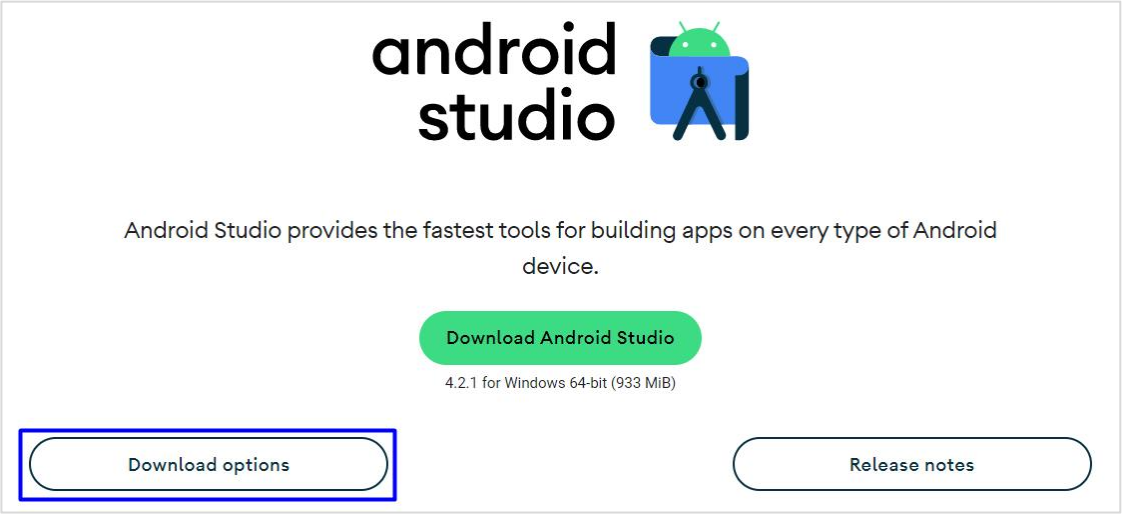
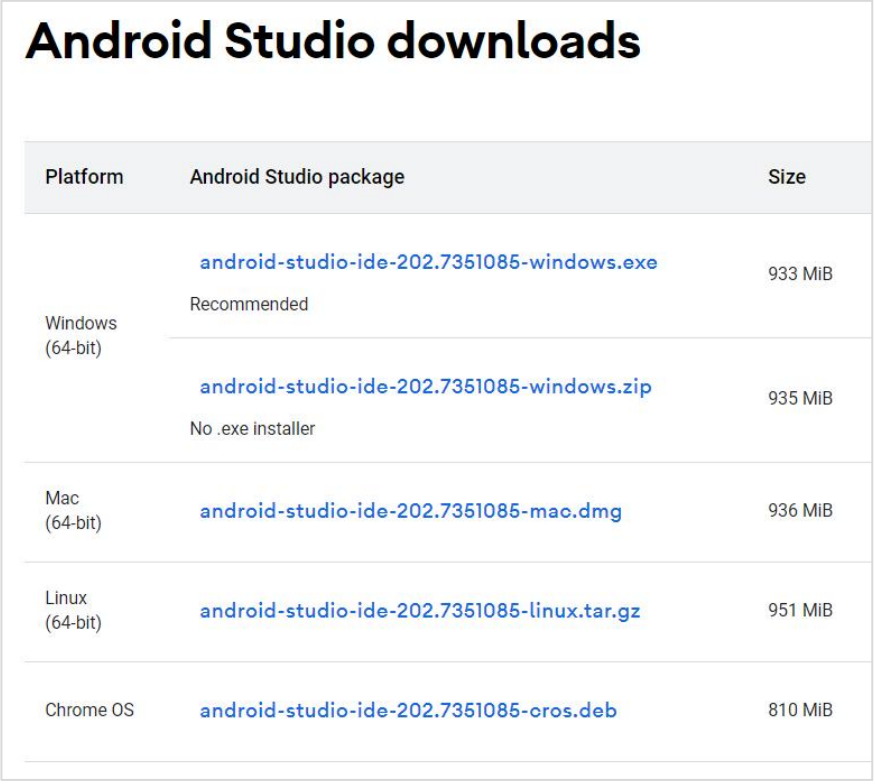
Exercises: Testing Techniques

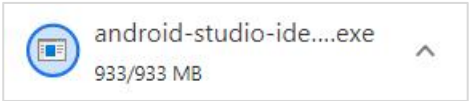
Problems for exercises and homework for the "Software Quality Assurance" course from the official "Applied Programmer" curriculum.

1. Install Android Studio

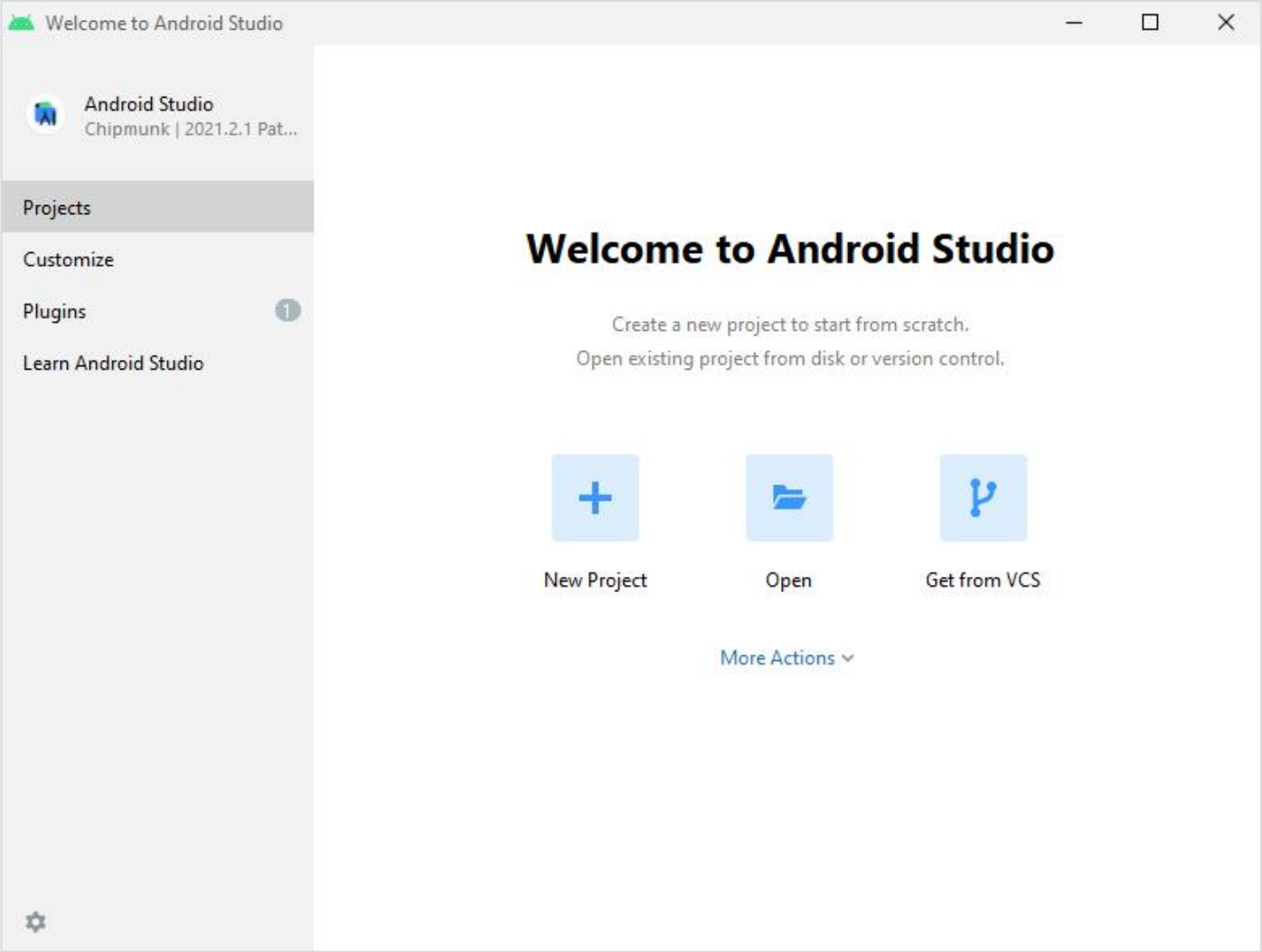
For this exercise, you should install **`Android Studio`**, which contains the **`Android SDK`**. Download it from here: https://developer.android.com/studio. Go to **`[Download options]`** and **download** the installation for your OS:





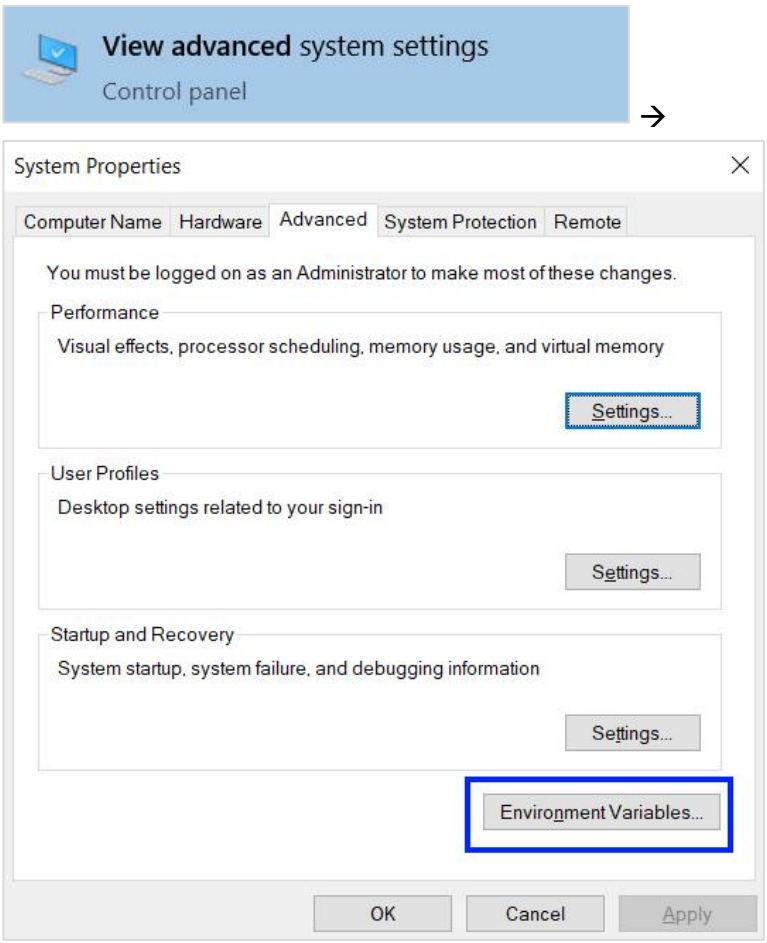


Open the file and follow the **installation guide**. When ready, **`Android Studio`** should open:

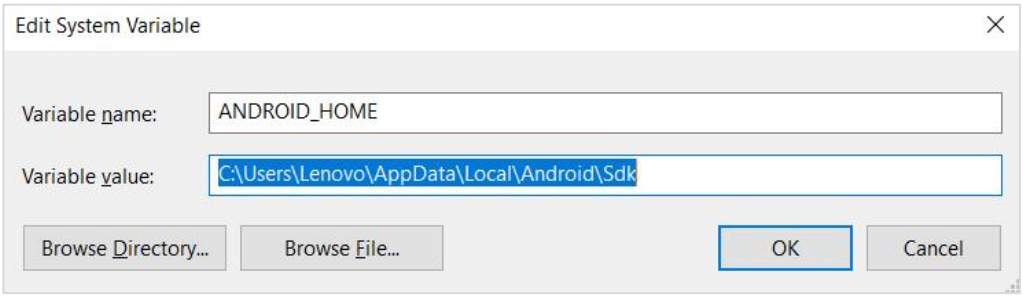


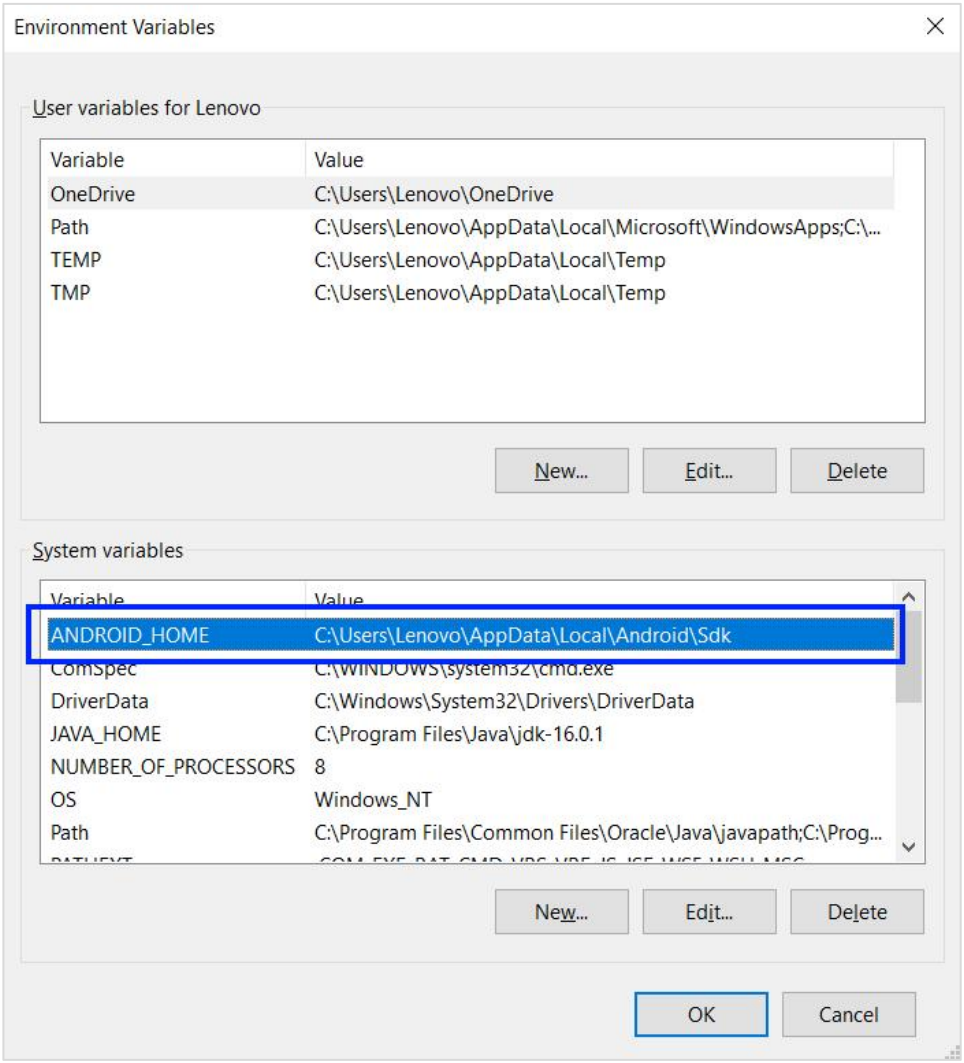
2. Set ANDROID_HOME

The environment variable **`ANDROID_HOME`** is necessary for **`Appium`** to connect to **`Android Studio`**. Type **`[View advanced system settings]`** in Windows search bar and press **`[Environment Variables…]`**:

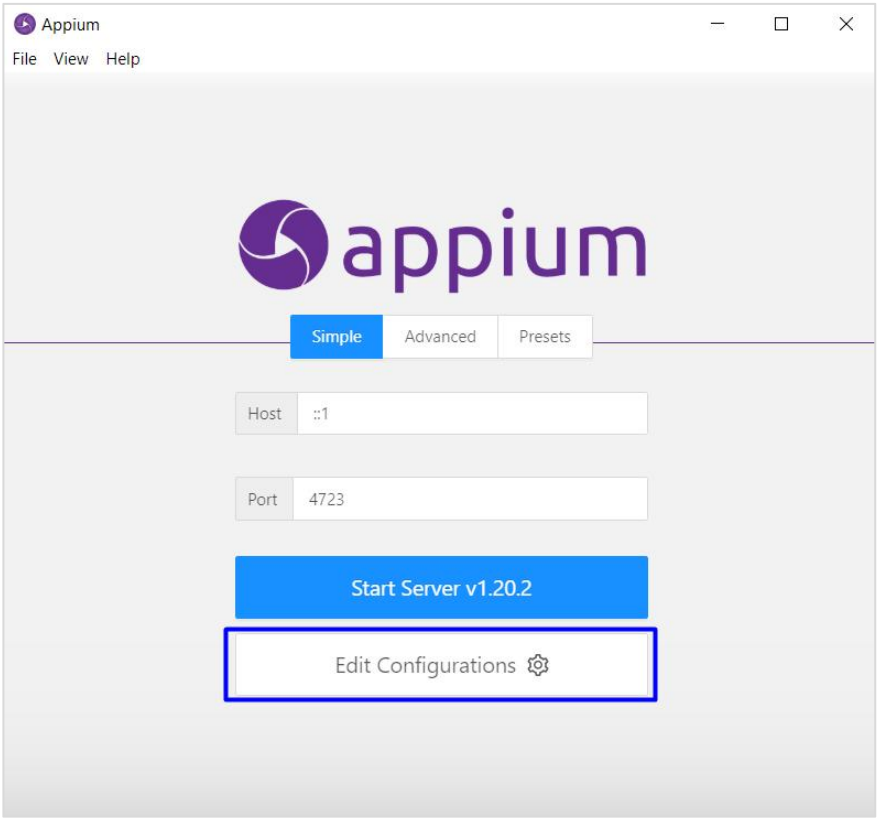


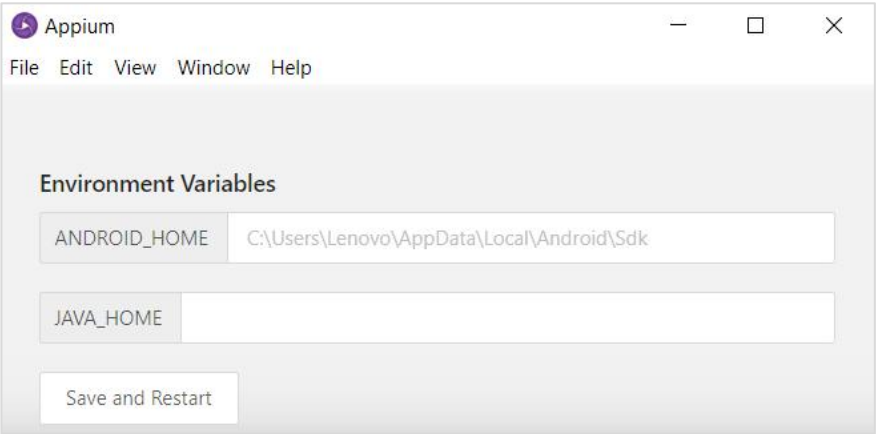
On the next window, press **[New…]** under **[System variables]** and add the "**ANDROID_HOME**" variable with the **path to Android SDK**. The path is "**C:\Users\<your-username>\AppData\Local\Android\Sdk**" by default:





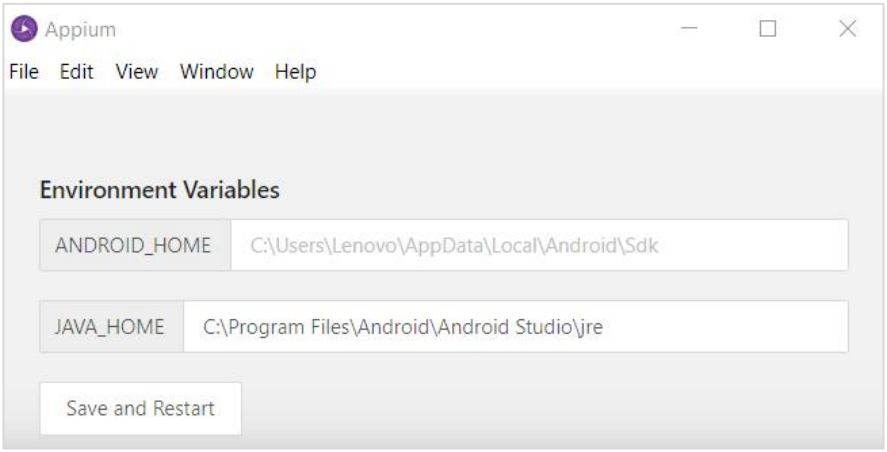
If you have **`Appium`** opened, close it, and **open it again**. Go to **`[Edit Configurations]`** and make sure the **path** for "**ANDROID_HOME**" is right or **set it up** manually:





3. Set JAVA_HOME

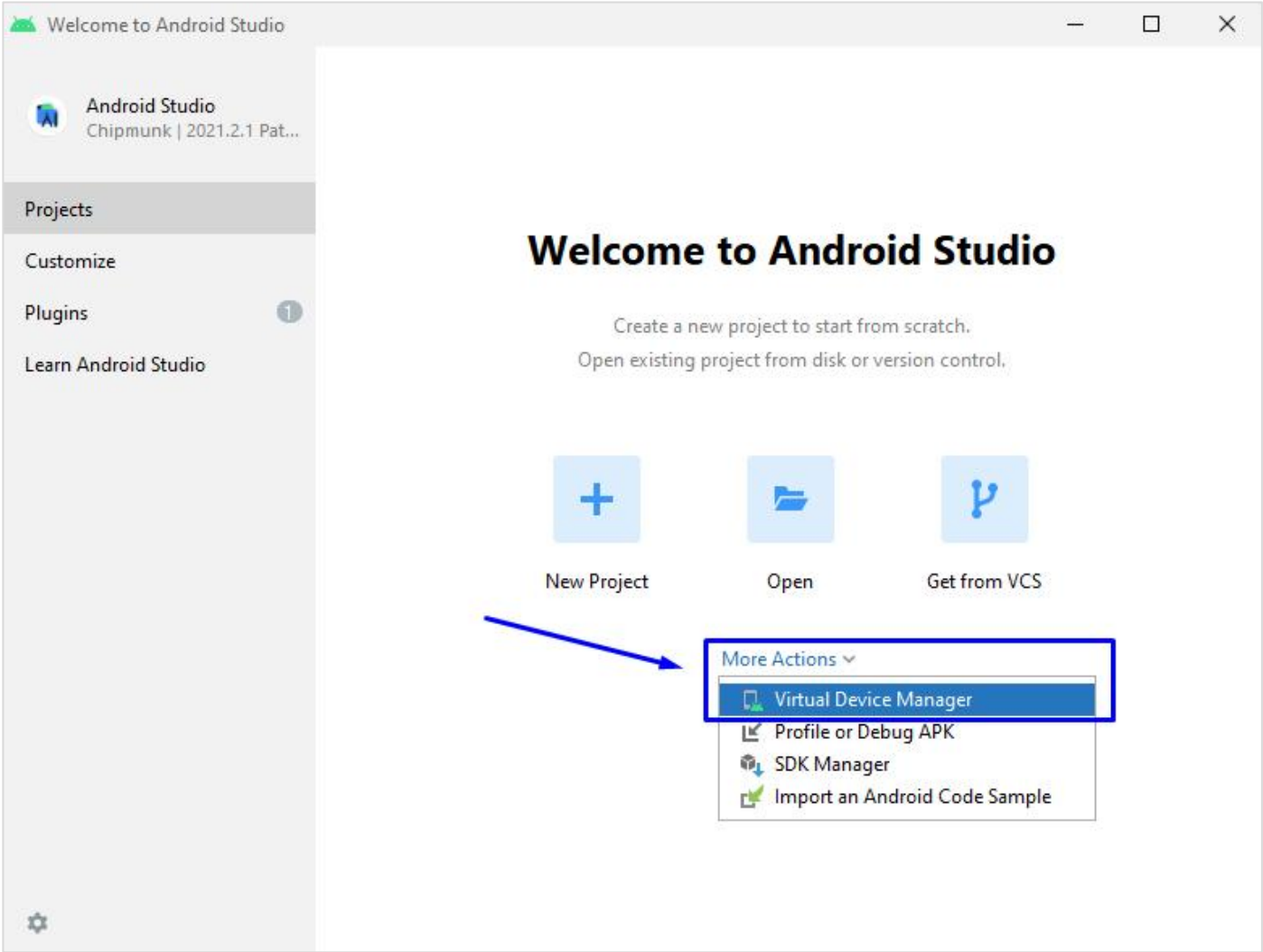
`Appium` needs **`Java`** to run and "**`JAVA_HOME`**" is necessary to point out the path to **`Java`**. We have not installed Java, but **`Android Studio`** has it as part of its files. The path to **`Java`** files is "**`C:\Program Files\Android\Android Studio\jre`**". Put the path as a value of "**`JAVA_HOME`**" in the **Appium configurations**:



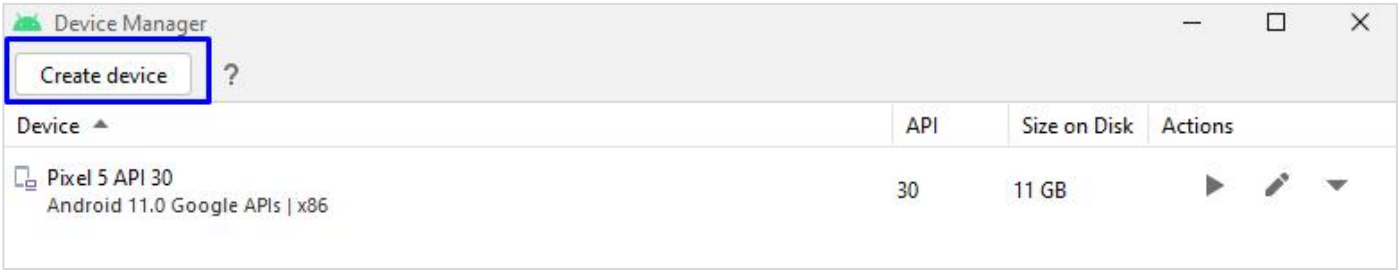
Close and **open `Appium`** again.

4. Create and Run an Android Virtual Device (AVD)

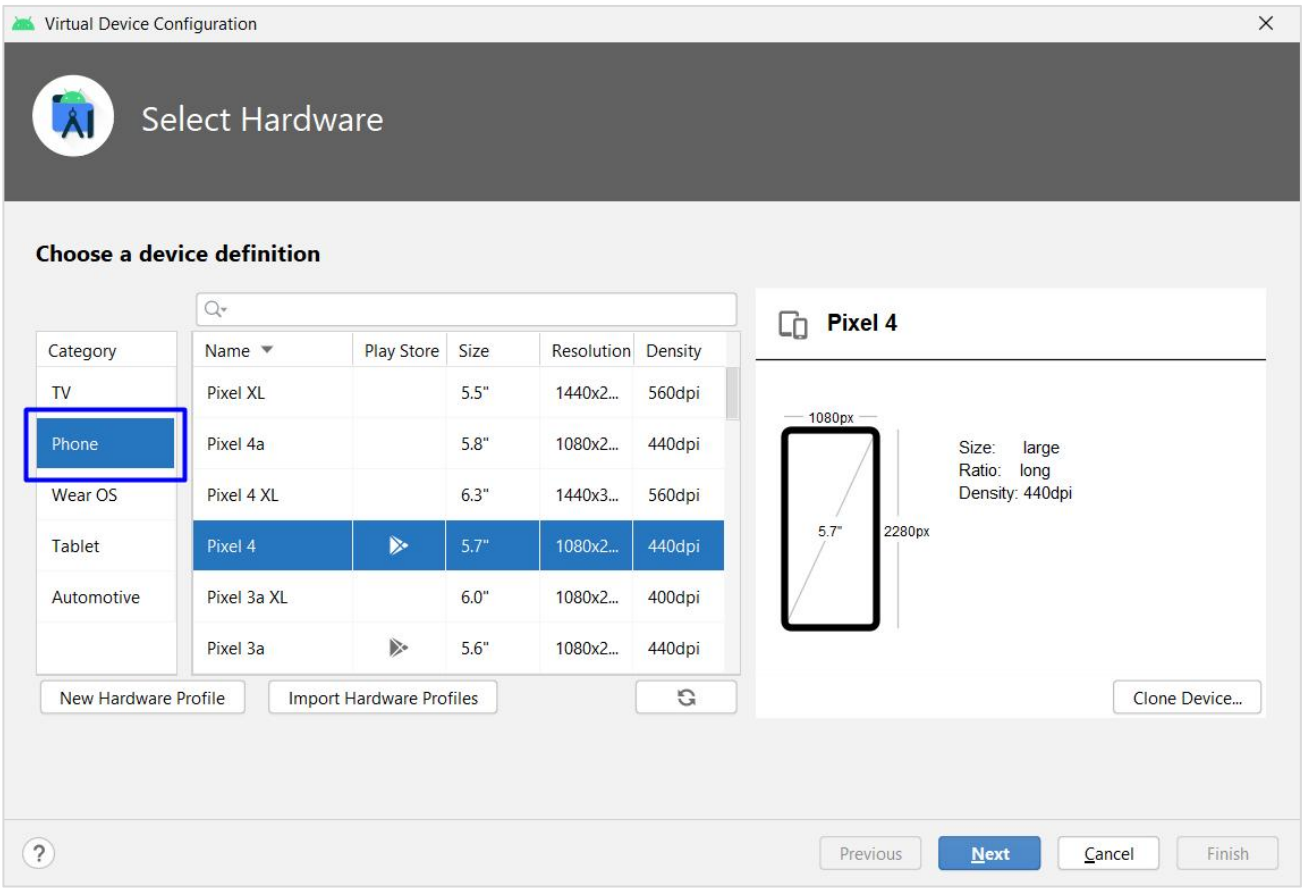
Go back to **`Android Studio`** and **click on [Virtual Device Manager] under [More Actions]**:

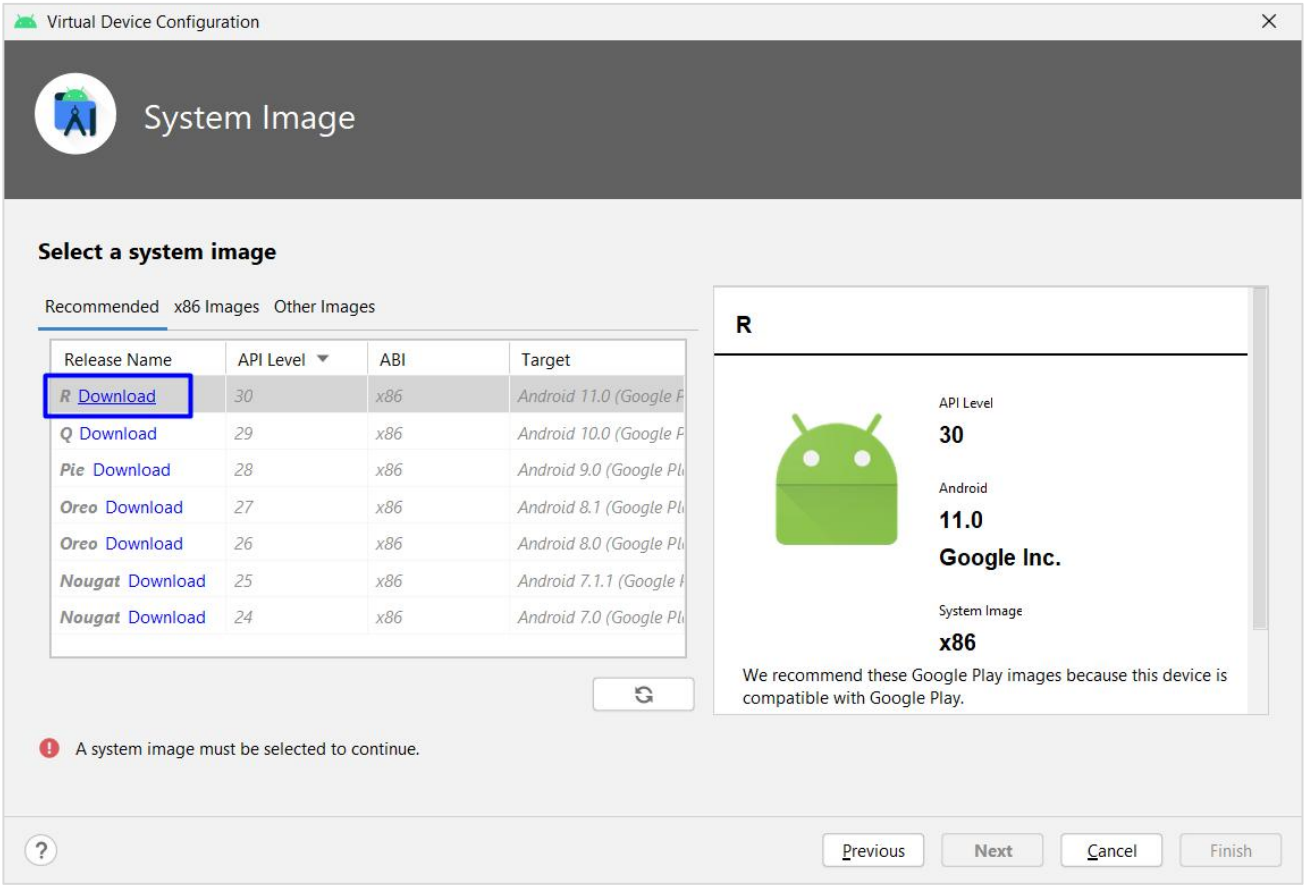


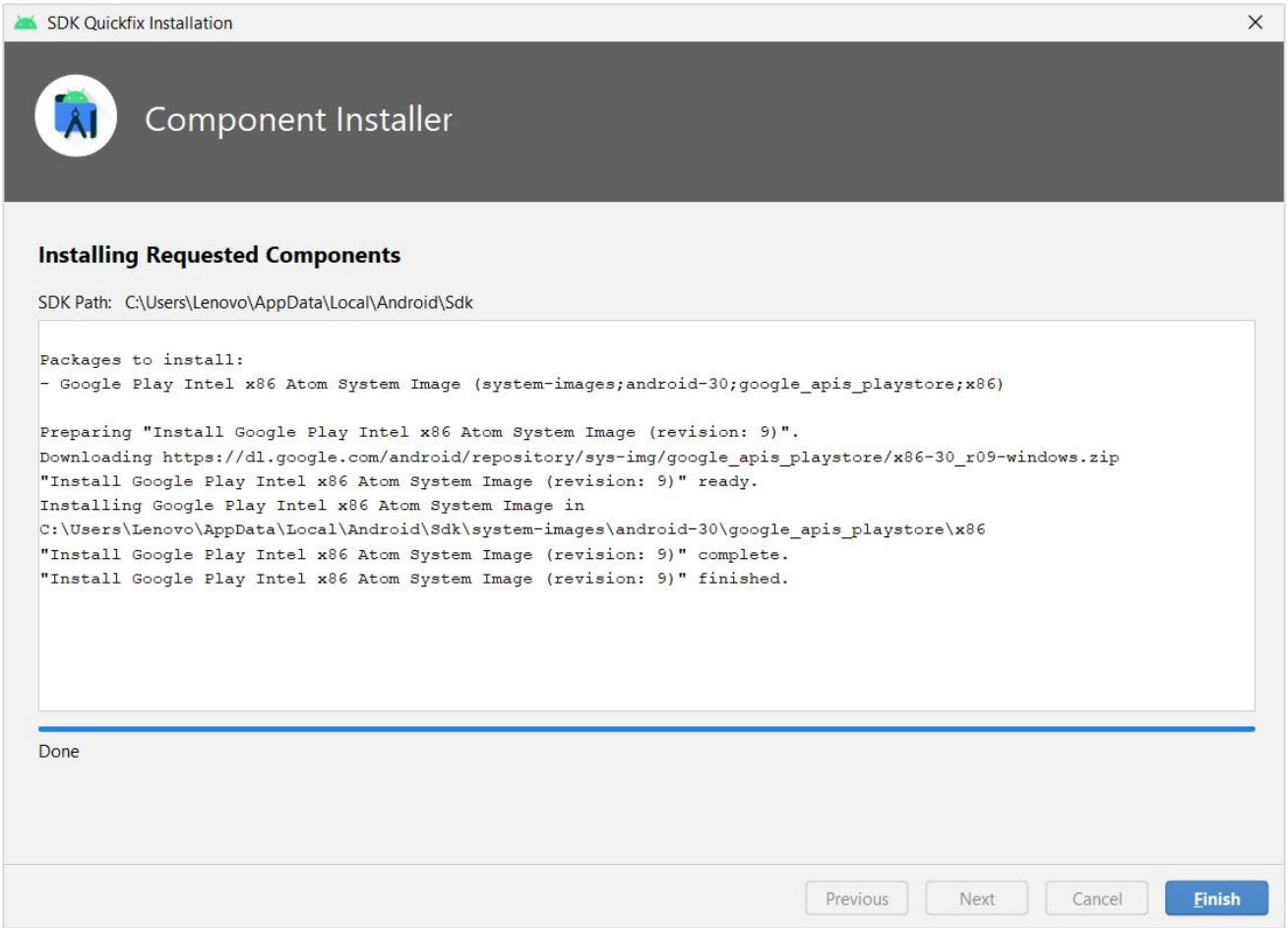
Now you should create an **`Android virtual machine`**, on which tests will be executed. Press the **`[Create Virtual Device…]`** button on the next window:



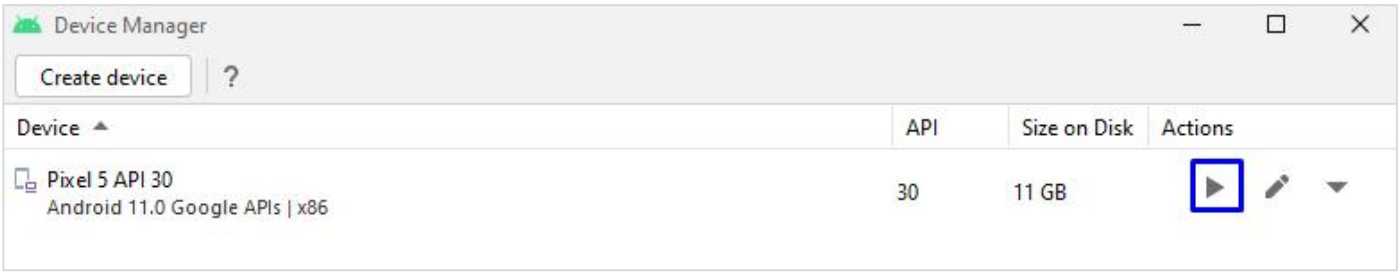
You can choose whichever **phone device definition** you want for your **AVD**. Here we will choose "**Pixel 4**". Then, select the **first recommended system image**, press **[Download]** and **wait** for the downloading of the **SDK** to finish. Steps are shown below:







Next, you can **change the name** of the device, if you want. Press **[Finish]** at the end. Your **AVD** is now created. **Start** it by pressing its **[Launch]** button:



Wait for the **AVD** to load:

5. Appium Automated Tests for the "Summator" Android App

Implement **Appium UI automated tests** for the following sample **Android mobile app** "Summator": https://github.com/nakov/AndroidApp-Summator. You can get its .apk file from here (you also have it in the resources): https://github.com/nakov/AndroidApp-Summator/releases/tag/v1.0.

The Automated Testing Scenario

1. Open the **`Summator` app**
2. Test with **valid** and **invalid data**
   - With valid data: assert that result is **correct**
   - With invalid data: assert that "**`error`**" is displayed in the result field

Get **ids** of elements, using **`Appium`**, connected to an **`Android Virtual Device`** in **Android Studio**.

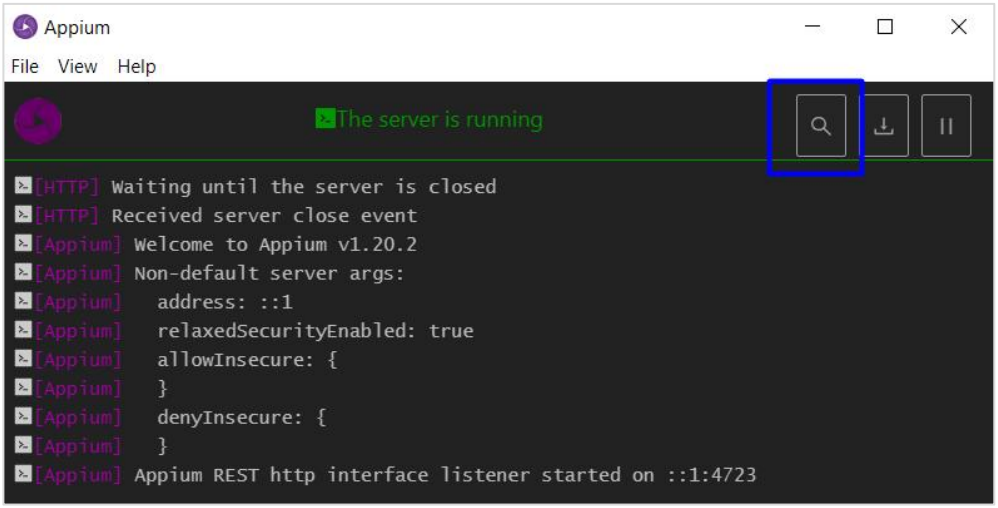
Use C#, NUnit, Appium, Android Driver, Android SDK (for the Android Device Emulator).

Hints and Guidelines

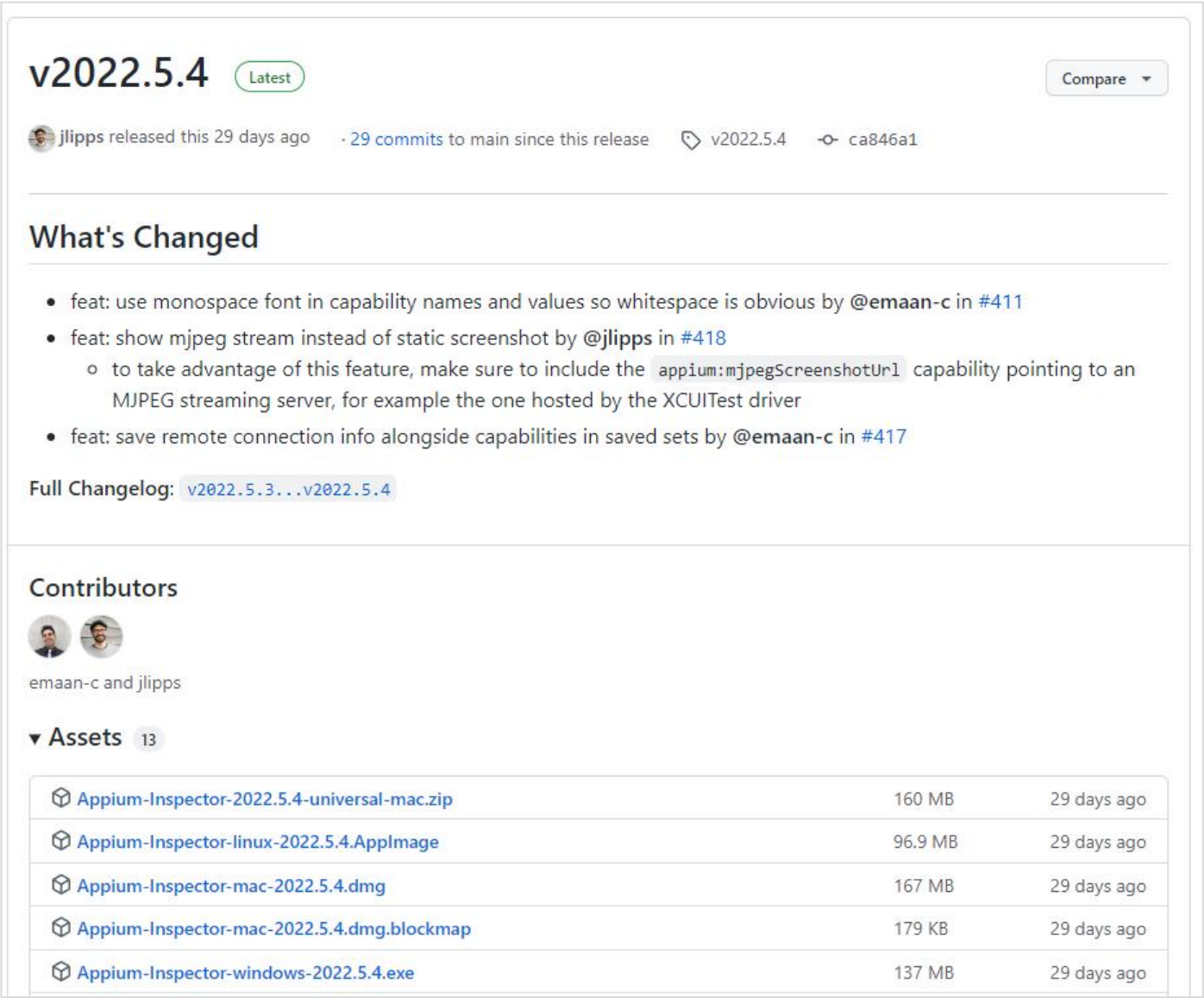
First, let's connect **`Appium`** to the **AVD**, so that we can get **ids** of elements, as we did in the previous exercise with the WAD UI Recorder and Windows Calculator App.

