Bilkent University

CS 458 Project 2

Section 1

Group members

Izaan Aamir	22001488
Ghulam Ahmed	22101001
Mehshid Atiq	22101335
Gülin Çetinus	21902115

1. Introduction

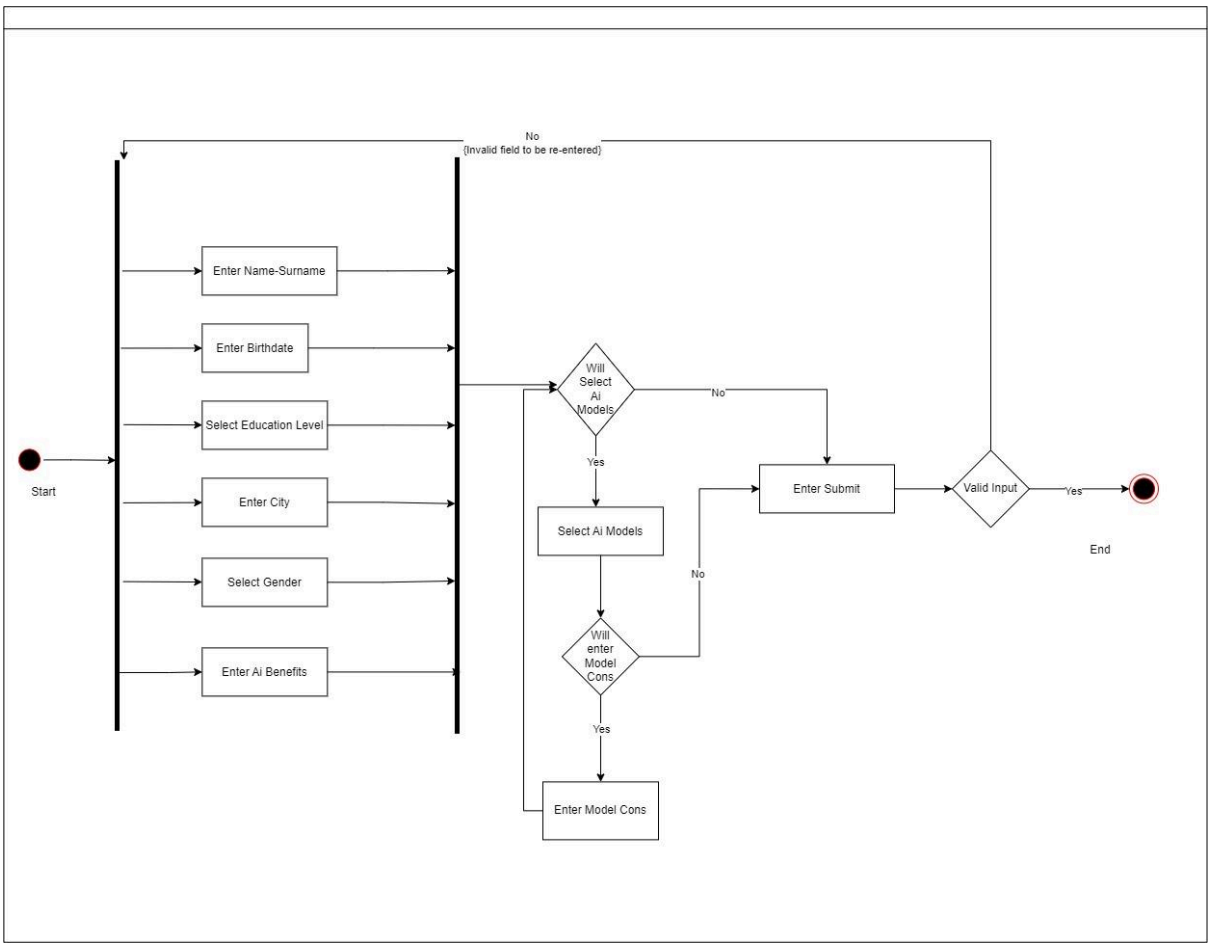
For this project, we've developed a survey within an Android application focusing on artificial intelligence usage. The survey collects detailed information, including names, surnames, dates of birth, education levels, genders, and which AI tools users have previously engaged with.

