Bilkent University

Department of Computer Engineering

Project 2 Report

INTRODUCTION TO TEST AUTOMATION (MOBILE)

VaccForm

Ahmet Işık	21702226
Furkan Ahi	21501903
Bora Fatih Kazancı	21801753
Yusuf Mert Yıldırım	21802848

Instructor: Dr. Haluk Altunel

Project 2 Report Mar 24, 2022
This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Software Verification and Validation CS 458 course.

Table of Contents

1. Introduction 3

2. Vaccine Pollster App 3

3. UML Diagrams 6

3.1. Use Case Diagram 6

3.2. Activity Diagram 7

3.3. State Diagram 8

4. Capabilities of Appium 8

5. Test Cases 8

5.1. Test Case #1 9

5.2. Test Case #2 9

5.3. Test Case #3 9

5.4. Test Case #4 9

5.5. Test Case #5 9

6. Test Automation 9

6.1. Automating Test Case #1 10

6.2. Automating Test Case #2 11

6.3. Automating Test Case #3 12

6.4. Automating Test Case #4 13

6.5. Automating Test Case #5 16

7. Automation Experience 18

8. Selenium vs. Appium 19

9. Repository Link 19

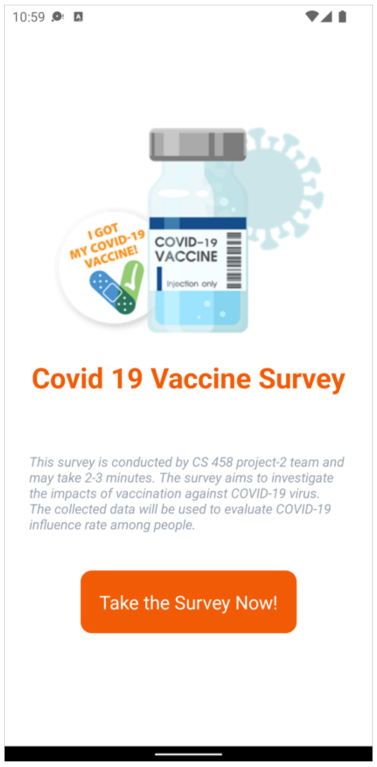
References 20

1. Introduction

This project aims to develop an application reflecting a mobile survey page. The project was developed cross-platform using React Native and Expo technologies. The user is expected to enter personal and vaccination information in the application. After filling in the required fields, it is aimed to store the user data.
In addition, test cases developed through Appium are tested and controlled to test the project from various aspects. In parallel with these, various capabilities of Appium are listed in the report. In the following sections, many test cases are covered in detail, impressions of the mobile automation experience are shared, and Appium vs. Selenium comparisons are made.

2. Vaccine Pollster App

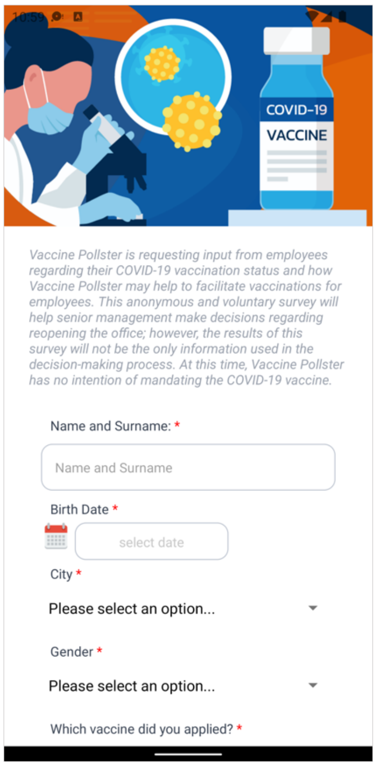
The project developed with React Native can run on both iOS and Android platforms. Input fields such as name, surname, and vaccine type are text inputs. In fields such as date of birth and city, pickers such as date picker and drop-down picker were used. Finally, a picker was used in the gender selection part, and the radio buttons were used for yes/no questions.



Covid 19 Vaccine Survey

This survey is conducted by CS 458 project-2 team and may take 2-3 minutes. The survey aims to investigate the impacts of vaccination against COVID-19 virus. The collected data will be used to evaluate COVID-19 influence rate among people.

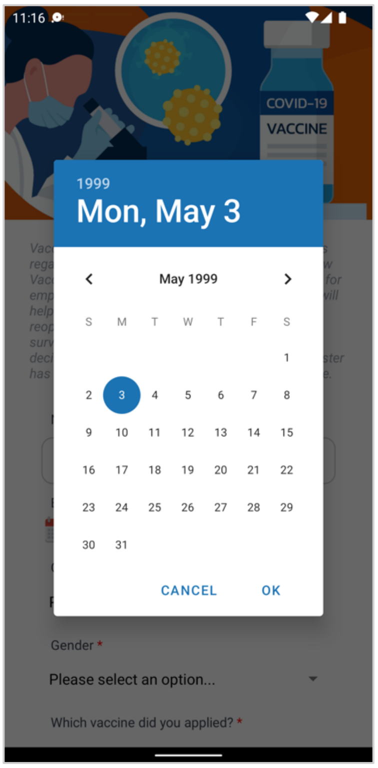
Figure 1: Welcome Screen



Vaccine Pollster is requesting input from employees regarding their COVID-19 vaccination status and how Vaccine Pollster may help to facilitate vaccinations for employees. This anonymous and voluntary survey will help senior management make decisions regarding reopening the office; however, the results of this survey will not be the only information used in the decision-making process. At this time, Vaccine Pollster has no intention of mandating the COVID-19 vaccine.