Open **`Appium`** and **start server** with host **`[::1]`**. Then, press **`[Start Inspector Session]`**:

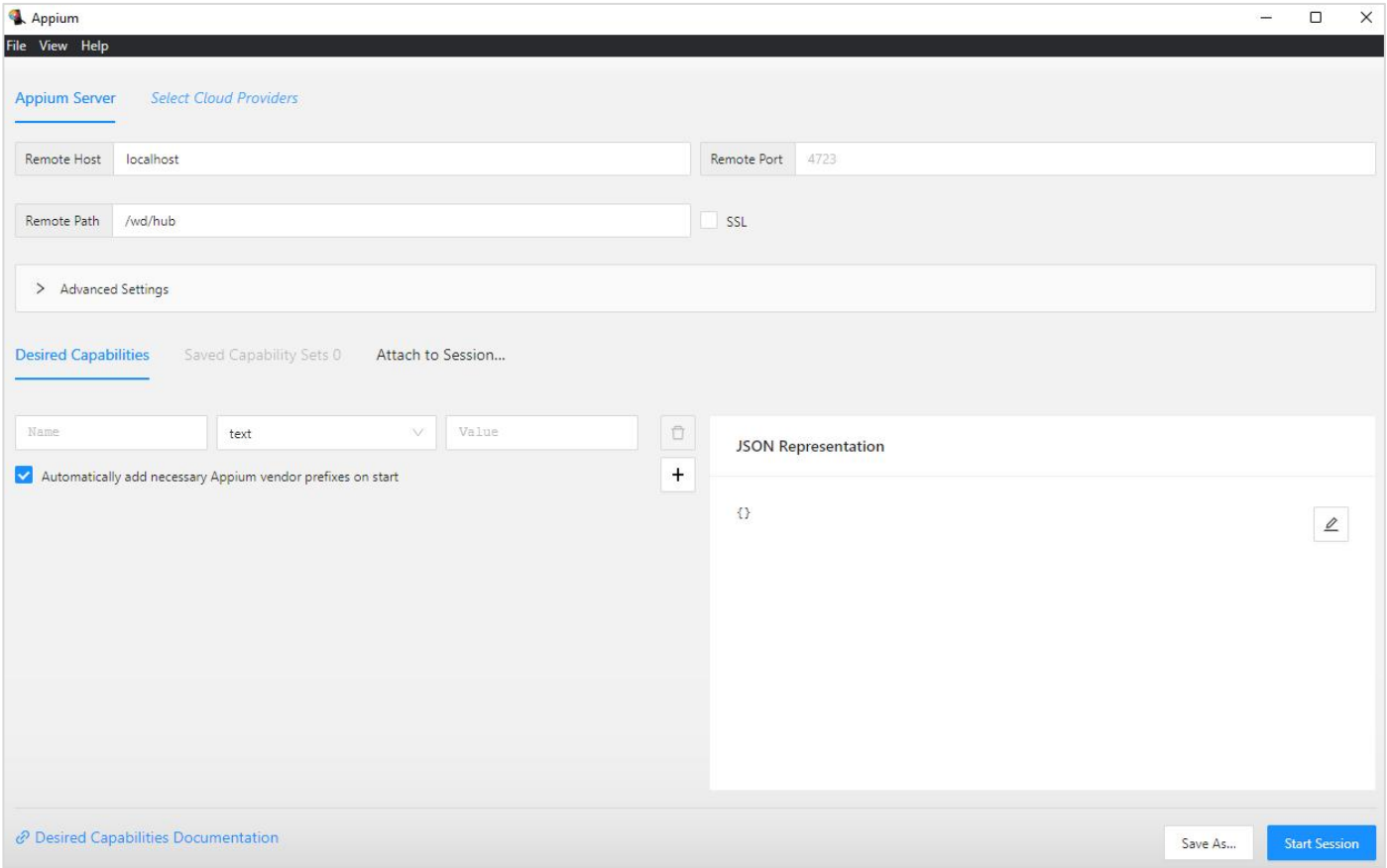
→



This will open a tab in your browser to the "**Appium Inspector**" click on [**Releases**], download it and install it:

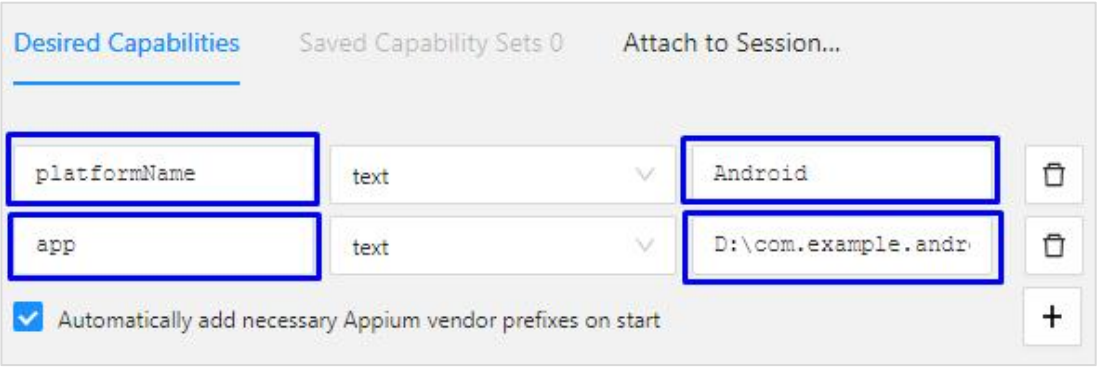


When you open it you will be presented with this screen:



You need to put **`localhost`** in the "**Remote Host**" textbox and **`/wd/hub`** in the "**Remote Path**".

Under "**Desired Capabilities**" you need to add **`platformName`** and **`app`**. Note that **`platformName`** should be "**`Android`**" and the "**`app`**" value should be the **path** to the "**`com.example.androidappsummator.apk`**" app (it is part of the resources for this exercise). In our case, the file is placed directly in the "**`D:`**" drive:

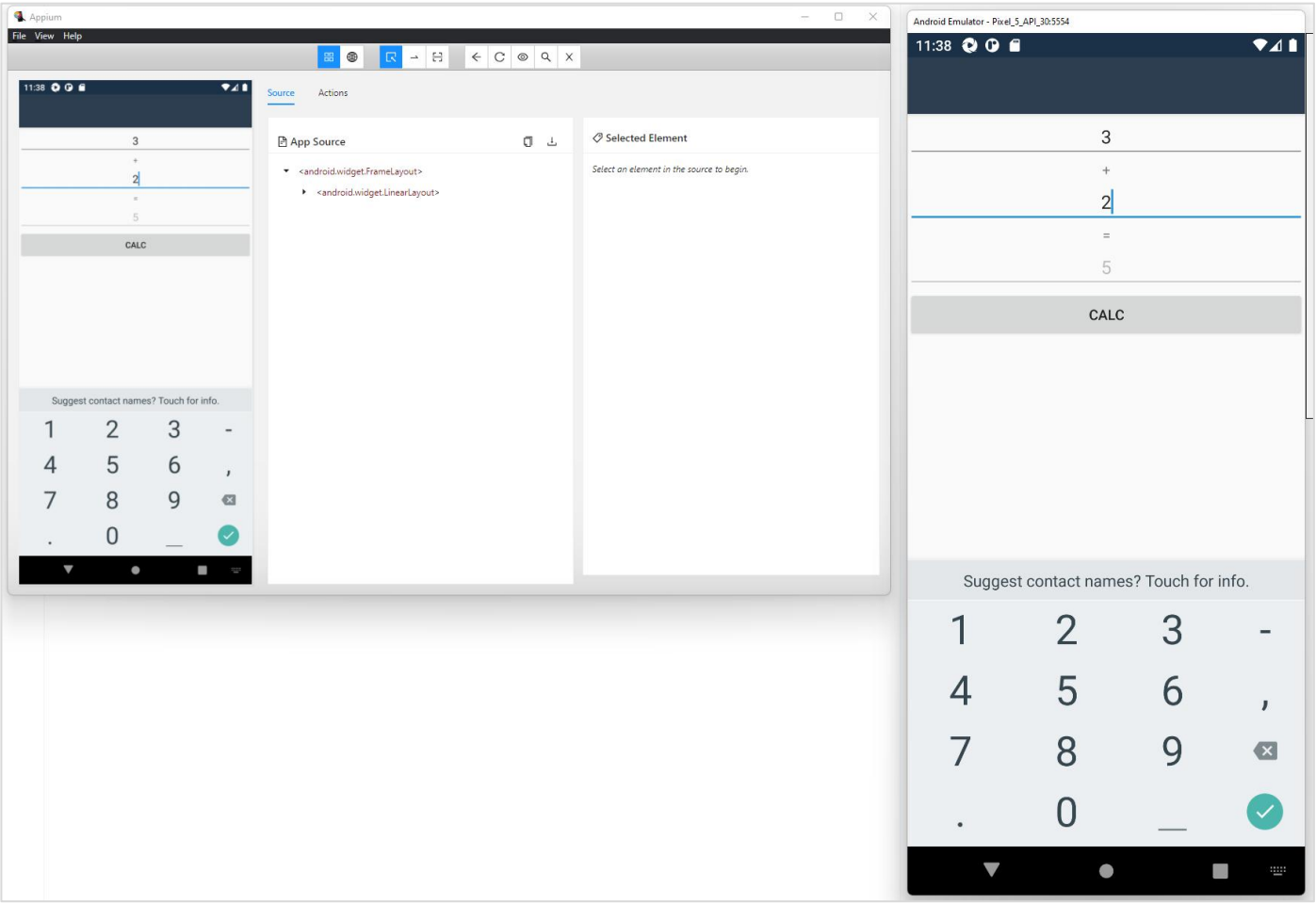


→

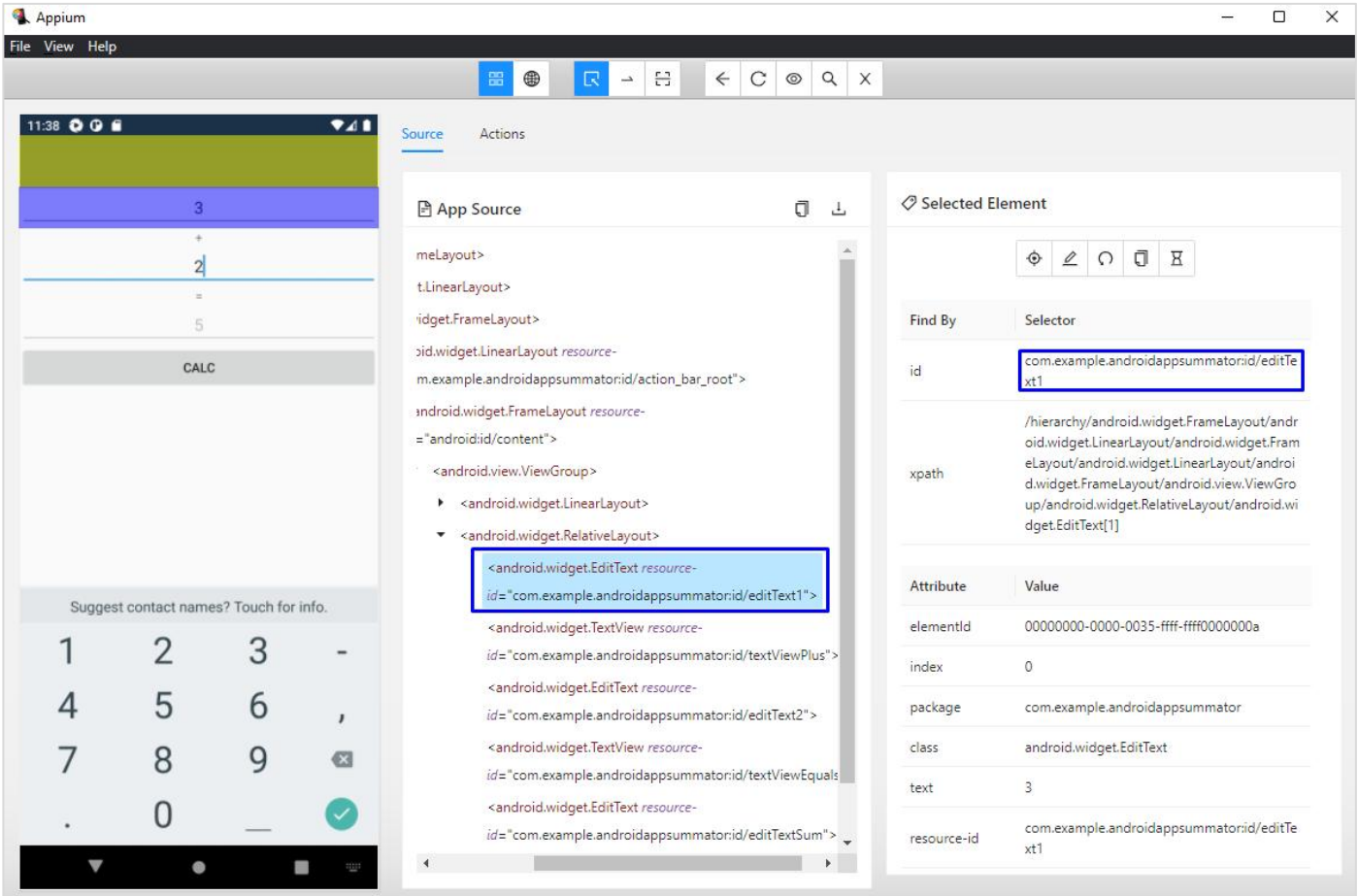
JSON Representation

```
{
  "platformName": "Android",
  "app": "D:\\com.example.androidappsummator.apk"
}
```

When **session is started** successfully, **`Appium`** should be connected to your **AVD** - whatever happens on the device happens in the server, too, and in reverse:



You can get the **`id`** of any element you need. **Click** on the element, e.g., on the **first number field**, and its **`id`** will be shown under "**`Selected Element`**":



You should do the same to **get ids of all the elements** you will need.

Let's write the test itself. Open **`Visual Studio`** and create a new **`C# NUnit Test Project`**. In it, install the **`Appium.WebDriver`** package from NuGet.

Start by creating **fields** for the **Appium Server Uri**, the path to the **Summator App** and the **Android driver** like this:

```
public class AndroidSummatorTests
{
    private const string AppiumServerUri = "http://[::1]:4723/wd/hub";
    private const string SummatorAppPath = @"D:\com.example.androidappsummator.apk";
    private AndroidDriver<AndroidElement> driver;
```

Then, write the **`Setup()`** method, which should **initialize** the Android driver with **options** for the **platform name** and the **app** (the same capabilities we added in the Appium Inspector to start the server):

```
    [OneTimeSetUp]
    0 references
    public void Setup()
    {
        AppiumOptions options = new AppiumOptions() { PlatformName = "Android" };
        options.AddAdditionalCapability("app", SummatorAppPath);
        this.driver = new AndroidDriver<AndroidElement>(
            new Uri(AppiumServerUri), options);
    }
```

Create the **`TearDown()`** method, which should **quit the driver**, as well:

```
    [OneTimeTearDown]
    0 references
    public void TearDown()
    {
        this.driver.Quit();
    }
}
```

Let's write the test method itself. Get the **first number field** by its **`id`**, which you got from the **`Appium Inspector`**. **Clear** the field, otherwise if it is not empty, new text will be just added after the old one. Next, **write** "**44**" in the field. The commands look like this:

```
[Test]
0 references
public void Test_Android_Summator_ValidData()
{
    // Arrange
    AndroidElement firstNumField = this.driver.
        FindElementById("com.example.androidappsummator:id/editText1");
    firstNumField.Clear();
    firstNumField.SendKeys("44");
```

Do the same with the **second number field**, where you should type "**55**". Press the **`[Calculate]`** button and assert that the text in the **result field** is "**99**". Get ids of elements with the **`Appium Inspector`**. The test should look like this:

```
[Test]
0 references
public void Test_Android_Summator_ValidData()
{
    // Arrange
    AndroidElement firstNumField = this.driver.
        FindElementById("com.example.androidappsummator:id/editText1");
    firstNumField.Clear();
    firstNumField.SendKeys("44");

    AndroidElement secondNumField = this.driver.
        FindElementById("com.example.androidappsummator:id/editText2");
    [illegible]
    [illegible]

    // Act
    AndroidElement calcButton = this.driver.
        [illegible]
    calcButton.Click();

    // Assert
    AndroidElement resultField = this.driver.
        [illegible]
    Assert.AreEqual("99", resultField.Text);
}
```

Write the test with **invalid data** and "**`error`**" as a result, as well. You may type "**`.`**" in the first number field and leave the second field **empty**. Then, "**`error`**" will be displayed. **Run all tests together** – they should be **successful**:

6. Appium Automated Tests for the "Contact Book" Android App

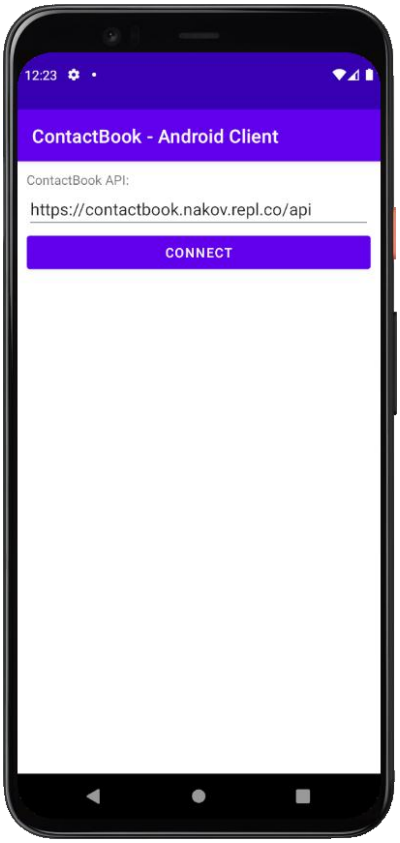
Implement **Appium UI automated tests** for the "**`Contact` Book**" Android app:

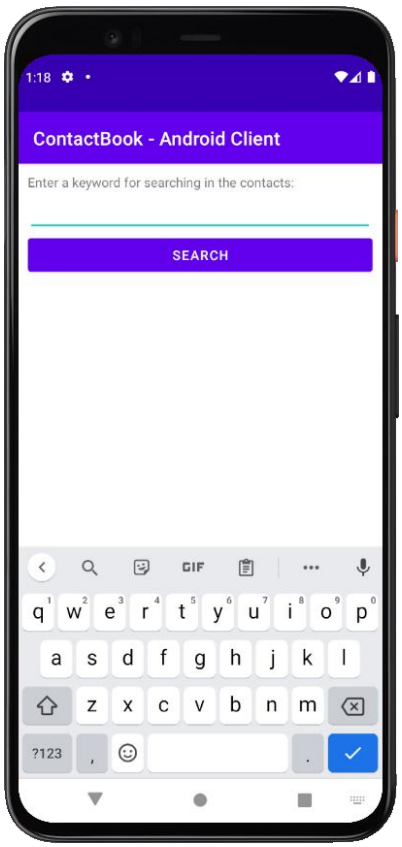
- APK's download release: https://github.com/nakov/ContactBook-AndroidClient/releases
- GitHub repo: https://github.com/nakov/ContactBook
- Web app live demo: https://contactbook.nakov.repl.co
- RESTful API live demo: https://contactbook.nakov.repl.co/api
- Play with code at: https://repl.it/@nakov/contactbook

The Automated Testing Scenario

3. Open the **`Contact` Book** app
4. Test with **existing** and **non-existing contact name**
   - With valid data and a **single result**: assert that result is **correct**
   - With valid data and **multiple results** (in our case multiple contacts): assert that result is **correct**
   - With invalid data: assert that "**`Contacts found: 0`**" is displayed

Get **ids** of elements, using the **`Appium Inspector`**, connected to an **`Android Virtual Device`**.

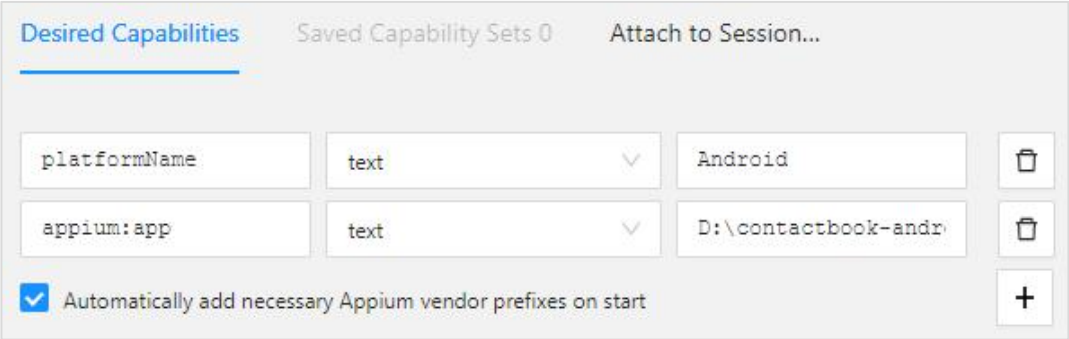


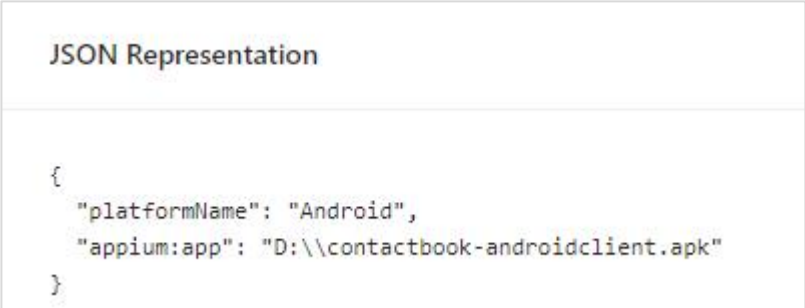


Use C#, NUnit, Appium, Android Driver, Android SDK (for the Android Device Emulator).