- **Mandatory Fields:** All provided fields are required to be filled by the participants, ensuring comprehensive data collection. The sole exception is the selection of AI models; participants may opt to leave this optional, catering to those with limited exposure to specific AI tools.
- **AI Model Selection:** Upon choosing an AI tool, users have the flexibility to input any drawbacks or cons associated with each selected tool. However, this is not mandatory, allowing users who may not have critical feedback to proceed without input in this area.
- **Date of Birth Validation:** Participants must be 12 years of age or older to partake in the survey. The form will reject dates of birth that result in ages under 12 or any dates that are set in the future, ensuring that all respondents meet the age criteria for meaningful feedback.
- **Dynamic Fields for AI Models:** If a user deselects an AI model after previously selecting it and entering corresponding cons, the input field for those cons will automatically be removed. This dynamic adjustment keeps the survey form clean and focused only on the relevant information.
- **Default Education Level:** To streamline data entry, the default education level is set to 'High School.' Users can change this as necessary, but the default option simplifies the process for a significant user base.
- **Gender Selection:** The form is designed to prevent users from selecting more than one gender, maintaining data integrity and clarity in demographic information.
- **Beneficial Use Cases:** At the survey's conclusion, there is a section for users to share beneficial AI applications in daily life. This open-ended question aims to gather insights on the positive impacts of AI from the perspectives of the respondents.

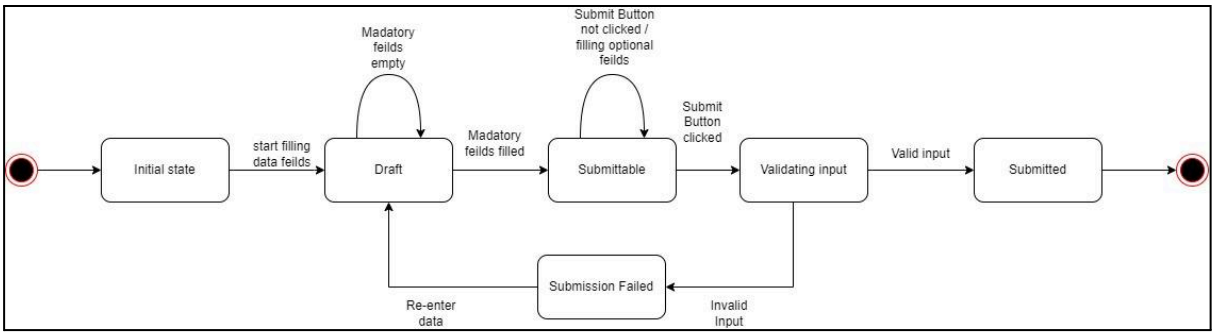
These enhancements and nuances are integrated to ensure the survey is not only comprehensive but also user-friendly, accommodating a wide range of experiences and feedback on artificial intelligence usage.
The app's source code can be found here.