Name and Surname: *
Name and Surname
Birth Date *
select date
City *
Please select an option...
Gender *
Please select an option...
Which vaccine did you applied? *

Figure 2: Survey Screen



11:16

1999

Mon, May 3

May 1999

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

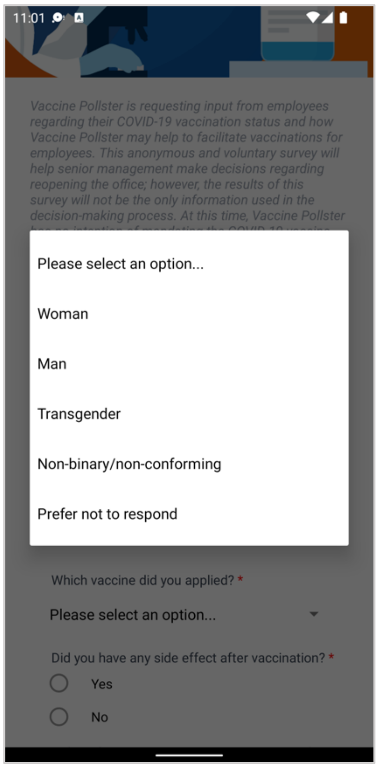
CANCEL OK

Gender *

Please select an option...

Which vaccine did you applied? *

Figure 3: Birthday Picker



11:01

Vaccine Pollster is requesting input from employees regarding their COVID-19 vaccination status and how Vaccine Pollster may help to facilitate vaccinations for employees. This anonymous and voluntary survey will help senior management make decisions regarding reopening the office; however, the results of this survey will not be the only information used in the decision-making process. At this time, Vaccine Pollster

Please select an option...

Woman

Man

Transgender

Non-binary/non-conforming

Prefer not to respond

Which vaccine did you applied? *

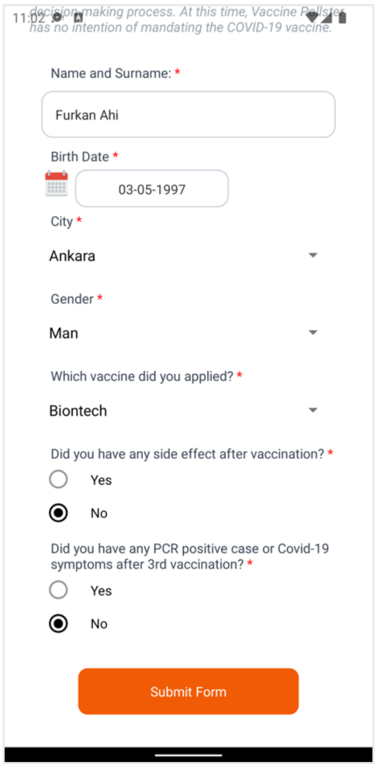
Please select an option...

Did you have any side effect after vaccination? *

Yes

No

Figure 4: Gender Picker



11:02

decision-making process. At this time, Vaccine Pollster has no intention of mandating the COVID-19 vaccine.

Name and Surname: *

Furkan Ahi

Birth Date *

03-05-1997

City *

Ankara

Gender *

Man

Which vaccine did you applied? *

Biontech

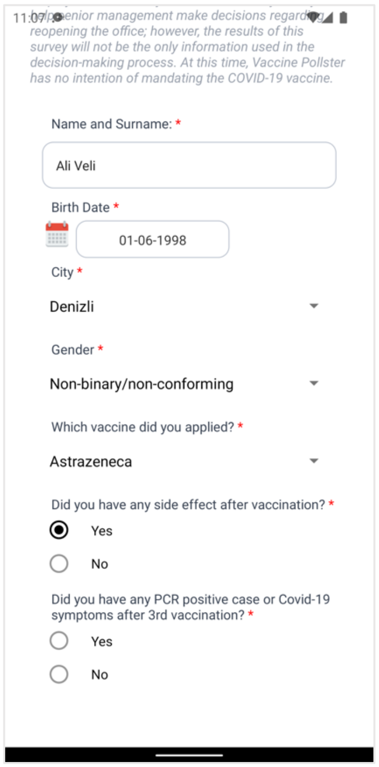
Did you have any side effect after vaccination? *

Yes [✓ No]

Did you have any PCR positive case or Covid-19 symptoms after 3rd vaccination? *

Yes [✓ No]

Figure 5: Completed Survey with the Submit Button



11:07

help senior management make decisions regarding reopening the office; however, the results of this survey will not be the only information used in the decision-making process. At this time, Vaccine Pollster has no intention of mandating the COVID-19 vaccine.

Name and Surname: *

Ali Veli

Birth Date *

01-06-1998

City *

Denizli

Gender *

Non-binary/non-conforming

Which vaccine did you applied? *

Astrazeneca

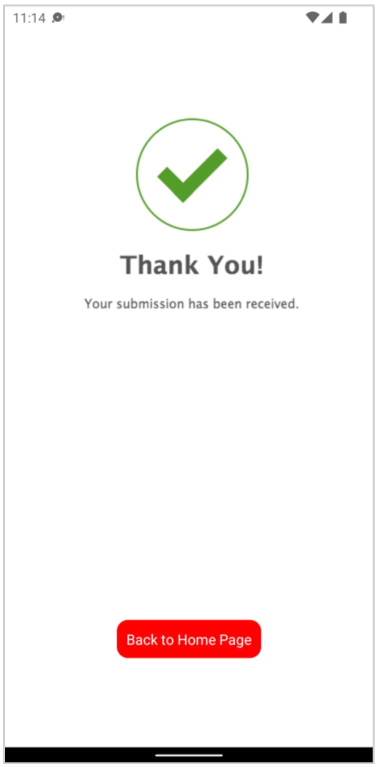
Did you have any side effect after vaccination? *

[✓ Yes] No

Did you have any PCR positive case or Covid-19 symptoms after 3rd vaccination? *

Yes No

Figure 6: Uncompleted Survey without the Button



11:14

Thank You!

