS, Context.MODE_PRIVATE);
        return pref.getString(PREF_NAME, defValue: "");
    }
    public static String loadSurnameFromPref(Context context){
        SharedPreferences pref = context.getSharedPreferences(SURVEY_PREFS, Context.MODE_PRIVATE);
        return pref.getString(PREF_SURNAME, defValue: "");
    }
    public static String loadBirthDateFromPref(Context context){
        SharedPreferences pref = context.getSharedPreferences(SURVEY_PREFS, Context.MODE_PRIVATE);
        return pref.getString(PREF_BIRTH_DATE, defValue: "");
    }
    public static String loadCityFromPref(Context context){
```

Figure 8: PrefConfig Class

This class has the implementation of the data storage methods. It can access and edit the saved private XML file inside the phone. The methods here are only “save” and “load” methods.

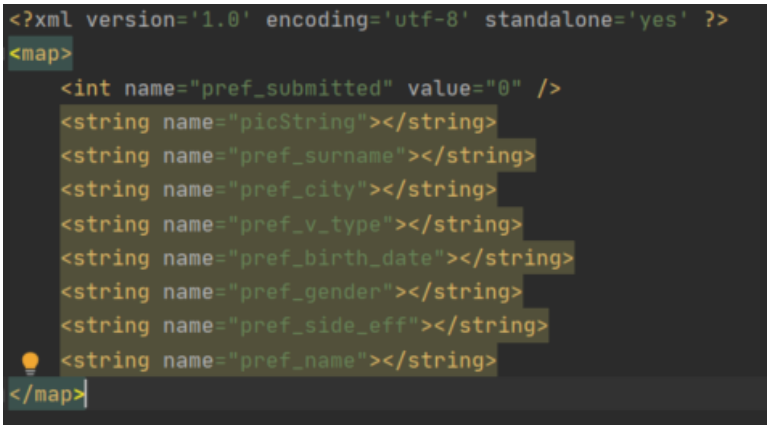
## 1.3 Screenshot of Test Code and Test Cases

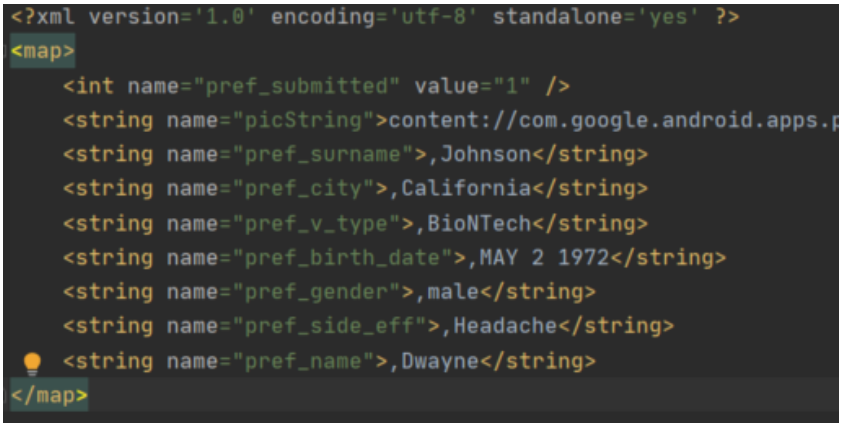
The following test cases have been implemented to ensure the integrity of the mobile application’s functionalities:-

1) Invalid Credentials
2) Responsiveness of Submit Button
3) Installation of APK file / Running in the background check
4) Rotation / Validation of Image upload
5) State of application after successful submission.

### 1.3.1 Invalid Credentials

A valid submission will result on the success page( see figure 3). The XML file acts as a local database in our app and it can be only accessed by the developers. The protected file is located at “device memory -> data -> data -> com.svavers.covidsurvey -> shared_prefs -> com.svavers.covidsurvey.xml”.