2. UML Diagrams and Screenshots

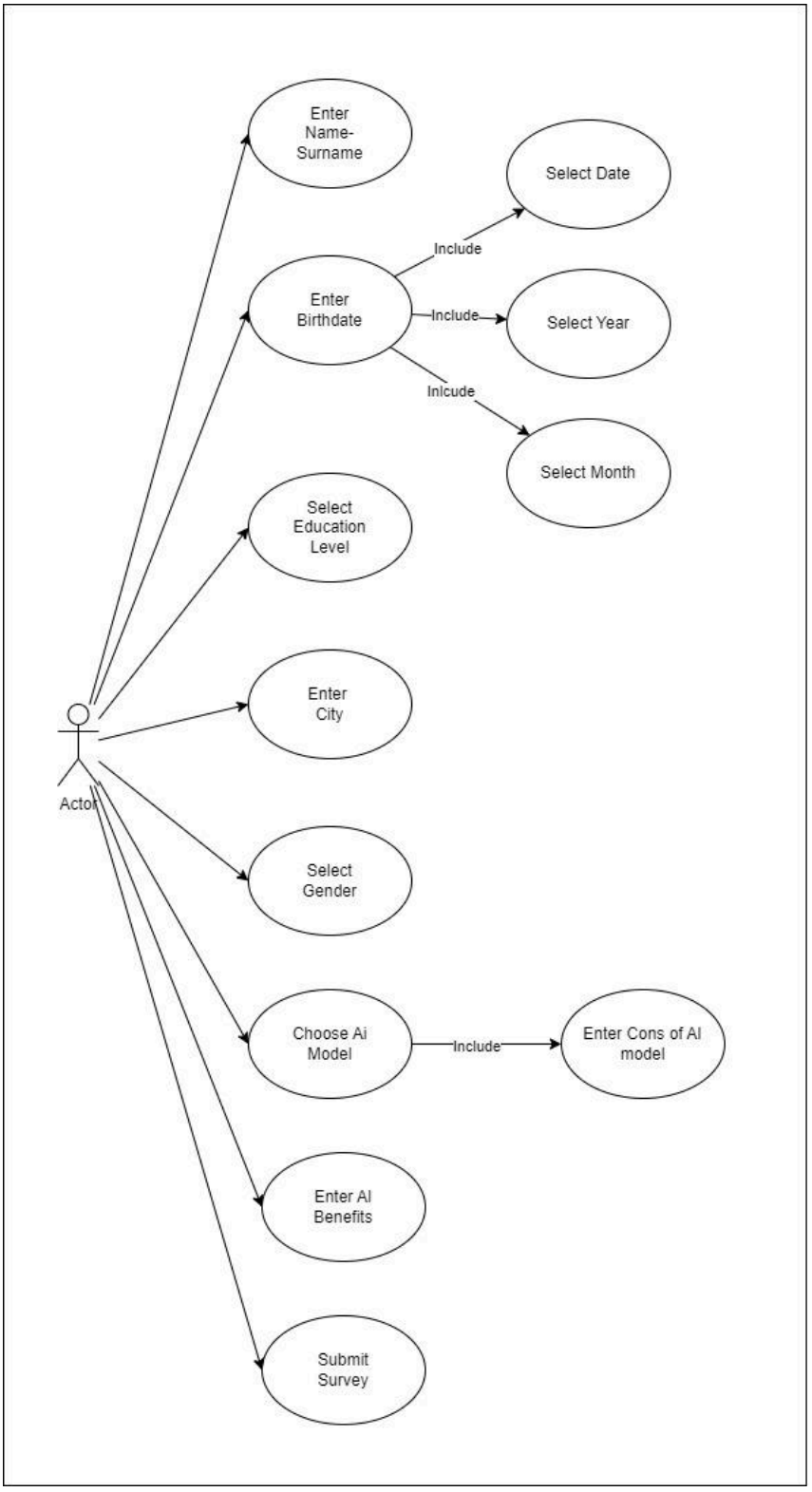
1. Activity Diagram



2. State Diagram

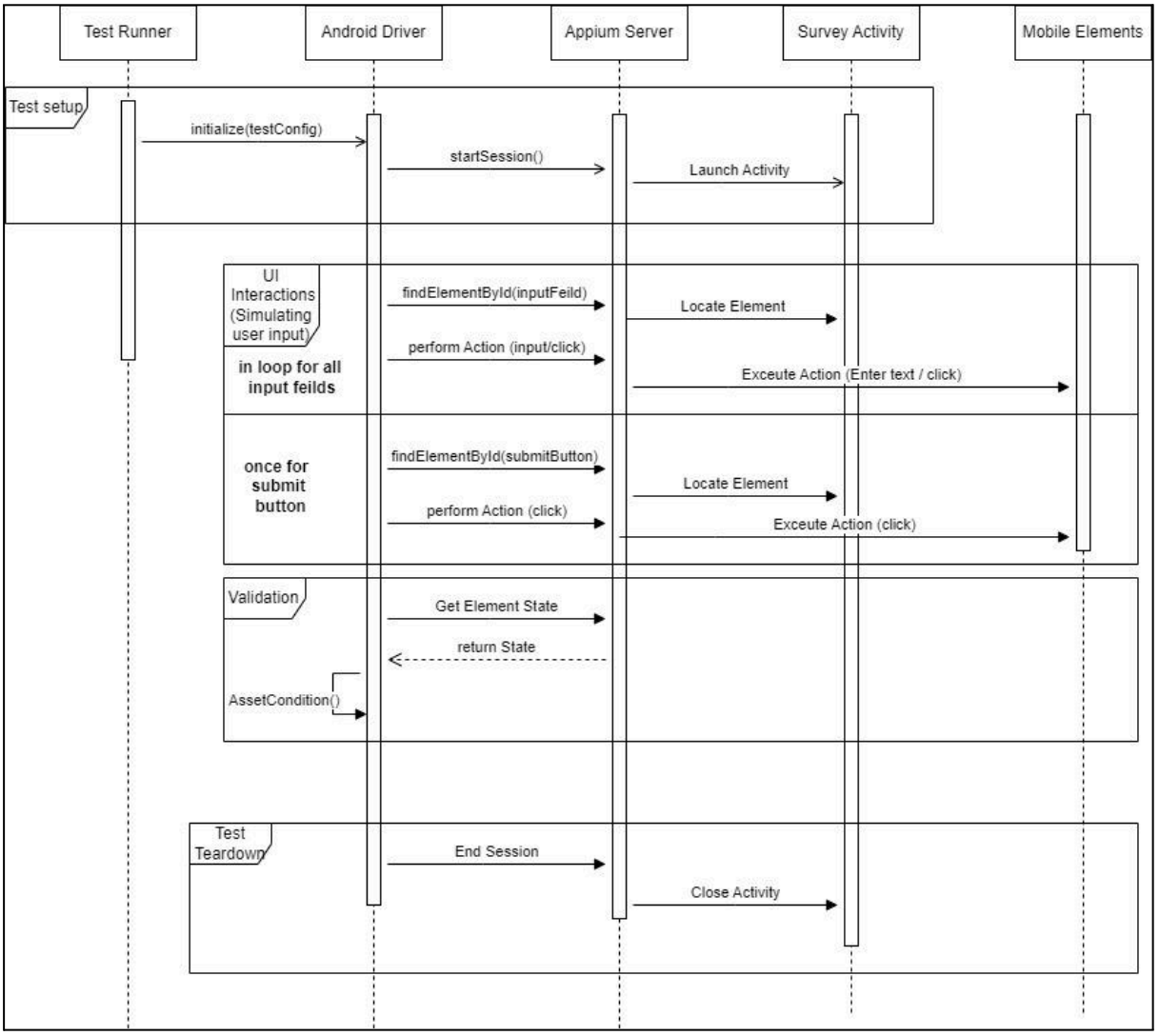


3. Usecase Diagram

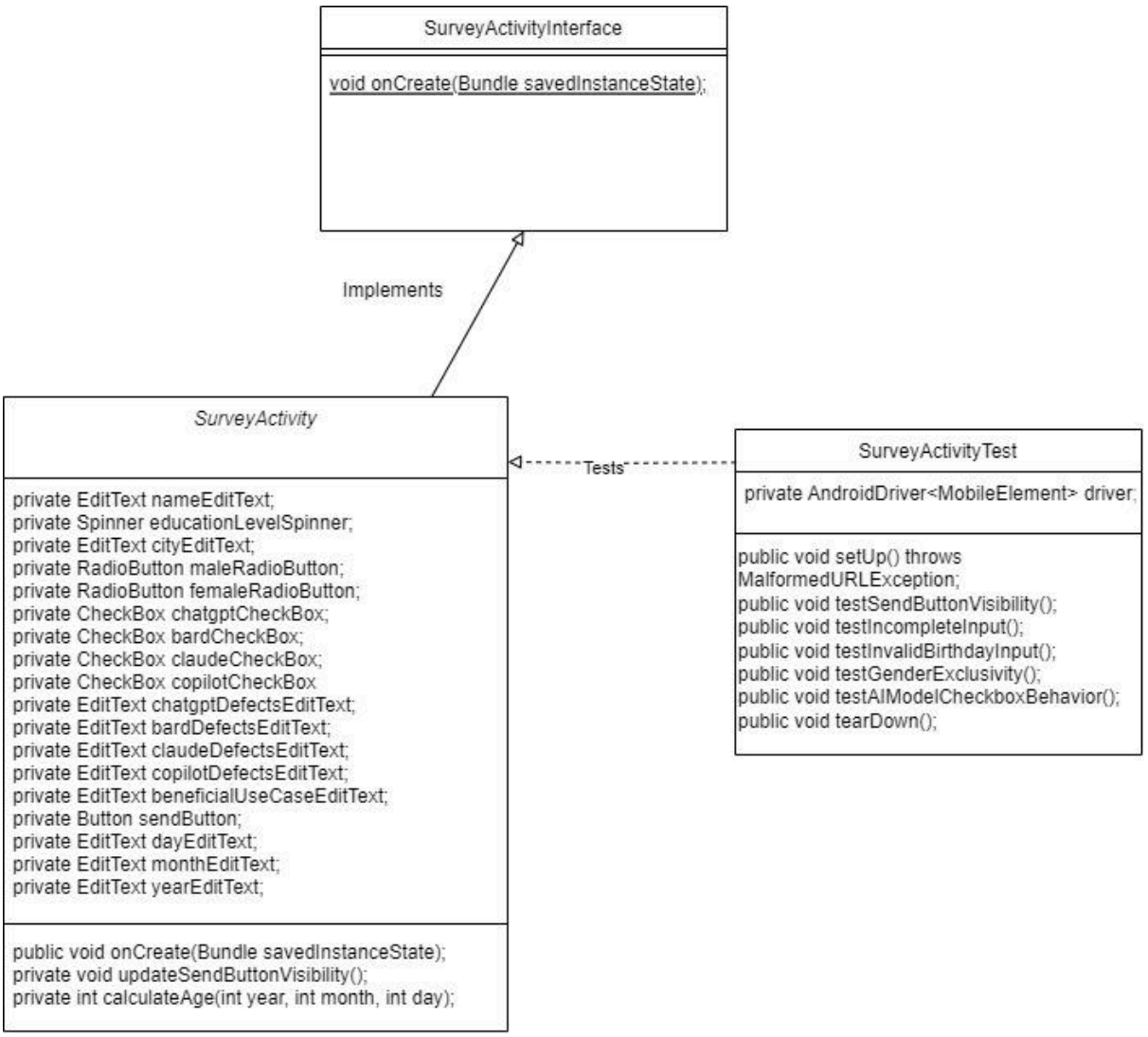


4. Sequence Diagram

The sequence diagram provided simplifies the representation of user inputs and validation checks by grouping them into singular streams. This approach helps in keeping the diagram uncluttered and easy to follow. It however overlooks the specific handling and individual validation of each input field. Consequently, it doesn't detail the distinct interactions required for different types of inputs or the nuanced validation each field undergoes.