Your submission has been received.

Figure 7: Post-Submission Page

3. UML Diagrams

3.1. Use Case Diagram

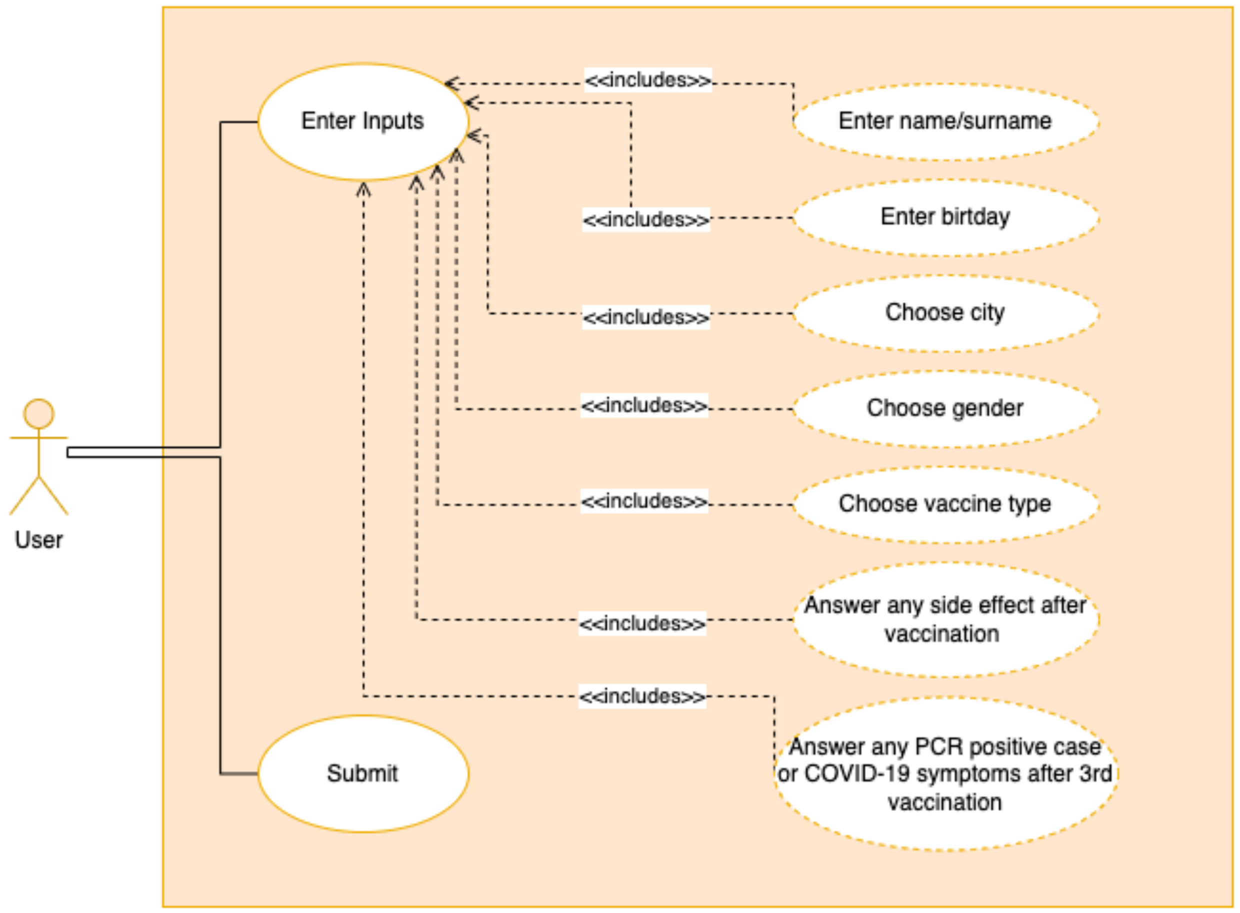


Figure 8: Use-Case Diagram for the Survey Page

Use case:

Use Case	A user takes a survey
Actor(s)	Any User
Entry Condition(s)	The survey page should be loaded
Exit Condition(s)	The user needs to enter valid inputs and click the submit button.
Main Flow of Event(s)	1. The user enters a name and surname. 2. The user enters a birthday. 3. The user chooses a city. 4. The user chooses a gender 5. The user chooses a vaccine-type 6. The user answers about whether s/he has had any side effects.

	7. The user answers whether s/he has had any PCR positive case or COVID-19 symptoms after the 3rd vaccination. 8. The user taps the submit button.
Alternative Flow of Events	• The user enters invalid input. • The user leaves the input boxes blank or unselected.