Figures 9, 10: Database before any submission and Database after a successful submission

As mentioned in the previous chapter (see chapter 1.1), the database would only keep one entry per variable per phone, however, for quicker testing purposes we implemented a method to submit many more surveys into our database. The entries are separated by a comma, and they can easily be accessed if needed. These entries will be pulled systematically rather than by observing manually (see Figure 11).

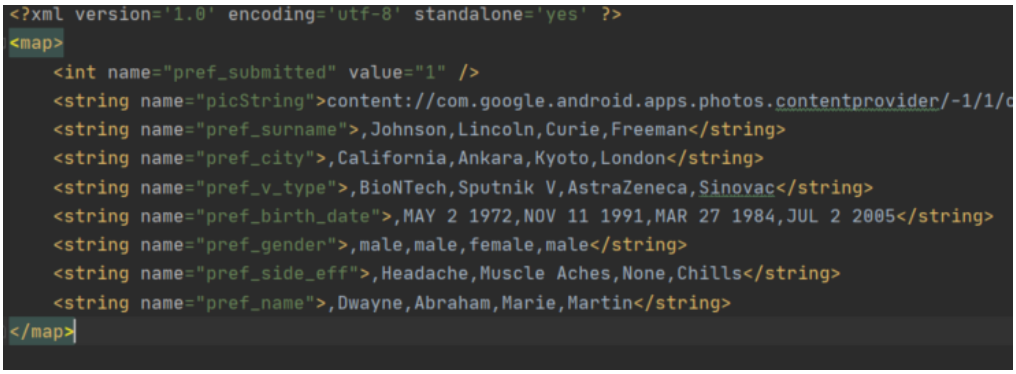



Figure 11: Database after multiple entries

In case of an invalid entry, such as inputting integers or special characters into the name, surname, and city fields, the user will receive a warning (see Figure 12).

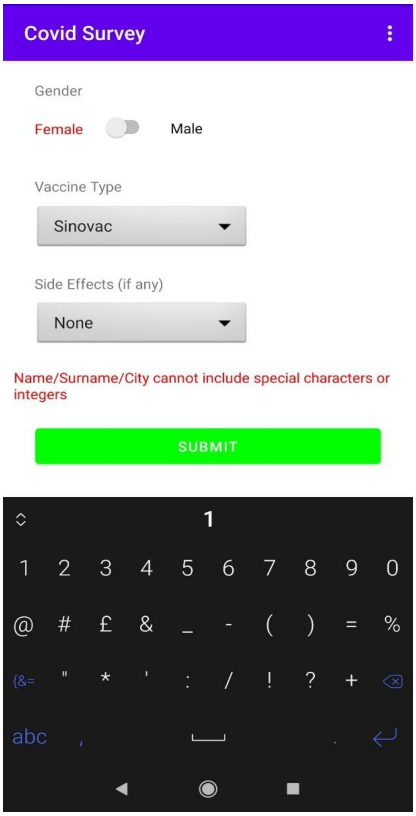



Figure 12

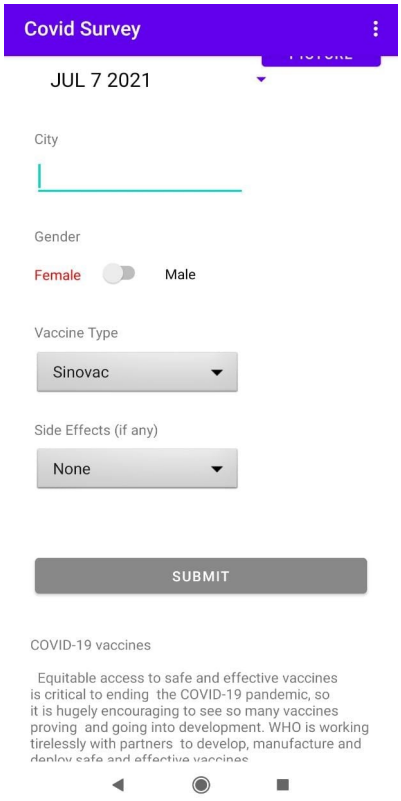



Figure 13

### 1.3.2 Responsiveness of Submit Button

The submit button will not be clickable until the name, surname, and city fields are filled by the User and will be greyed out as shown in Figure 13.

### 1.3.3 Installation of APK / Running in the background check

This test case deals with ensuring that the APK file is properly installed on the android system. In our test case, a physical android device “Xiaomi Redmi 9” was utilized for this purpose. Furthermore, Google Pixel was also used as a virtual emulator in Android Studio.