5. Class Diagram



3. Manual Test Cases

1. testSendButtonVisiblity

This test case, testSendButtonVisibility, aims to verify the visibility of the "Send" button in a survey application under specific conditions. Initially, it populates all required fields in the survey form with valid data, including selecting an AI model and entering defects related to it. After filling out the form, it explicitly waits to check if the "Send" button becomes visible, affirming the application's logic to display the

button only when all necessary fields are filled correctly. The test then clears one of the fields to simulate incomplete data and waits briefly to assert that the "Send" button becomes invisible or is not present. This tests and validates the application's functionality to hide the send button again if the form is incomplete, ensuring users cannot submit incomplete forms.

```
@Test
    public void testSendButtonVisibility() {
        // Fill all fields with valid data
        MobileElement nameEditText =
driver.findElementById("nameEditText");
        nameEditText.sendKeys("John Doe");

        MobileElement dayEditText =
driver.findElementById("dayEditText");
        dayEditText.sendKeys("15");

        MobileElement monthEditText =
driver.findElementById("monthEditText");
        monthEditText.sendKeys("6");

        MobileElement yearEditText =
driver.findElementById("yearEditText");
        yearEditText.sendKeys("1990");

        MobileElement educationLevelSpinner =
driver.findElementById("educationLevelSpinner");
        educationLevelSpinner.click();

driver.findElementByXPath("//*[@text='Bachelor']").click();

        MobileElement cityEditText =
driver.findElementById("cityEditText");
        cityEditText.sendKeys("New York");

        MobileElement maleRadioButton =
driver.findElementById("maleRadioButton");
        maleRadioButton.click();

        MobileElement chatgptCheckBox =
driver.findElementById("chatgptCheckBox");
        chatgptCheckBox.click();
```

```
        MobileElement chatgptDefectsEditText =
driver.findElementById("chatgptDefectsEditText");
        chatgptDefectsEditText.sendKeys("Some defects");

        MobileElement beneficialUseCaseEditText =
driver.findElementById("beneficialUseCaseEditText");
        beneficialUseCaseEditText.sendKeys("AI can assist in
various tasks");

        // Wait for the send button to become visible
        WebDriverWait wait = new WebDriverWait(driver, 2);

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("s
endButton")));

        // Assert that the send button is visible
        MobileElement sendButton =
driver.findElementById("sendButton");
        assertTrue(sendButton.isDisplayed());

        // Clear the "BeneficialUseCase" field
        beneficialUseCaseEditText.clear();

        // Wait for a short duration
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Assert that the send button is not visible
        List<MobileElement> sendButtonList =
driver.findElementsById("sendButton");
        assertTrue(sendButtonList.isEmpty()); // Assert that the
send button is not found
    }
```

2. testInvalidBirthdayInput:

The test case testInvalidBirthdayInput specifically checks the application's ability to validate a user's age based on the birthday provided in the survey form. It ensures that the form does not allow the submission if the entered birthday corresponds to a user being less than 12 years old. By programmatically entering a birthday that makes the user under the age limit and attempting to proceed, the test verifies two critical behaviors: firstly, that the "Send" button becomes inaccessible or hidden to prevent form submission under this condition, and secondly, that an appropriate error message is displayed to the user, indicating the reason for the restriction. This test is crucial for enforcing age-related rules or requirements within the application's data collection process.

```
@Test
    public void testInvalidBirthdayInput() {
        // Fill all fields except birthday
        MobileElement nameEditText =
driver.findElementById("nameEditText");
        nameEditText.sendKeys("John Doe");

        MobileElement educationLevelSpinner =
driver.findElementById("educationLevelSpinner");
        educationLevelSpinner.click();

driver.findElementByXPath("//*[@text='Bachelor']").click();

        MobileElement cityEditText =
driver.findElementById("cityEditText");
        cityEditText.sendKeys("New York");

        MobileElement maleRadioButton =
driver.findElementById("maleRadioButton");
        maleRadioButton.click();

        MobileElement beneficialUseCaseEditText =
driver.findElementById("beneficialUseCaseEditText");
        beneficialUseCaseEditText.sendKeys("AI can assist in
various tasks");

        // Enter an invalid birthday (less than 12 years old)
```

```
        MobileElement dayEditText =
driver.findElementById("dayEditText");
        dayEditText.sendKeys("15");

        MobileElement monthEditText =
driver.findElementById("monthEditText");
        monthEditText.sendKeys("6");

        MobileElement yearEditText =
driver.findElementById("yearEditText");
        int currentYear =
Calendar.getInstance().get(Calendar.YEAR);
        yearEditText.sendKeys(String.valueOf(currentYear - 10));
// Set year to 10 years ago

        // Wait for a short duration
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Assert that the send button is not visible
        List<MobileElement> sendButtonList =
driver.findElementsById("sendButton");
        assertTrue(sendButtonList.isEmpty()); // Assert that the
send button is not found

        // Assert that the error message is displayed
        MobileElement birthdayErrorTextView =
driver.findElementById("birthdayErrorTextView");
        assertTrue(birthdayErrorTextView.isDisplayed());
        assertEquals("User must be at least 12 years old",
birthdayErrorTextView.getText());
    }
```