3.2. Activity Diagram

This diagram models the app behaviors while a user tries to take a survey. It also shows the system decisions and reactions to the actions of the user

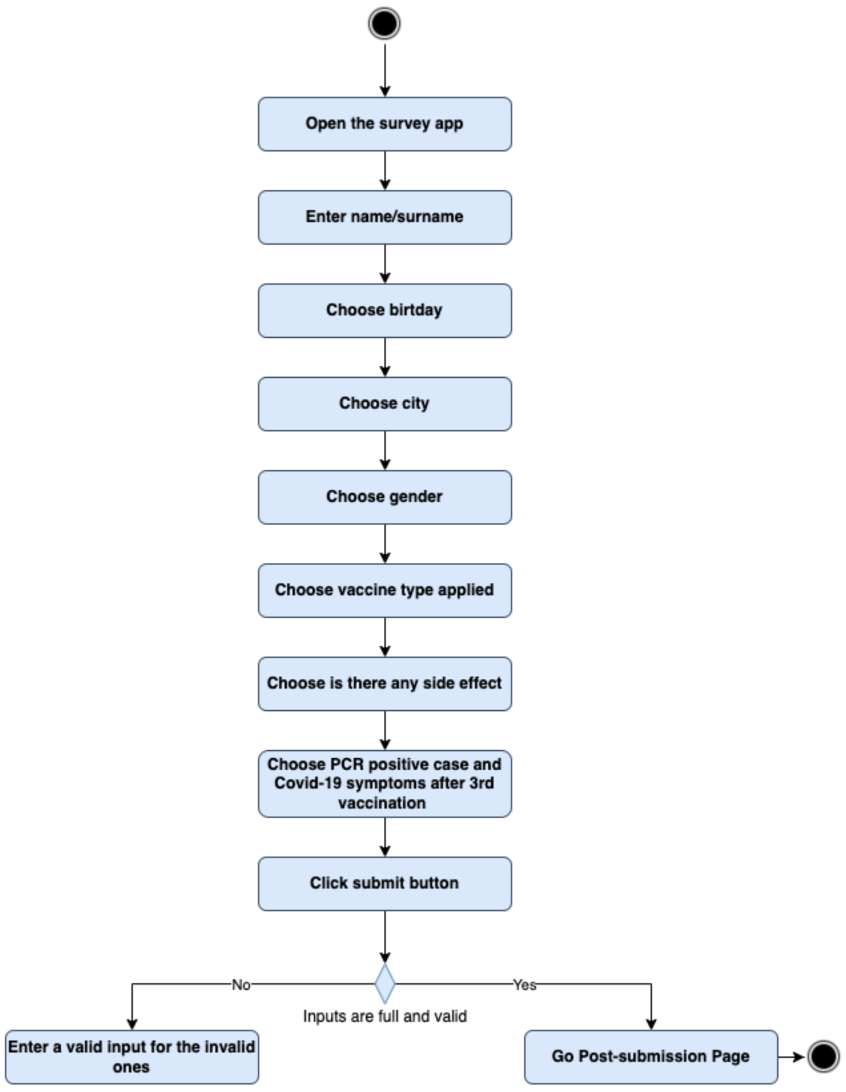

Open the survey app
Enter name/surname
Choose birtday
Choose city
Choose gender
Choose vaccine type applied
Choose is there any side effect
Choose PCR positive case and Covid-19 symptoms after 3rd vaccination
Click submit button
No
Yes
Inputs are full and valid
Enter a valid input for the invalid ones
Go Post-submission Page

Figure 9: Activity Diagram for the Survey App

3.3. State Diagram

This diagram models the app behaviors while the user takes a survey and the following behaviors. It also shows the system decisions and reactions to the user's actions.

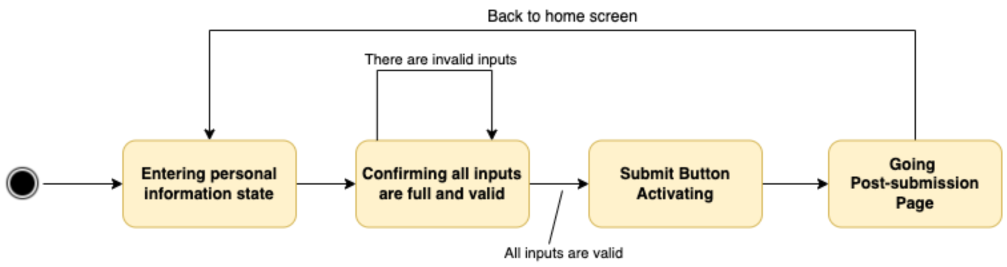

Back to home screen
There are invalid inputs
Entering personal information state
Confirming all inputs are full and valid
Submit Button Activating
Going Post-submission Page
All inputs are valid

Figure 10: State Diagram for the Survey App

4. Capabilities of Appium

1. Appium is an open-source, cross-platform automation tool. That means, Appium can automate tests independently of operating systems, namely iOS, Android, and FirefoxOS platforms.
2. Appium can automate tests on all hybrid, native, and web apps.
3. Appium can be used to test both real devices and emulators
4. Appium does not require application source code or library.
5. Appium provides a solid and active community.
6. In Appium, a small change does not require the re-installation of the application.
7. Appium allows the parallel execution of test scripts.
8. Instead of simulating a button press by running the code, Appium can simulate user input in native apps using the JSON wire protocol.
9. Appium supports various languages like C#, Python, Java, Ruby, PHP, JavaScript with node.js, and many others that have a Selenium client library. [1]
10. The capabilities the developer wants can be written in the WebDriver test or set in the Appium Server GUI. [2]

5. Test Cases

Five test cases were designed for the application. When designing these cases, the efficiency and common app logic were considered. In the tests, various cases were prepared considering the situations that a standard user tests in much more time. In these cases, efficiency in testing is considered.