Hints and Guidelines

First, create an **Appium inspector session** with the **path** to the **.apk** file of the app (get it from the resources or **download** it from **GitHub**):





When **session is started** successfully, **Appium** should be connected to your **AVD** and the AVD should have the "**Contact Book**" app. You can get the **id** of any element you need:

Now, write the test itself. Start by creating **fields** for the **Appium Server Uri**, the path to the **"Contact Book" App**, the **API service URL** (https://contactbook.nakov.repl.co/api – you will need it), the **Android driver** and for the **WebDriverWait** instance. The **WebDriverWait** class is applied on certain element with defined expected condition and time (this is **explicit waiting** in Selenium):

```
public class ContactBookTests
{
    private const string AppiumServerUri = "http://[::1]:4723/wd/hub";
    private const string ContactBookAppPath = @"D:\contactbook-androidclient.apk";
    private const string ApiServiceUrl = "https://contactbook.nakov.repl.co/api";
    private AndroidDriver<AndroidElement> driver;
    private WebDriverWait wait;
```

Then, write the **Setup()** method, which should initialize the **AndroidDriver** with **URL** and **Appium options** and the **WebDriverWait** class. You also need to **set implicit wait** for the driver or tests may not execute correctly:

```
[SetUp]
0 references
public void Setup()
{
    AppiumOptions options = new AppiumOptions() { PlatformName = "Android" };
    options.AddAdditionalCapability("app", ContactBookAppPath);
    this.driver = new AndroidDriver<AndroidElement>(
        new Uri(AppiumServerUri), options);
    this.driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(5);
    this.wait = new WebDriverWait(this.driver, TimeSpan.FromSeconds(5));
}
```

Create the **`TearDown()`** method, which should **quit the driver**:

```
    [TearDown]
    0 references
    public void TearDown()
    {
        this.driver.Quit();
    }
}
```

Let's write the test method itself. First, get the **field** with the **API URL**, **clear** it, and **send** the new value, which we set as constant:

```
[Test]
0 references
public void Test_AndroidApp_SearchSteveContacts()
{
    // Connect to the RESTful service
    AndroidElement editTextApiUrl = this.driver.FindElementById(
        "contactbook.androidclient:id/editTextApiUrl");
    editTextApiUrl.Clear();
    editTextApiUrl.SendKeys(ApiServiceUrl);
```

Then, **click** on the **`[Connect]`** button to connect to the **RESTful service**:

```
    AndroidElement buttonConnect = this.driver.FindElementById(
        "contactbook.androidclient:id/buttonConnect");
    buttonConnect.Click();
```

On the next screen, send "**`Steve`**" as a **valid keyword** and press the **`[Search]`** button:

```
    // Search for "Steve"
    AndroidElement editTextKeyword = this.driver.FindElementById(
        "contactbook.androidclient:id/editTextKeyword");
    editTextKeyword.Clear();
    editTextKeyword.SendKeys("Steve");

    AndroidElement buttonSearch = this.driver.FindElementById(
        "contactbook.androidclient:id/buttonSearch");
    buttonSearch.Click();
```

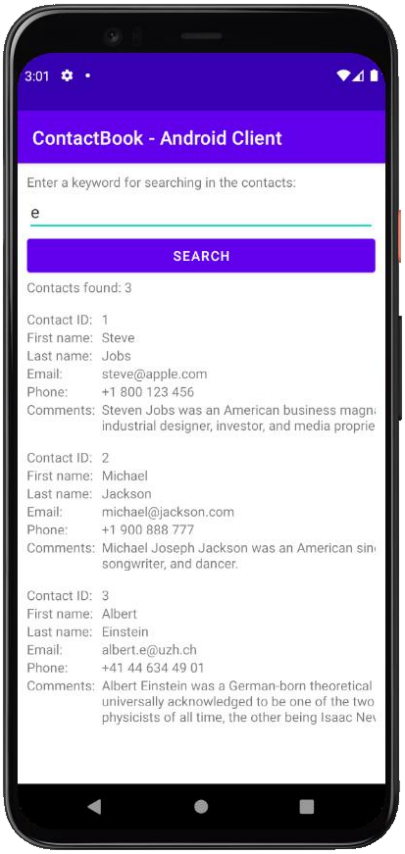
At the end, make the needed **assertions**. Assert that "**`Contacts found:`**" message is displayed, as well as the first and last name of "**`Steve Jobs`**" in the corresponding fields:

```
    // Assert that one contact is displayed
    AndroidElement textViewSearchResult = this.driver.FindElementById(
        "contactbook.androidclient:id/textViewSearchResult");
    this.wait.Until(t => textViewSearchResult.Text != "");
    string text = textViewSearchResult.Text;
    Assert.That(text.Contains("Contacts found: 1"));

    // Assert that the contact is "Steve Jobs"
    AndroidElement textViewFirstName = this.driver.FindElementById(
        "contactbook.androidclient:id/textViewFirstName");
    Assert.AreEqual("Steve", textViewFirstName.Text);

    AndroidElement textViewLastName = this.driver.FindElementById(
        "contactbook.androidclient:id/textViewLastName");
    Assert.AreEqual("Jobs", textViewLastName.Text);
}
```

After that, write a test where **multiple contacts are found**. For example, if you search for "**e**" as keyword, all **3 contacts** in the app will be displayed ("**Steve Jobs**", "**Michael Jackson**" and "**Albert Einstein**"):



Assert that the "**`Contacts found: 3`**" message is displayed

Assert that the **names of the contacts in the list are correct**. To get all contacts, use the **`FindElementsById()`** method.

At the end, write a test, where there are **no contacts found** for a given **keyword**. For example, there are **no contacts** for "**`Pesho`**". Assert that "**`Contacts found: 0`**" message is displayed.

Finally, **run all tests**. They should be **successful**:

7. Appium Automated Tests for the "Vivino" Android App

Implement **Appium UI automated tests** for the **"Vivino" mobile app**:

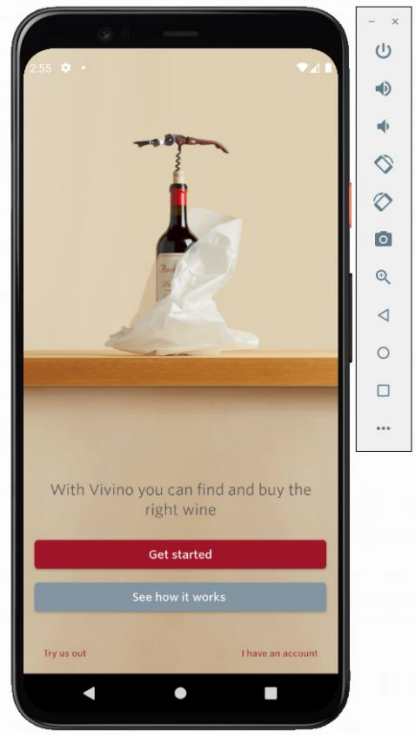
"**Vivino**" is a popular wine catalog and community app for wine lovers. We shall use a specific **app version** to ensure the tests will run exactly as in this specific version:

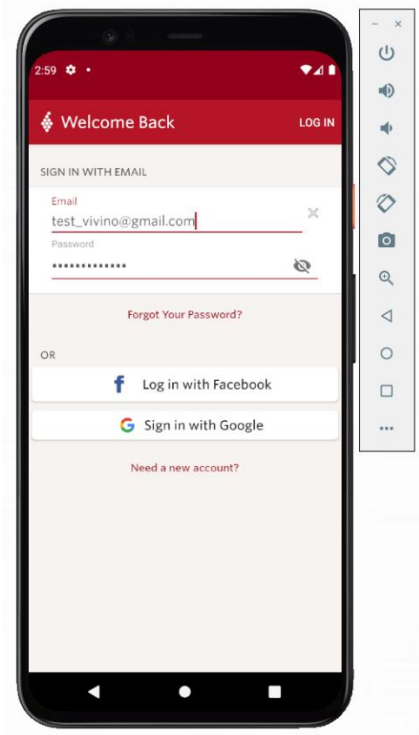
- **"Vivino" app for Android, version 8.18.11**, which can be downloaded as APK package from: https://www.apkmirror.com/apk/vivino/vivino-wine-scanner/vivino-wine-scanner-8-18-11-release/vivino-buy-the-right-wine-8-18-11-3-android-apk-download/download

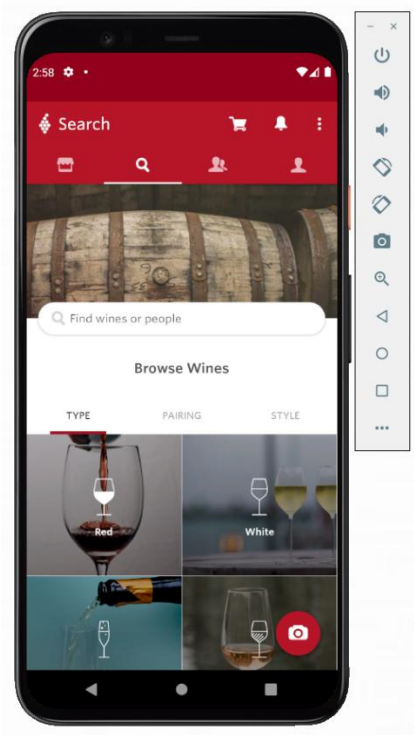
The Automated Testing Scenario

1. **Run** the Vivino app in the Android emulator.
2. **Login** with pre-registered **email** + **password**.
3. Click on the **[Search]** tab.

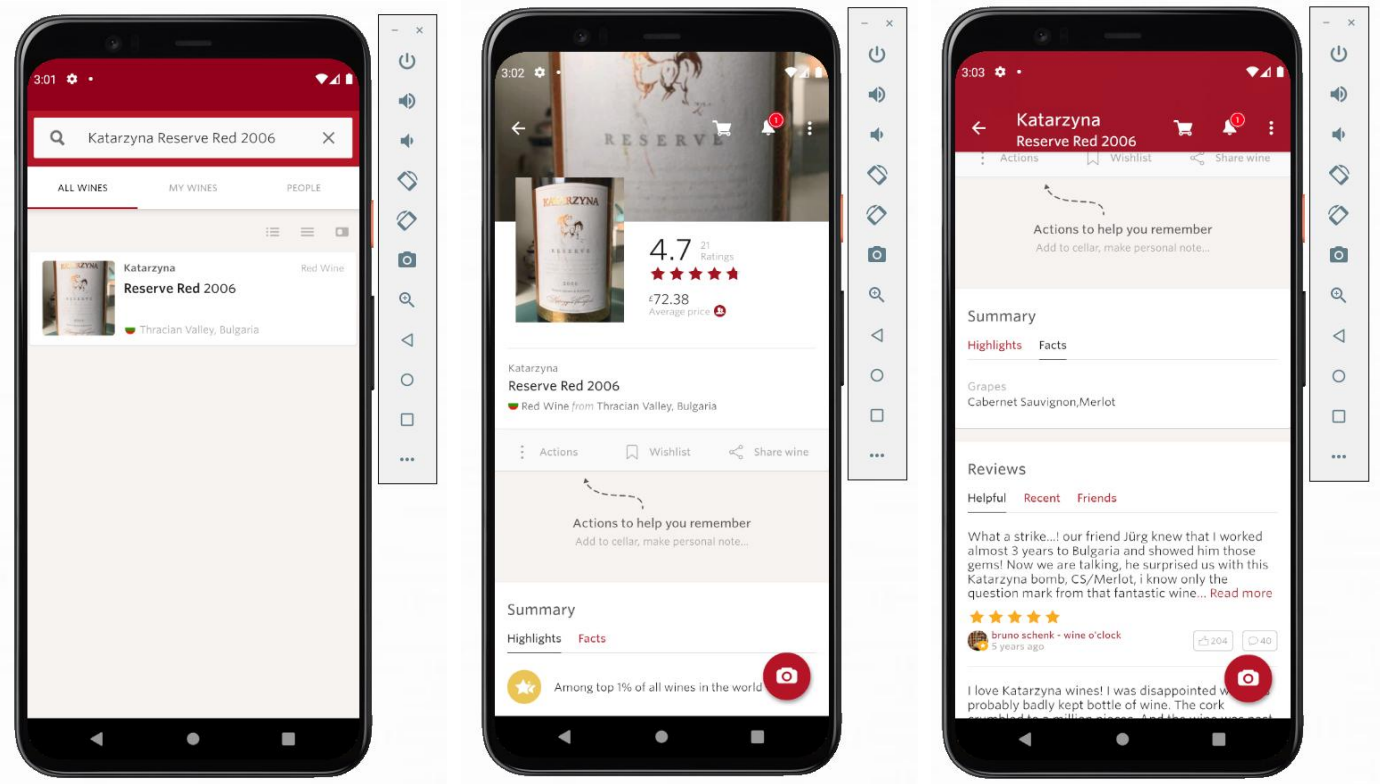
The **screenshots** bellow visualizes the above steps.