3. testGenderExclusivity :

The `testGenderExclusivity` test case ensures that gender selection within a survey form is mutually exclusive. After populating the form with valid data, the test toggles between "Male" and "Female" radio buttons to confirm that selecting one

automatically deselects the other, ensuring users can only choose one gender. The process validates the form's logic for handling gender options correctly. Upon successfully verifying gender selection exclusivity, the test submits the form and checks for a confirmation message, affirming the form's overall submission process works as intended when integrating gender selection behavior.

```
@Test
    public void testGenderExclusivity() {
        // Fill all fields with valid data
        MobileElement nameEditText =
driver.findElementById("nameEditText");
        nameEditText.sendKeys("John Doe");

        MobileElement dayEditText =
driver.findElementById("dayEditText");
        dayEditText.sendKeys("15");

        MobileElement monthEditText =
driver.findElementById("monthEditText");
        monthEditText.sendKeys("6");

        MobileElement yearEditText =
driver.findElementById("yearEditText");
        yearEditText.sendKeys("1990");

        MobileElement educationLevelSpinner =
driver.findElementById("educationLevelSpinner");
        educationLevelSpinner.click();

driver.findElementByXPath("//*[@text='Bachelor']").click();

        MobileElement cityEditText =
driver.findElementById("cityEditText");
        cityEditText.sendKeys("New York");

        MobileElement beneficialUseCaseEditText =
driver.findElementById("beneficialUseCaseEditText");
        beneficialUseCaseEditText.sendKeys("AI can assist in
various tasks");

        // Click on the maleRadioButton
```

```
        MobileElement maleRadioButton =
driver.findElementById("maleRadioButton");
        maleRadioButton.click();

        WebDriverWait wait = new WebDriverWait(driver, 10);

wait.until(ExpectedConditions.attributeToBe(maleRadioButton,
"checked", "true"));
        assertEquals("true",
maleRadioButton.getAttribute("checked"));
        assertEquals("false",
driver.findElementById("femaleRadioButton").getAttribute("checked
"));

        // Click on the femaleRadioButton
        MobileElement femaleRadioButton =
driver.findElementById("femaleRadioButton");
        femaleRadioButton.click();


wait.until(ExpectedConditions.attributeToBe(femaleRadioButton,
"checked", "true"));
        assertEquals("true",
femaleRadioButton.getAttribute("checked"));
        assertEquals("false",
maleRadioButton.getAttribute("checked"));

        MobileElement chatgptCheckBox =
driver.findElementById("chatgptCheckBox");
        chatgptCheckBox.click();
        MobileElement chatgptDefectsEditText =
driver.findElementById("chatgptDefectsEditText");
        chatgptDefectsEditText.sendKeys("Some defects");

        // Wait for the send button to become visible

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("s
endButton")));

        MobileElement sendButton =
driver.findElementById("sendButton");
        sendButton.click();
```

```
        // Assert that the survey was submitted successfully
        String toastMessage =
driver.findElementByXPath("//android.widget.Toast").getText();
        assertEquals("Survey submitted", toastMessage);
    }
```

4. testIncompleteInput

The testIncompleteInput assesses the form's submission logic for optional AI model selections, confirming submission success without these selections and evaluating send button visibility upon altering input completion. Initially, it populates mandatory fields—excluding AI model checkboxes—to simulate a user submitting the form with these options left unselected. Despite this omission, the form submits successfully, as evidenced by a "Survey submitted" toast message, “indicating AI model selection is optional”.