5.1. Test Case #1

In test case #1, it is checked that the user can only enter alphanumeric without numbers. This test case aims to see whether the user enters something different than the expected input types. In order to check this, an array list is created with varying inputs of the test. When the wrong input type is entered, the name field gives an error message. In automation, while trying test inputs, the visibility of the error message is checked with the expected visibility constraints to check test inputs are failed or passed.

5.2. Test Case #2

In test case #2, it is checked that the user can select a date in the given date boundaries and cannot select a date outside of the date boundaries in the date picker. A survey can only be taken within a specific age interval. Therefore, some dates must not be chosen by the user. Thus, one valid and one invalid date is selected, and it is checked if dates are enabled or not.

5.3. Test Case #3

In test case #3, it is checked that the selected city appears on the city selection box after a user selects it. In mobile apps, when the user clicks outside of the choice box selection list, it exists from the selection list without the intended selection, and the city choice box has a lot of city choices. Therefore, when clicked on the selection element, the element's text should appear on the city choice box. The text of the element from the selection menu and the text of the city choice box are checked if the test is passed or failed.

5.4. Test Case #4

In test case #4, it is checked that the Submit Button is not visible before all the required fields are filled and visible after all the necessary fields are filled. All the necessary fields are filled until the last field and visibility of the Submit Button is checked. Then the last field is filled, and again, the button's visibility is checked. The results are checked if the test is passed or failed.

5.5. Test Case #5

In test case #5, it is checked that form is submitted successfully and goes to the page with the success message. All the required fields are filled successfully the "Submit Button" is clicked. After the click, the form should be submitted and must go to a page that gives a message of the successful submission. The existence of this message is checked if the test is passed or failed.

6. Test Automation

This section explains how the automation code is written in the project. We used Appium for mobile testing. Before the test cases, the application starts. There are 5 test cases, and they have some common feature: each of them clicks the "Take Survey Button." In automation code, to get an element from the screen, *`findElementByAccessibilityId()`* and *`findElementByXPath()`*

methods are used. First of all, React-Native elements cannot take the id property as in the HTML. Therefore, *accessibilityId* property is given to elements. However, some elements don't take *accessibilityId,* such as radio buttons and a date picker. Hence the usage of the *xPath* property became a necessity. With these methods, elements are found in the given methods. Try-catch is used to give meaningful and precise test result messages. When the test automation is completed, various outputs are reflected in the console as output, and the application closes and, if there are more test cases, starts again.

During the automation code, *Appium. sleep()* is used a lot because automation runs on the Android Emulator, and in the emulator, there are loading times for screens and processes. Finding an element during the loading screens or the screen where the targeted element is not present, find element methods gives error. Hence, the usage of the *Appium.sleep()* function is vital in the automation code.

6.1. Automating Test Case #1

For our test case #1, we have the following code snippet that checks if the user enters only alphanumeric characters without numbers.

```java
public class TestSuite1 {
    @Test
    public void testSuite1(){
        System.out.println("Test Suite 1: Started");
        Appium.sleep(200);
        WebElement attendPoll = Appium.getDriver().findElementByAccessibilityId("take-survey");
        attendPoll.click();
        Appium.sleep(1000);

        WebElement nameField = Appium.getDriver().findElementByAccessibilityId("name-field");
        boolean flag;
        String[] names = {"bora kazanci","borak","bora fatih kazanci","bora kazanci35", "bora
kazanci%%%%&&&&"};
        boolean[] values = {false,true,false,true,true};

        System.out.println("    Test Case 1 : Started");
        for (int i = 0; i < names.length; i++){
            nameField.sendKeys(names[i]);
            Appium.sleep(200);
            try{
                WebElement nameFieldError = Appium.getDriver().findElementByAccessibilityId("name-
warning");
                flag = true;
            }catch(Exception e){
                flag = false;
            }

            try{
                Assert.assertEquals(flag,values[i]);
                System.out.println("        Test is passed on input: " + names[i]);
            }catch (AssertionError a){
                System.out.println("    ***** Test is failed on input: " + names[i]);
            }
            nameField.clear();
        }
        System.out.println("    Test Case 1 : Finished");
        Appium.sleep(1000);
        System.out.println("----------------------");
    }
}
```

Figure 11: Script of Test case #1

In the code snippet, the driver finds the Take Survey Button and clicks. Then the driver finds the name field for the test case. There is a flag variable for comparison, a String array for the test inputs, and a Boolean array for the expected appearance of error messages when the test inputs are entered. For all the test inputs, after the test input String is written in the field, the *accesibilityId* of the error message is checked and holds as the flag input. This flag variable is compared with the test input's expected result, and the message is printed according to the comparison. Then, the field is cleared for the following input. This process is repeated for all the test inputs.