In addition, it was checked whether the application worked simultaneously with other applications running in the background and the user was able to multitask while using it. For this purpose, an app called Vysor (see Figure 14) was opened and interacted with during the testing process of the CovidSurvey App.

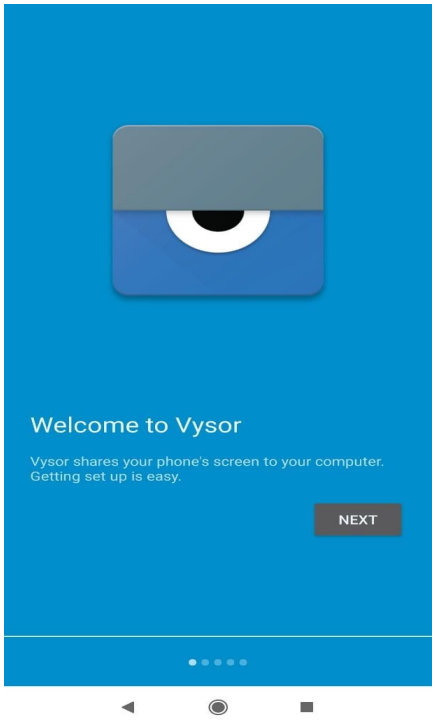



Figure 14

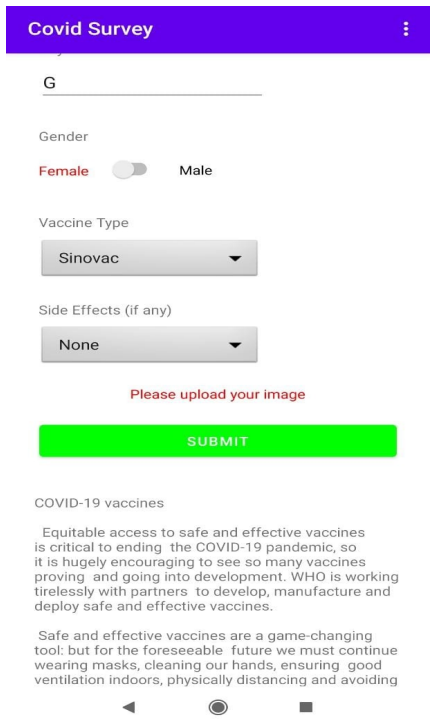



Figure 15


Figure 16

### 1.3.4 Rotation / Validation of image upload

This test case deals with testing if the android application functions correctly and is displayed correctly to the user when the user rotates and uses it horizontally in Landscape Mode (See Figure 17). Furthermore, it also deals with checking that the user has uploaded an image to the image field located on the app screen by clicking the upload button (See Figure 16) otherwise the user will be shown an error, as in Figure 15.

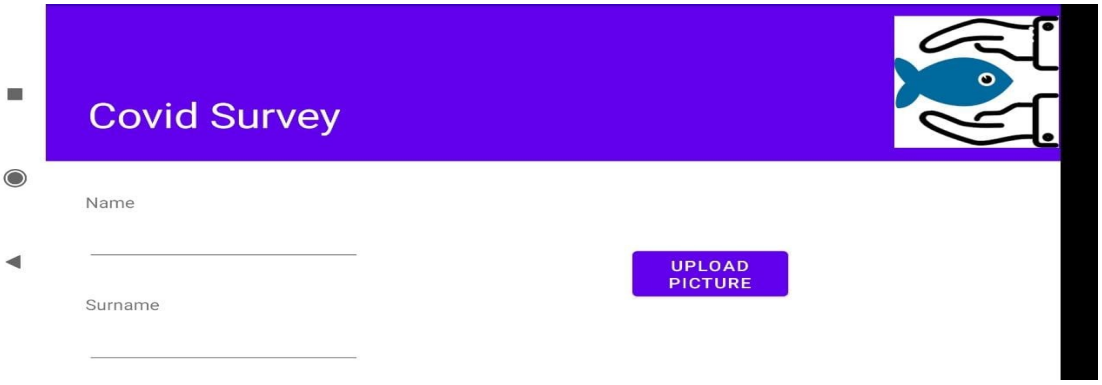



Figure 17

### 1.3.5 Maintaining the state of the application after successful submission

This test case deals with ensuring that the user is not allowed to send the form repeatedly and once the user has successfully submitted the survey then he will need to click on either the reset submission button or the clear database button as shown in Figure 18. Otherwise, whenever the user reopens the android application he will be directed to the screen as shown in Figure 18.

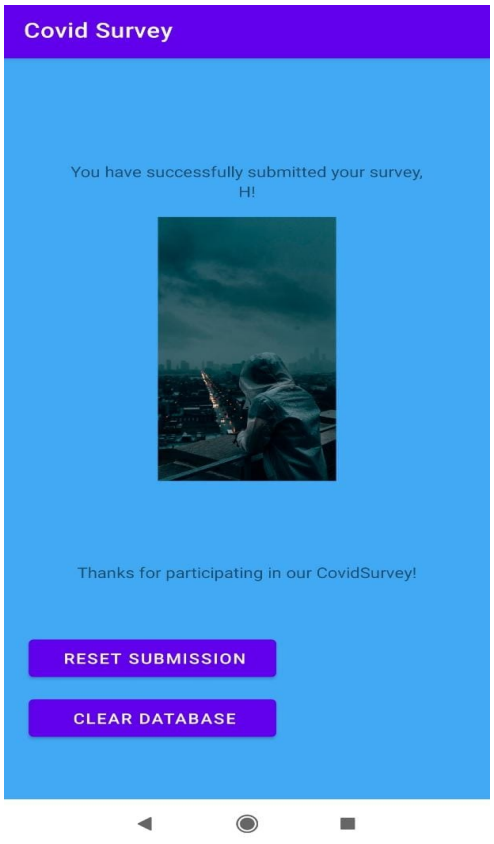



Figure 18

## 1.4 Excerpts of Test Code

Important excerpts of test code can be viewed below that are integral to the functioning and understanding of the automation code using selenium.