Subsequently, the test clears a filled field to observe the send button's behavior (AI benefits feild). The send button disappears, demonstrating the form's “dynamic” response to incomplete inputs and enforcing complete data before submission.

```
@Test
    public void testIncompleteInput() {
        // Fill all fields except AI model checkboxes
        MobileElement nameEditText =
driver.findElementById("nameEditText");
        nameEditText.sendKeys("John Doe");

        MobileElement dayEditText =
driver.findElementById("dayEditText");
        dayEditText.sendKeys("15");

        MobileElement monthEditText =
driver.findElementById("monthEditText");
        monthEditText.sendKeys("6");

        MobileElement yearEditText =
driver.findElementById("yearEditText");
        yearEditText.sendKeys("1990");
```

```
        MobileElement cityEditText =
driver.findElementById("cityEditText");
        cityEditText.sendKeys("New York");

        MobileElement maleRadioButton =
driver.findElementById("maleRadioButton");
        maleRadioButton.click();

        MobileElement beneficialUseCaseEditText =
driver.findElementById("beneficialUseCaseEditText");
        beneficialUseCaseEditText.sendKeys("AI can assist in
various tasks");

        // Get the selected education level from the spinner
        MobileElement educationLevelSpinner =
driver.findElementById("educationLevelSpinner");
        String selectedEducationLevel =
educationLevelSpinner.getText();

        // Assert that the default education level is selected
        assertEquals("", selectedEducationLevel);

        // Wait for the send button to become visible
        WebDriverWait wait = new WebDriverWait(driver, 2);

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("s
endButton")));

        // Click the send button
        MobileElement sendButton =
driver.findElementById("sendButton");
        sendButton.click();

        // Assert that the survey was submitted successfully
        String toastMessage =
driver.findElementByXPath("//android.widget.Toast").getText();
        assertEquals("Survey submitted", toastMessage);

        // Clear the "BeneficialUseCase" field
        beneficialUseCaseEditText.clear();

        // Wait for a short duration
        try {
```

```
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Assert that the send button is not visible
        List<MobileElement> sendButtonList =
driver.findElementsById("sendButton");
        assertTrue(sendButtonList.isEmpty()); // Assert that the
send button is not found
    }
```

5. testAIModelCheckboxBehavior

The `testAIModelCheckboxBehavior` examines the link between selecting AI models and the visibility of their associated text fields in a survey form. It verifies that when an AI model, like ChatGPT, is selected and its defect description field is filled, deselecting the AI model checkbox causes the relevant text field to disappear, reflecting the conditional display logic tied to AI model selections. The test ensures the UI correctly reacts to user inputs by hiding detail fields when their corresponding selections are undone, maintaining a clean and relevant interface based on user choices.

```
@Test
    public void testAIModelCheckboxBehavior() {
        // Fill all fields with valid data
        MobileElement nameEditText =
driver.findElementById("nameEditText");
        nameEditText.sendKeys("John Doe");

        MobileElement dayEditText =
driver.findElementById("dayEditText");
        dayEditText.sendKeys("15");

        MobileElement monthEditText =
driver.findElementById("monthEditText");
        monthEditText.sendKeys("6");

        MobileElement yearEditText =
driver.findElementById("yearEditText");
```

```
        yearEditText.sendKeys("1990");

        MobileElement educationLevelSpinner =
driver.findElementById("educationLevelSpinner");
        educationLevelSpinner.click();

driver.findElementByXPath("//*[@text='Bachelor']").click();

        MobileElement cityEditText =
driver.findElementById("cityEditText");
        cityEditText.sendKeys("New York");

        MobileElement maleRadioButton =
driver.findElementById("maleRadioButton");
        maleRadioButton.click();

        MobileElement beneficialUseCaseEditText =
driver.findElementById("beneficialUseCaseEditText");
        beneficialUseCaseEditText.sendKeys("AI can assist in
various tasks");

        // Select the ChatGPT checkbox and add a description
        MobileElement chatgptCheckBox =
driver.findElementById("chatgptCheckBox");
        chatgptCheckBox.click();
        MobileElement chatgptDefectsEditText =
driver.findElementById("chatgptDefectsEditText");
        String chatgptDescription = "Some defects";
        chatgptDefectsEditText.sendKeys(chatgptDescription);

        // Assert that the ChatGPT defects EditText is visible
and contains the description
        assertTrue(chatgptDefectsEditText.isDisplayed());
        assertEquals(chatgptDescription,
chatgptDefectsEditText.getText());

        // Deselect the ChatGPT checkbox
        chatgptCheckBox.click();

        // Assert that the ChatGPT defects EditText is not
present in the DOM
        List<MobileElement> chatgptDefectsEditTextList =
driver.findElementsById("chatgptDefectsEditText");
```

```
        assertTrue(chatgptDefectsEditTextList.isEmpty());

        // Select the ChatGPT checkbox again
        chatgptCheckBox.click();

        // Wait for the ChatGPT defects EditText to be visible
again
        WebDriverWait wait = new WebDriverWait(driver, 2);
        chatgptDefectsEditText = (MobileElement)
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("c
hatgptDefectsEditText")));

        // Assert that the ChatGPT defects EditText is visible
and the description is same
        assertTrue(chatgptDefectsEditText.isDisplayed());
        assertEquals("Some defects",
chatgptDefectsEditText.getText());

        // Wait for the send button to become visible

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("s
endButton")));

        // Click the send button
        MobileElement sendButton =
driver.findElementById("sendButton");
        sendButton.click();

        // Assert that the survey was submitted successfully
        String toastMessage =
driver.findElementByXPath("//android.widget.Toast").getText();
        assertEquals("Survey submitted", toastMessage);
    }
```