6.2. Automating Test Case #2

For our test case #2, we have the following code snippet, which checks that the user can select a date in the given date boundaries and cannot select a date outside of the date boundaries in the date picker.

```java
public class TestSuite2 {
    @Test
    public void testSuite2(){
        System.out.println("    Test Case 2 : Started");
        Appium.sleep(5000);
        WebElement attendPoll = Appium.getDriver().findElementByAccessibilityId("take-survey");
        attendPoll.click();
        Appium.sleep(1000);

        WebElement datePicker =
Appium.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayou
t/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Fram
eLayout/android.widget.FrameLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewG
roup/android.view.ViewGroup/android.view.ViewGroup/android.view.ViewGroup/android.widget.ScrollView/and
roid.view.ViewGroup/android.view.ViewGroup/android.view.ViewGroup");
        datePicker.click();
        Appium.sleep(1000);

        WebElement validDate = Appium.getDriver().findElementByAccessibilityId("01 June 2004");
        validDate.click();
        WebElement invalidDate = Appium.getDriver().findElementByAccessibilityId("09 June 2004");
        invalidDate.click();

        boolean[] expectedRes = {true, false};
        boolean[] results = {validDate.isEnabled(), invalidDate.isEnabled()};

        for (int i = 0; i < 2; i++) {
            try{
                Assert.assertEquals(expectedRes[i],results[i]);
                System.out.println("        Test is passed on input: " + (i + 1));
            }catch (AssertionError a){
                System.out.println("    ***** Test is failed on input: " + (i + 1));
            }
        }
        System.out.println("    Test Case 2 : Finished");
        System.out.println("---------------------");
    }
}
```

Figure 12: Script of Test case #2

In the code snippet, the driver finds the Take Survey Button and clicks. Then the driver finds the date picker via xPath for the test case. When the date picker is opened, one valid date and invalid date are selected according to date boundaries by the *`findElementByAccessibilityId()`* `methods.` Their enable properties are taken as variables and compared with the expected results. After the comparison, it prints if the test is failed or passed.

6.3. Automating Test Case #3

For our test case #3, we have the following code snippet that checks if the selected city appears on the city selection box after a user selects it.

```
public class TestSuite3 {
    @Test
    public void testSuite3(){
        System.out.println("    Test Case 3 : Started");
        Appium.sleep(4000);
        WebElement attendPoll = Appium.getDriver().findElementByAccessibilityId("take-survey");
        attendPoll.click();
        Appium.sleep(1000);

        WebElement cityBox = Appium.getDriver().findElementByAccessibilityId("city-field");
        cityBox.click();
        Appium.sleep(500);
        WebElement cityBoxElement = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget." +

"FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayo
utCompat/" +

"android.widget.FrameLayout/android.widget.ListView/android.widget.CheckedTextView[4]");
        String cityName = cityBoxElement.getText();
        cityBoxElement.click();
        Appium.sleep(500);


        WebElement cityBoxName =
Appium.getDriver().findElementByXPath("//android.widget.Spinner[@content-desc=\"city-
field\"]/android.widget.TextView");
```

```
        try{
            Assert.assertEquals(cityBoxName.getText(), cityName);
            System.out.println("        Test is passed");
        }catch (AssertionError a){
            System.out.println("    ***** Test is failed");
        }

        System.out.println("    Test Case 3 : Finished");
        System.out.println("---------------------");

    }
}
```

Figure 13: Script of Test case #3

In the code snippet, the driver finds the Take Survey Button and clicks. Then the driver finds the city field via Accessibility Id for the test case. When the city field is opened, the 4th element on the selection area is selected via xPath and gets the element's text as a variable, and then it is clicked. After the selection, the driver finds the city field's name box in order to get the element's text property and compares the selected text with text in the name box. After the comparison, it prints if the test is failed or passed.

6.4. Automating Test Case #4

For our test case #4, we have the following code snippet that checks if the Submit Button is not visible before all the required fields are filled and visible after all the necessary fields are filled.

```
public class TestSuite4 {
    @Test
    public void testSuite4(){
        System.out.println("    Test Case 4: Started");
        Appium.sleep(6000);
        WebElement attendPoll = Appium.getDriver().findElementByAccessibilityId("take-survey");
        attendPoll.click();
        Appium.sleep(1000);

        WebElement nameField = Appium.getDriver().findElementByAccessibilityId("name-field");
        nameField.sendKeys("Bora Kazanci");
        WebElement datePicker =
Appium.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayou
t/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Fram
eLayout/android.widget.FrameLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewG
roup/android.view.ViewGroup/android.view.ViewGroup/android.view.ViewGroup/android.widget.ScrollView/and
roid.view.ViewGroup/android.view.ViewGroup/android.view.ViewGroup");
        datePicker.click();
        Appium.sleep(1000);

        WebElement validDate = Appium.getDriver().findElementByAccessibilityId("01 June 2004");
        validDate.click();
        WebElement okButton = Appium.getDriver().findElementById("android:id/button1");
        okButton.click();
        Appium.sleep(1000);
        WebElement cityBox = Appium.getDriver().findElementByAccessibilityId("city-field");
        cityBox.click();
        Appium.sleep(500);
        WebElement cityBoxElement = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget." +

"FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayo
utCompat/" +

"android.widget.FrameLayout/android.widget.ListView/android.widget.CheckedTextView[4]");
        cityBoxElement.click();
        Appium.sleep(500);

        TouchAction touchAction = new TouchAction(Appium.getDriver());
        touchAction.press(PointOption.point(10, 2000))
                .waitAction(WaitOptions.waitOptions(Duration.ofMillis(3000)))
                .moveTo(PointOption.point(10, 200))
                .release();

        touchAction.perform();

        WebElement genderBox = Appium.getDriver().findElementByAccessibilityId("gender-field");
        genderBox.click();
        Appium.sleep(500);
        WebElement genderBoxElement = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget." +

"FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayo
utCompat/" +

"android.widget.FrameLayout/android.widget.ListView/android.widget.CheckedTextView[3]");
        genderBoxElement.click();
        Appium.sleep(1000);

        WebElement vacBox = Appium.getDriver().findElementByAccessibilityId("vaccine-field");
        vacBox.click();
        Appium.sleep(500);
        WebElement vacBoxElement = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget." +

"FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayo
utCompat/" +

"android.widget.FrameLayout/android.widget.ListView/android.widget.CheckedTextView[2]");
        vacBoxElement.click();
        Appium.sleep(1000);

        WebElement radBtn1 = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.FrameLayou
t/android.widget.FrameLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewGroup/a
ndroid.view.ViewGroup/android.view.ViewGroup/android.view.ViewGroup/android.widget.ScrollView/android.v
iew.ViewGroup/android.view.ViewGroup/android.widget.RadioButton[1]");
        WebElement radBtn2 = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.FrameLayou
t/android.widget.FrameLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewGroup/a
ndroid.view.ViewGroup/android.view.ViewGroup/android.view.ViewGroup/android.widget.ScrollView/android.v
iew.ViewGroup/android.view.ViewGroup/android.widget.RadioButton[3]");

        boolean btnFlagBefore, btnFlagAfter;
        WebElement submitButton;
        try{
            submitButton = Appium.getDriver().findElementByAccessibilityId("submit-btn");
            btnFlagBefore = false;
        }catch (Exception e){
            btnFlagBefore = true;
        }

        radBtn1.click();
        radBtn2.click();
        Appium.sleep(1000);

        try{
            submitButton = Appium.getDriver().findElementByAccessibilityId("name-field");
            btnFlagAfter = true;
        }catch (Exception e){
            btnFlagAfter = false;
        }
        try{
            Assert.assertEquals(btnFlagBefore, btnFlagAfter);
            System.out.println("        Test is passed");
        }catch (AssertionError a){
            System.out.println("    ***** Test is failed");
        }
        System.out.println("    Test Case 4: Finished");
        System.out.println("---------------------");

    }
}
```