```
public class AppiumTest {

    static DesiredCapabilities cap = new DesiredCapabilities();
    static AndroidDriver<MobileElement> driver;

    public static void main(String[] args) {

        try {
            System.out.println("Test Case 3(installing the apk)");
            installCovidSurvey(); //part of test case 3
            System.out.println("Running Test Case 2(Empty field disables submit button");
            testCase2();
            System.out.println("Running Test Case 1 / Test Case 3(Switching apps)");
            testCase1();
            System.out.println("Running Test Case 4 (User will stay on the success page until he retries");
            testCase4();
            testCase5Rotation();
            Thread.sleep( millis: 2000);
            driver.rotate(ScreenOrientation.PORTRAIT);

        } catch (Exception exp) {
            System.out.println(exp.getCause());
            System.out.println(exp.getMessage());
            exp.printStackTrace();

        }


    }
```

Figure 19

Figure 19, shows the main method of the java test file where all the methods for each test case are run.

```
//THIS DEALS WITH TEST CASE 3 and installs the app
public static void installCovidSurvey() throws Exception {



    cap.setCapability( capabilityName: "deviceName",  value: "fc72616"); //android device id being used
    cap.setCapability( capabilityName: "platformName",  value: "android"); //implemented on android
    cap.setCapability(MobileCapabilityType.AUTOMATION_NAME,  value: "UiAutomator2");
    cap.setCapability( capabilityName: "app",  value: "C:\\Users\\Hassam\\Desktop\\Appium\\Covid_Survey.apk"); //this directory will need to be modfied
    cap.setCapability( capabilityName: "appPackage",  value: "com.svavers.covidsurvey");
    cap.setCapability( capabilityName: "appActivity",  value: "com.example.covidsurvey.ScrollingActivity");
    cap.setCapability( capabilityName: "skipDeviceInitialization",  value: true);
    cap.setCapability( capabilityName: "skipServerInstallation",  value: true);
    cap.setCapability( capabilityName: "noReset",  value: true);
    cap.setCapability( capabilityName: "dontStopAppOnReset",  value: true);
    URL url = new URL( spec: "http://127.0.0.1:4723/wd/hub"); //appium server being used
    driver = new AndroidDriver<MobileElement>(url, cap);
    System.out.println("Covid Survey Application has started");

}
```

Figure 20

Figure 20, shows the method that installs the Covid Survey app onto the mobile device. As shown, certain capabilities such as “deviceName” have to be set for each individual mobile phone, the directory of the apk file, the application package name, and the application main activity name, as well, amongst other things

```
public static void switchApps()
    {
        //App1 capabilities
        String covidAppPackageName = "com.svavers.covidsurvey";
        String covidActivityName = "com.example.covidsurvey.ScrollingActivity";

        // App2 capabilities
        String browserAppPackageName="com.koushikdutta.vysor";
        String browserAppActivityName=".StartActivity";

        //launch settings App
        driver.startActivity(new Activity(browserAppPackageName, browserAppActivityName));

        driver.findElement(By.id("com.koushikdutta.vysor:id/next")).click();



        driver.launchApp();


}
```

Figure 21

Figure 21, shows the method that is used to instruct Appium to switch android applications and during the test run an app call to an application called Vysor is invoked

```
//test case 2 where empty fields would disable the submit button completely
public static void testCase2(){
    //the fields in the app will be empty at this point and the submit button should not be clickable

    scrollByID( id: "com.svavers.covidsurvey:id/button", index: 0);
    MobileElement el5 = driver.findElementById("com.svavers.covidsurvey:id/button");

    el5.click();

    if(el5.isEnabled()){
        System.out.println("The button is disabled, empty test case is working properly");
    }
}


//This is testcase 1 dealing with invalid credentials and has partially testcase 3 by implementing app switching
public static void testCase1(){
    driver.manage().timeouts().implicitlyWait( l: 500, TimeUnit.SECONDS);



    //now fields will be not empty and will be filled

    //Now go back upto name field
    scrollByID( id: "com.svavers.covidsurvey:id/editTextTextPersonName", index: 0);
    MobileElement el1 = driver.findElementById("com.svavers.covidsurvey:id/editTextTextPersonName");

    //Try first name has integer
    String name = "Jonathan*-123";
    String surname = "Davis//123";
    String city = "Ankara0Hello";

    System.out.println("Now entering the following fields, name, surname, city: " + name + ", " + surname + ", " + city);

    //FIRST EXAMPLE ALL FIELDS ARE INVALID
```

Figure 22

These two methods shown in Figure 22, are used to facilitate in implementing the test cases specified in 1.3.1, 1.3.2, and partially 1.3.3 (Switching of apps).

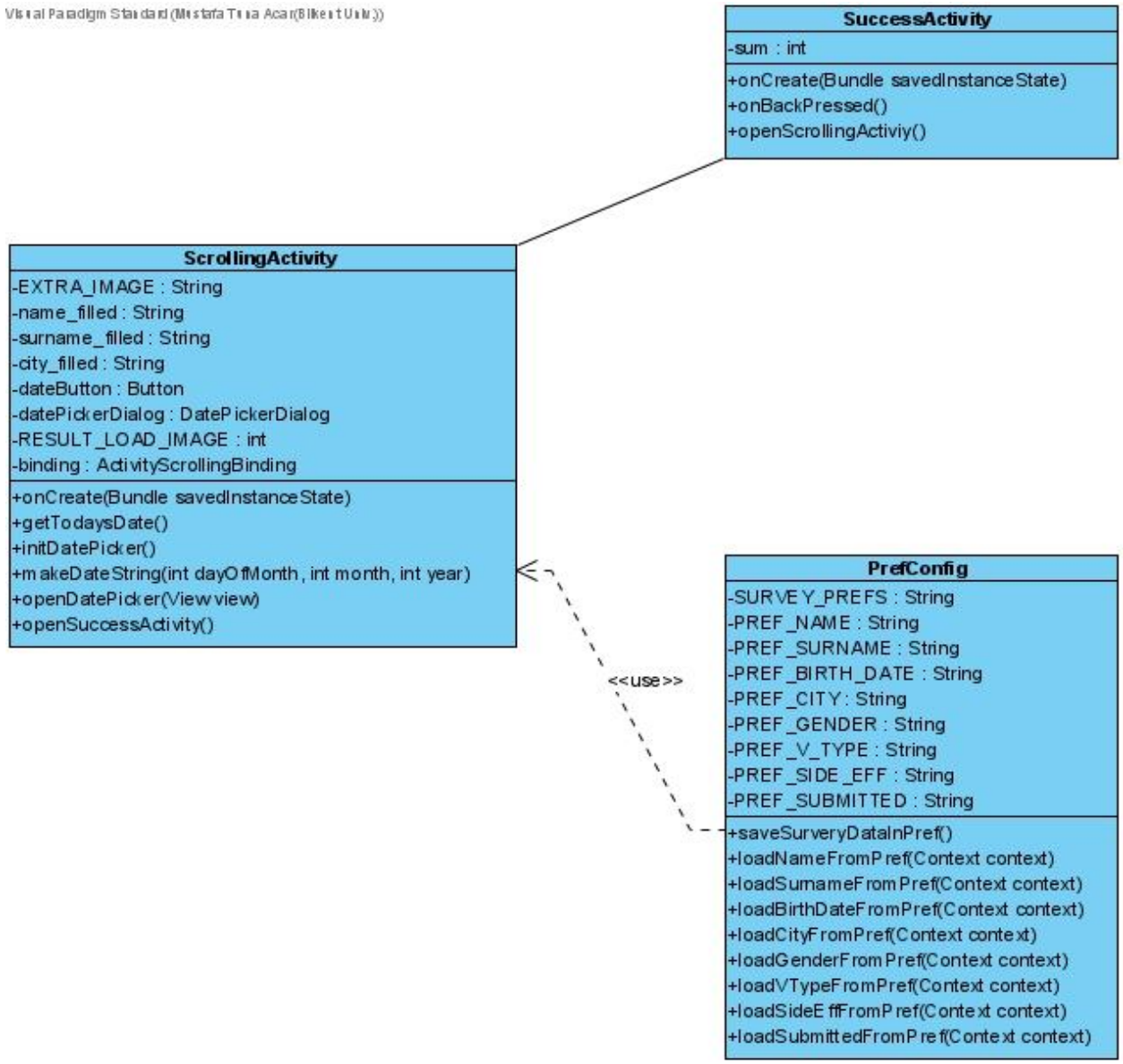
```
//This method scrolls the app till element is found on the screen
public static void scrollByID(String Id, int index) {

    try {

        driver.findElement(MobileBy.AndroidUIAutomator( uiautomatorText: "new UiScrollable(new UiSelector().scrollable(true).instance(0)).scrollIntoView(new UiSelector().resourceId(\""+Id+"\").instance("+index+"));"));

    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static void verticalSwipeByPercentages(double startPercentage, double endPercentage, double anchorPercentage) {
    Dimension size = driver.manage().window().getSize();
    int anchor = (int) (size.width * anchorPercentage);
    int startPoint = (int) (size.height * startPercentage);
    int endPoint = (int) (size.height * endPercentage);
    new TouchAction(driver)
            .press(point(anchor, startPoint))
            .waitAction(waitOptions(ofMillis(1000)))
            .moveTo(point(anchor, endPoint))
            .release().perform();
}
```

Figure 23

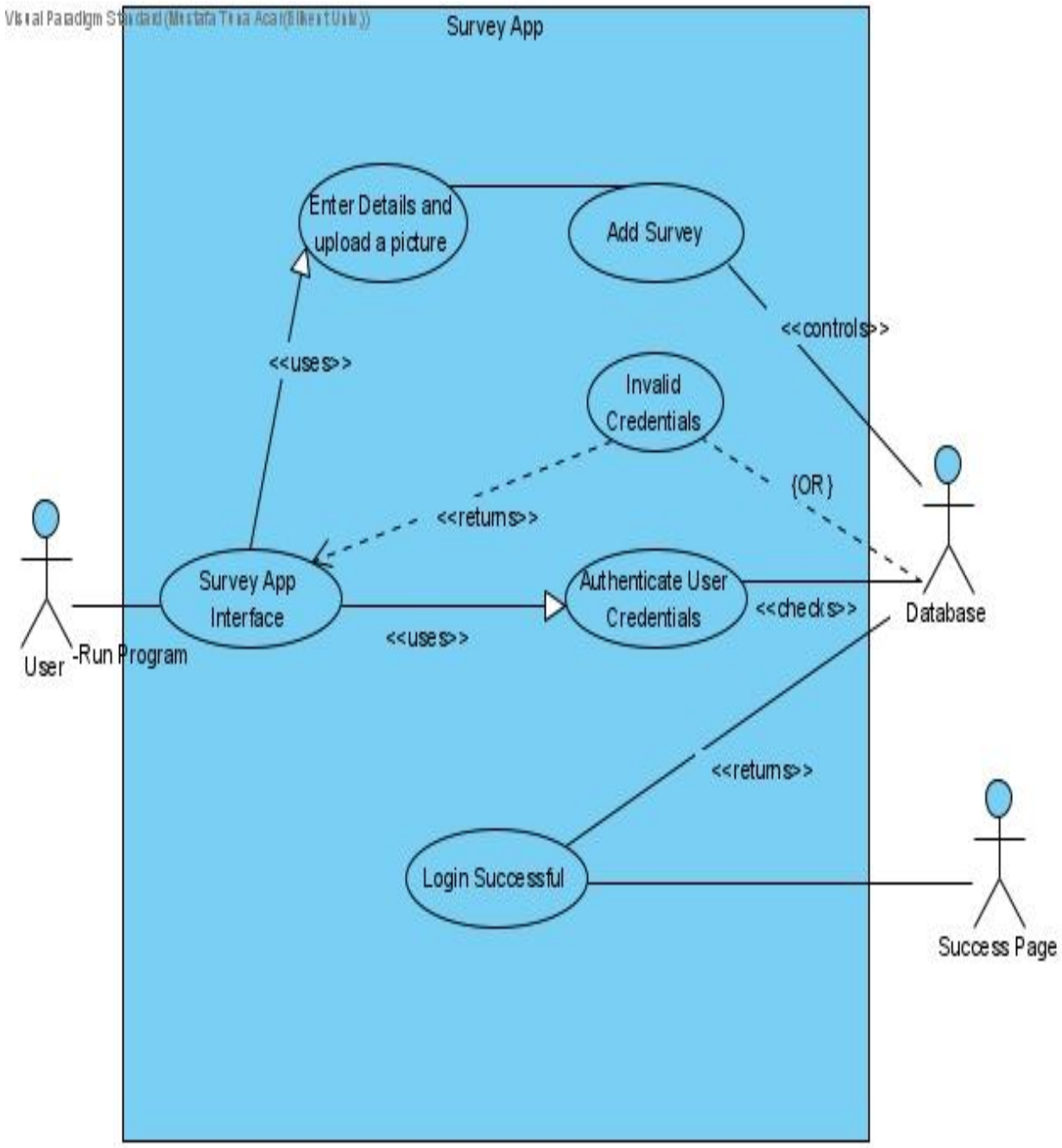
The methods shown in Figure 23, are crucial to the running of the application and are used to move to a certain mobile element until it is visible on the screen and also to simulate the swiping vertically upwards on an android screen.

# 2 UML Diagrams

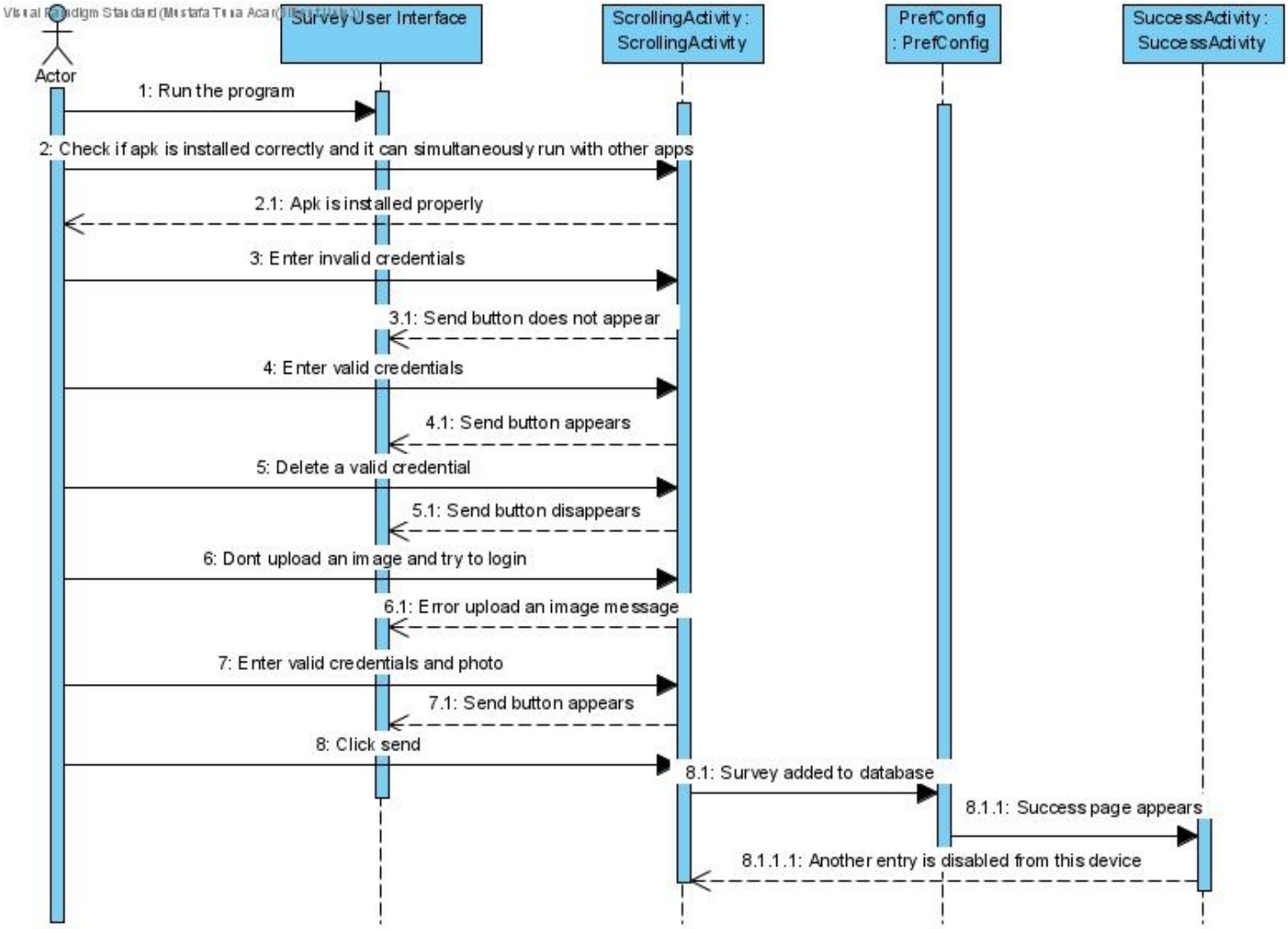
## 2.1 Class Diagram



## 2.2 Use Case Diagram



## 2.3 Sequence Diagram



# 3 Analysis of Appium’s Capabilities

Appium is a testing automation tool that is available as open-source and can be used in mobile web, iOS mobile, Android mobile, and Windows desktop platforms. Native, Mobile web, and Hybrid apps can be developed and tested using Appium.

**List of Appium's Capabilities:**

1 - It supports popular programming languages such as Java, PHP, Python, Ruby, and Node.

2 - It works with Selenium WebDriver JSON wire protocol which enables it to work using native apps.