4. Appium

4.1. Appium Capabilities

1. **Cross-platform:** Appium supports the automation of mobile apps across both iOS and Android platforms, allowing testers to write tests once and execute them on multiple platforms.
2. **Native, hybrid, and mobile web apps:** Appium supports testing native, hybrid, and mobile web applications, providing flexibility in testing various types of mobile applications.
3. **Multiple programming language support:** Appium allows testers to write automation scripts in multiple programming languages such as Java, JavaScript, Python, Ruby, and more, enabling teams to use the language they are most comfortable with.
4. **Integration with testing frameworks:** Appium integrates well with popular testing frameworks like Selenium WebDriver, XCTest, and Google's Espresso, allowing testers to leverage existing frameworks and tools.
5. **Actual device and emulator testing:** Appium supports testing on real devices and emulators/simulators, enabling comprehensive testing across different environments.
6. **Access to native APIs:** Testers can access and interact with native device APIs (e.g., GPS, camera, contacts) using Appium, allowing for testing of various functionalities within the app.
7. **Support for gestures:** Appium supports gestures and interactions such as tap, swipe, pinch, and rotate, enabling testers to simulate fundamental user interactions with the app.
8. **Element identification:** Appium provides mechanisms to locate elements on the app's user interface (UI) using attributes like ID, name, XPath, CSS selectors, etc., facilitating interaction with UI elements during test execution.
9. **Parallel execution:** Test execution can be parallelized across multiple devices, allowing faster execution and reducing overall testing time.
10. **Integration with CI/CD pipelines:** Appium can be integrated seamlessly into Continuous Integration and Continuous Deployment (CI/CD) pipelines, enabling automated testing as part of the software development lifecycle.

4.2. Appium Code

The Appium code can be seen here.

5. Automation Experience

With Appium, the project could be tested way faster than testing it manually, making the process of finding the bugs of the project way faster than doing it manually. Since Appium is available in multiple programming languages, we were able to choose the one we were the most familiar with, in this case Java, and use the tool without needing to learn another programming language.

6. Comparison of Selenium and Appium

Selenium excels in automating tests for web applications, offering comprehensive coverage across various browsers and operating platforms. It's the go-to framework for validating web application functionality within a wide array of web browser environments. Appium, on the other hand, is specifically designed for the mobile app testing domain, adept at handling both straightforward native applications and more intricate hybrid and web apps. It plays a crucial role in ensuring application integrity across the diverse spectrum of mobile devices and their operating systems.

However, given that Appium is built on the Selenium framework, they share notable similarities. Both frameworks support multiple programming languages and adopt the same client-server architecture, facilitating a seamless testing process. The primary distinction lies in their application domains: Selenium is the framework of choice for web-based projects, ensuring compatibility and performance across different web browsers and operating systems, whereas Appium extends these capabilities into mobile testing, catering to both Android and iOS platforms. This foundational connection between Appium and Selenium bridges web and mobile testing, offering a unified approach to automated testing across the full spectrum of digital applications.

7. Test Automation Contribution

Reflecting on our personal experience with mobile automation in this project, here's how I'd summarize the key points:

- **Efficiency and Speed:** We found that automated testing dramatically sped up the testing process, handling repetitive tasks far more quickly than we could have managed manually.

- **Enhanced Reliability:** The ability to conduct comprehensive testing coverage with automation played a crucial role in enhancing the reliability of our final product. It allowed us to uncover and fix more errors early in the development cycle.

- **Error Detection:** Automation’s broad test coverage was a game-changer for us. It enabled the detection of a higher volume of errors, which was essential for maintaining the quality of the product before its release.

- **Reusability of Test Scripts:** One of the standout benefits for us was the reusability of test scripts. This aspect saved us considerable time and resources as the application evolved across different versions.

- **Challenges and Considerations:** Despite the evident benefits, we faced challenges such as the initial setup time and the need for technical expertise to write and maintain test scripts. These challenges highlighted the importance of balancing automated and manual testing for us.

Conclusion: Our personal experience underscores the transformative impact of test automation on software development. It not only made our testing phase more efficient but also contributed significantly to delivering a high-quality final product. This experience reinforced the critical role of test automation, particularly in projects aiming for high quality under tight deadlines.