Figure 14: Script of Test case #4

In the code snippet, the driver finds the Take Survey Button and clicks. Then the driver finds the name field and sends proper/valid keys; finds the date picker and selects a proper/valid date between the date boundaries and selects it; finds the city field and selects a city; swipes the screen at the bottom with the touch action; finds gender field and selects a random gender; finds vaccine field and selects a random vaccine; finds both of the radio buttons, however, do not click them because visibility of the Submit button needs to be checked first. Driver tries to find the Submit Button, and its result is set as a Boolean variable. After the button check, both of the radio buttons are clicked, and the visibility of the button is checked again. Finally, two parameters of the visibility are compared and printed if the test is failed or passed.

6.5. Automating Test Case #5

For our test case #5, we have the following code snippet that checks if the form is submitted successfully and goes to the page with the success message

```java
public class TestSuite5 {
    @Test
    public void testSuite5(){
        System.out.println("    Test Case 5: Started");
        Appium.sleep(4000);
        WebElement attendPoll = Appium.getDriver().findElementByAccessibilityId("take-survey");
        attendPoll.click();
        Appium.sleep(1000);

        WebElement nameField = Appium.getDriver().findElementByAccessibilityId("name-field");
        nameField.sendKeys("Bora Kazanci");
        WebElement datePicker =
Appium.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayou
t/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Fram
eLayout/android.widget.FrameLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewG
roup/android.view.ViewGroup/android.view.ViewGroup/android.view.ViewGroup/android.widget.ScrollView/and
roid.view.ViewGroup/android.view.ViewGroup/android.view.ViewGroup");
        datePicker.click();
        Appium.sleep(1000);

        WebElement validDate = Appium.getDriver().findElementByAccessibilityId("01 June 2004");
        validDate.click();
        WebElement okButton = Appium.getDriver().findElementById("android:id/button1");
        okButton.click();
        Appium.sleep(1000);

        WebElement cityBox = Appium.getDriver().findElementByAccessibilityId("city-field");
        cityBox.click();
        Appium.sleep(500);
        WebElement cityBoxElement = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget." +

"FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayo
utCompat/" +

"android.widget.FrameLayout/android.widget.ListView/android.widget.CheckedTextView[4]");
        cityBoxElement.click();
        Appium.sleep(500);

        TouchAction touchAction = new TouchAction(Appium.getDriver());
        touchAction.press(PointOption.point(10, 2000))
                .waitAction(WaitOptions.waitOptions(Duration.ofMillis(3000)))
                .moveTo(PointOption.point(10, 200))
                .release();

        touchAction.perform();

        WebElement genderBox = Appium.getDriver().findElementByAccessibilityId("gender-field");
        genderBox.click();
        Appium.sleep(500);
        WebElement genderBoxElement = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget." +

"FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayo
utCompat/" +

"android.widget.FrameLayout/android.widget.ListView/android.widget.CheckedTextView[3]");
        genderBoxElement.click();
        Appium.sleep(1000);

        WebElement vacBox = Appium.getDriver().findElementByAccessibilityId("vaccine-field");
        vacBox.click();
        Appium.sleep(500);
        WebElement vacBoxElement = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget." +

"FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayo
utCompat/" +

"android.widget.FrameLayout/android.widget.ListView/android.widget.CheckedTextView[2]");
        vacBoxElement.click();
        Appium.sleep(1000);

        WebElement radBtn1 = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.FrameLayou
t/android.widget.FrameLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewGroup/a
ndroid.view.ViewGroup/android.view.ViewGroup/android.view.ViewGroup/android.widget.ScrollView/android.v
iew.ViewGroup/android.view.ViewGroup/android.widget.RadioButton[1]");
        WebElement radBtn2 = Appium

.getDriver().findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/andr
oid.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.FrameLayou
t/android.widget.FrameLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewGroup/a
ndroid.view.ViewGroup/android.view.ViewGroup/android.view.ViewGroup/android.widget.ScrollView/android.v
iew.ViewGroup/android.view.ViewGroup/android.widget.RadioButton[3]");
        radBtn1.click();
        radBtn2.click();
        Appium.sleep(1000);
        WebElement submitButton = Appium.getDriver().findElementByAccessibilityId("submit-btn");
        submitButton.click();

        Appium.sleep(3000);

        try{
            WebElement homeBtn = Appium.getDriver().findElementByAccessibilityId("home-btn");
            System.out.println("        Test is passed");
        }catch (AssertionError a){
            System.out.println("    ***** Test is failed");
        }
        System.out.println("    Test Case 5: Finished");
        System.out.println("---------------------");

    }
}
```

Figure 15: Script of Test case #5

In the code snippet, the driver finds the Take Survey Button and clicks. Then the driver finds the name field and sends proper/valid keys; finds the date picker and selects a proper/valid date between the date boundaries and selects it; finds the city field and selects a city; swipes the screen at the bottom with the touch action; finds gender field and selects a random gender; finds vaccine field and selects a random vaccine; finds both of the radio buttons and both of the radio buttons are clicked; finds the visible submit button and clicks it. Then driver tries to find the success message on the submission screen and prints a message if the test is failed or passed.

7. Automation Experience

First of all, it was pretty challenging to set up the development environment of Appium and Android Emulator since team members are using different operating systems like Windows and macOS, the setup process for them was quite different, and error messages were not clear for example on macOS if the one did not set up the correct version of Android SDK, since not all of them are not supported by Appium Server, it throws a bunch of nonsense errors, and you have to search through the internet and GitHub issues to solve the issue. It can be said that setting up the development environment took a longer time than designing test cases and automated testing. The process was much easier on the web. However, after installing Appium, it can be said that what hindered the automation process was mostly installing the necessary tools rather than writing the required tools.

After installing Appium on the computer, it took a lot of time to integrate the program we wrote with Appium. At first, our application was working through Expo, and we had to build an APK again while testing each test case. With subsequent research, we tried to remove Expo from the project and move forward. Thus, our test automation development speed problem was solved. Due to our previous experience in Java, we used the Java and Appium Java libraries to write the automation code. Although we had a minor problem running the project on the Android emulator, it was quickly taken care of. One other minor issue was that with React Native projects, Appium is unable to locate elements with id's, one has to include an accessibility id to the component, but some components we are using do not support accessibility id's to locate such components; we had to use xPath's of those elements.

8. Selenium vs. Appium

Since Appium was mainly designed to automate testing in all kinds of mobile applications, it was mainly developed as an HTTP server based on Node JS. So, anyone who wants to use Appium for mobile app automation testing would need to install NodeJS on their system to set up the same. So NodeJS installation is one of the prerequisite criteria for Appium installation by any user.

On the other hand, Selenium is basically designed to create automation testing for any website or web application at any time in any browser. The web app mainly runs in one of the specific web browsers, so Selenium is automatically used in the same way. It primarily targets a particular purpose of interaction with the web application with each feature by automatically controlling the action of different types of web browsers. It has defined browser actions with a specific approach to run easily in a browser without any manual intervention.[3]

Another difference is that we use JavaScript to write automation code in Appium. Selenium IDE, on the other hand, has an interactive interface to prepare test cases and run them individually, in batches, or by command.

The interactive nature of Selenium IDE makes it not only very easy to set up and install but also very easy to learn how to use. It was enough to watch many instructional videos for learning.
On the other hand, Appium depends on all other programs and technologies. First, it requires using a programming language proficiently enough to write test cases. While this is valid for Selenium as well, it is not necessary. Similarly, Appium requires an actual mobile device or an emulator to run, while Selenium only works in a browser. [4]

The nature of mobile and desktop and the difference between the two make the testing experience wildly different. Appium needs to support all major mobile platforms, so it requires additional setup, while Selenium manages HTML pages and is, therefore, lighter in many ways.

However, there are similarities in each other. Selenium finds views by IDs of HTML tags, and Apium finds views with XML tags in Android testing.

Both are excellent tools for automating tests, running multiple tests in short time intervals and multiple times. Our testing experience has proven both tools to be very helpful and extremely efficient at what they do. However, Appium is much more difficult to install and use due to limitations.

9. Repository Link

https://github.com/AhmetIsk/Covid-19-Vaccine-Survey-App

References

[1] "Introduction to appium studio: Key benefits and features," *Software Testing Help*, 03-Mar-2022. [Online]. Available: https://www.softwaretestinghelp.com/appium-studio-overview-tutorial-1/. [Accessed: 30-Mar-2022].

[2] “Feature,” *Appium Discuss*. [Online]. Available: https://discuss.appium.io/c/feature. [Accessed: 30-Mar-2022].

[3] “Appium vs selenium: Topmost 2 comparison in detail to know,” *EDUCBA*, 17-Mar-2021. [Online]. Available: https://www.educba.com/appium-vs-selenium/#:~:text=Appium%20is%20used%20for%20automated,not%20support%20system%20%26%20mobile%20applications. [Accessed: 30-Mar-2022].

[4] “Appium vs selenium: Key differences,” *BrowserStack*, 18-Jun-2020. [Online]. Available: https://www.browserstack.com/guide/appium-vs-selenium. [Accessed: 30-Mar-2022].