3 - It is compatible with all testing frameworks.

4 - It does not need to be also installed on mobile devices in order to be working with the tool.

5 - It works with standard automation API'S on all platforms so there is no additional need to recompile or modify the app while it is under test.

6 - It can be integrated with Sauce labs which makes it be cost-effective since it gets rid of the need for setting up multiple platforms in order to test the apps on multiple devices and platforms.

7 - It can be easily set up on a different platform.

8 - Both native, hybrid, and mobile web applications can be automated using Appium.

9 - Since it is open-source, it is easy and free to acquire and use.

10 - The test scripts can be written using any of the programming languages which are compatible with the web driver.

# 4 Automation Experience

## 4.1 Automation Experience

During the past two weeks, we have had the opportunity to develop a working understanding of how to make an android application and automate it using Appium. Java was used as the programming language since we were well-acquainted with it from our past experiences. The automation script was written using Java as well to communicate with the

Appium server. However, we were not satisfied with the login page we had, so we implemented a scrollable login page with a pseudo-database that saved the user's data upon completion of the survey in the packages data folder. It was a challenging experience to work on this project on such short notice and accomplish our goal of creating and automating our test cases.

Another challenge was to implement all of the test cases in Java using Selenium. More complicated cases would require manual testing, although we wanted to make it fully automated, without any tester having to interfere with the process, and it turned out well. We had to learn to plan ahead and consider many perspectives in order to consider a wide variety of test cases. There were critical issues relating to the accessing of mobile elements relating to their availability on the current screen, so it had to be ensured that elements were only accessed when they were visible on screen.

## 4.2 Comparison between Appium and Selenium

Selenium is ideal for testing websites or web applications and achieves this by controlling the browser directly through its webdriver. On the other hand, Appium is developed as an HTTP server using NodeJS for mobile applications, and NodeJS and the server should be run on the system using Appium as a testing application.

Selenium achieves test automation either by using its Web-Driver or by its self-titled Selenium IDE, while Appium does so by using Selenium's Web-Driver to control iOS or Android sessions.

Test automations scripts can be written in most programming languages for Selenium and Appium but Appium does this by importing the Selenium Client library which has to be included in the project files.

Although Selenium can be used to automate mobile testing as well and has similar methods to Appium such as sendKeys() or findByElementId(), Appium has specialized tools such as its Element Inspector that make it easy and reliable to inspect mobile elements and write code for test automation.

As of the time of writing this, Appium does not support Linux therefore testers using Linux as their operating system are not able to utilize it and therefore valid options such as Windows must be used.