4. Enter the following **keywords** for searching: "***Katarzyna Reserve Red 2006***".
5. Click on the first search result.



6. **Assert** that the wine **name** is correct: "***Reserve Red 2006***".
7. **Assert** the wine **rating** is a number in the range [1.00 ... 5.00].
8. **Assert** the wine **highlights** contain "***Among top 1% of all wines in the world***".
9. **Assert** the wine **facts** hold "***Grapes: Cabernet Sauvignon, Merlot***".

Hints and Guidelines

We shall use C#, Visual Studio, NUnit, Appium WebDriver, Appium Desktop, Android Driver, and Android Studio + Android SDK (for the Android device emulator).

Create a C# NUnit project and install "**`Appium.WebDriver`**" library from NuGet.

Run **Android Studio**, then run the Android Virtual Devices Manager (the **AVD Manager**), then **run an Android emulator**, then run your **Android virtual device**. You can now close the Android Studio and the AVD Manager.

Download the **APK installation** package for the app **"Vivino" for Android, version 8.18.11**. You need to use exactly this version. Otherwise, some UI elements could be different, and your tests and element locators may fail.

Define some constants in your unit test class:

```
public class VivinoTests
{
    private const string AppiumServerUri = "http://[::1]:4723/wd/hub";
    private const string VivinoAppPath = @"D:\vivino_8.18.11-8181203.apk";
    private const string VivinoAppPackage = "vivino.web.app";
    private const string VivinoAppStartupActivity = "com.sphinx_solution.activities.SplashActivity";
    private const string VivinoTestAccountEmail = "test_vivino@gmail.com";
    private const string VivinoTestAccountPassword = "p@ss987654321";
    private AndroidDriver<AndroidElement> driver;
```

Note that "Vivino" app cannot be started in Appium unless you provide the exact name of the startup activity class "**com.sphinx_solution.activities.SplashActivity**".

Now, start writing the test automation code.

First, setup the Appium driver at **startup** to test the Vivino app in Android:

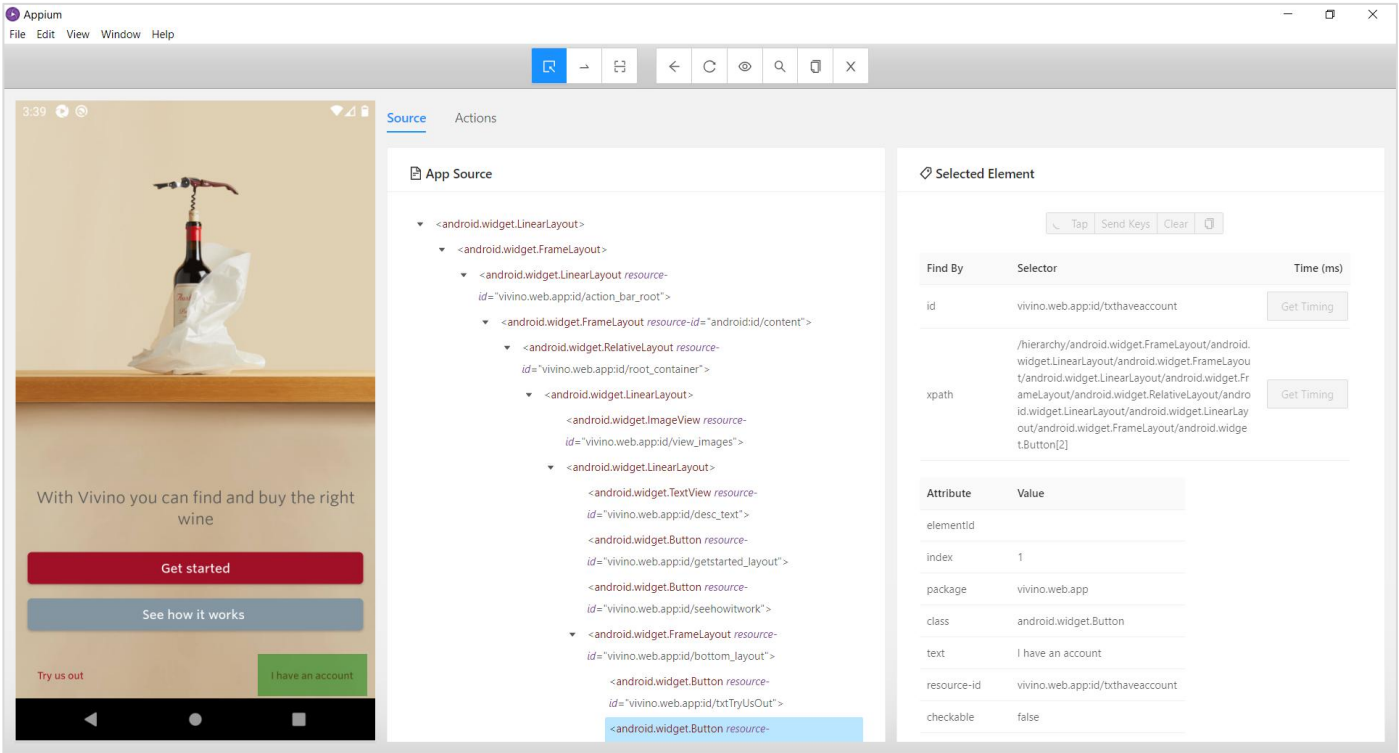
```
[OneTimeSetUp]
0 references
public void Setup()
{
    AppiumOptions appiumOptions = new AppiumOptions() { PlatformName = "Android" };
    appiumOptions.AddAdditionalCapability("app", VivinoAppPath);
    appiumOptions.AddAdditionalCapability("appPackage", VivinoAppPackage);
    appiumOptions.AddAdditionalCapability("appActivity", VivinoAppStartupActivity);
    this.driver = new AndroidDriver<AndroidElement>(
        new Uri(AppiumServerUri), appiumOptions);
    this.driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(120);
}
```

We shall use a **long implicit wait** because the app sometimes loads data from Internet, and this can take several seconds or even more.

Also write a standard **teardown** method to release the Appium session:

```
    [OneTimeTearDown]
    0 references
    public void TearDown()
    {
        this.driver.Quit();
    }
}
```

Use **Appium Desktop** to identify the **UI controls hierarchy** in the Vivino for Android app, to find their IDs and XPath expressions and to interact with them:



Implement the "**Login**" functionality:

```
[Test]
0 references
public void Test_WineSearch()
{
    // Login in the Vivino app
    AndroidElement linkLogin =
        this.driver.FindElementById("vivino.web.app:id/txthaveaccount");
    linkLogin.Click();

    AndroidElement textBoxLoginEmail =
        this.driver.FindElementById("vivino.web.app:id/edtEmail");
    textBoxLoginEmail.SendKeys(VivinoTestAccountEmail);

    AndroidElement textBoxLoginPassword =
        this.driver.FindElementById("vivino.web.app:id/edtPassword");
    textBoxLoginPassword.SendKeys(VivinoTestAccountPassword);

    AndroidElement buttonLogin =
        this.driver.FindElementById("vivino.web.app:id/action_signin");
    buttonLogin.Click();
```

Now implement "**Search**" for the specified keywords:

```
// Search for "Katarzyna Reserve Red 2006"
AndroidElement tabExplorer =
    this.driver.FindElementById("vivino.web.app:id/wine_explorer_tab");
tabExplorer.Click();

AndroidElement buttonSearch =
    this.driver.FindElementById("vivino.web.app:id/search_vivino");
buttonSearch.Click();

AndroidElement textBoxSearch =
    this.driver.FindElementById("vivino.web.app:id/editText_input");
textBoxSearch.SendKeys("Katarzyna Reserve Red 2006");
```

As next step, interact with the **search results** and assert they are as expected:

```
// Click on the first search result and assert it holds correct data
AndroidElement listSearchResults =
    this.driver.FindElementById("vivino.web.app:id/listviewWineListActivity");
AppiumWebElement firstResult =
    listSearchResults.FindElementByClassName("android.widget.FrameLayout");
firstResult.Click();

AndroidElement elementWineName =
    this.driver.FindElementById("vivino.web.app:id/wine_name");
Assert.AreEqual("Reserve Red 2006", elementWineName.Text);

AndroidElement elementRating =
    this.driver.FindElementById("vivino.web.app:id/rating");
double rating = double.Parse(elementRating.Text);
Assert.IsTrue(rating >= 1.00 && rating <= 5.00);
```

Now, we want to check the **"Summary" box** and the wine **highlights** and **facts**:

```
AndroidElement tabsSummary =
    this.driver.FindElementById("vivino.web.app:id/tabs");
AppiumWebElement tabHighlights =
    tabsSummary.FindElementByXPath("//android.widget.TextView[1]");
tabHighlights.Click();
```

We next get the text from the text view "**`vivino.web.app:id/highlight_description`**":

```
// Grab the text in the "Highlights" tab
AppiumWebElement highlightsDescription =
    this.driver.FindElementById("vivino.web.app:id/highlight_description");
Assert.AreEqual("Among top 1% of all wines in the world", highlightsDescription.Text);
```

The last step in our unit test is to check the **"Facts" tab** and assert it holds what we expect:

```
    // Check the text in the "Facts" tab
    AppiumWebElement tabFacts =
        tabsSummary.FindElementByXPath("//android.widget.TextView[2]");
    tabFacts.Click();
    AndroidElement factTitle =
        this.driver.FindElementById("vivino.web.app:id/wine_fact_title");
    Assert.AreEqual("Grapes", factTitle.Text);
    AndroidElement factText =
        this.driver.FindElementById("vivino.web.app:id/wine_fact_text");
    Assert.AreEqual("Cabernet Sauvignon,Merlot", factText.Text);
}
```

Finally, **run the unit test** to ensure it works correctly: