

---

# Windstream iOS App Automation using BDD Framework

17<sup>th</sup> January 2022,

## OVERVIEW

Behavior-driven development or BDD is a more accessible and effective way for teams, new to agile software delivery to test business behavior, rather than a computer function.

BDD approach uses basic concepts of “given, when, then” to describe various user scenarios. It offers an improvement in communication methods between product owners, developers, testers, and users with or without a testing tool. BDD is commonly used with automation using Gherkin and combined with unit testing.

## Advantages of BDD

Automation engineers no longer define ‘test’ but are defining ‘behavior.’

- Communication of business requirements between developers, testers, and product owners improves.
- The learning curve is much shorter when explaining BDD as it uses simple language.
- Being non-technical, it can reach a wider audience.
- The behavioral approach defines acceptance criteria before development

BDD helps develop, test, and think about the code from the business owner’s point of view.

## System Requirements

1. Java 1.8 or higher
2. Node.js 12 or higher
3. Homebrew
4. Git/Azure DevOps (ADO)

- 
5. macOS 10.15.7 or higher
  6. Xcode 12 or higher with command-line build tool
  7. Appium 1.6.0 or higher since XCUITest is available from this version
  8. Maven
  9. Eclipse v2021-06 (4.20.0)
  10. Major dependencies used:

- Cucumber

```
<dependency>
  <groupId>io.cucumber</groupId>
  <artifactId>cucumber-java</artifactId>
  <version>7.0.0</version>
</dependency>
<dependency>
  <groupId>io.cucumber</groupId>
  <artifactId>cucumber-testng</artifactId>
  <version>7.0.0</version>
</dependency>
```
- Appium:

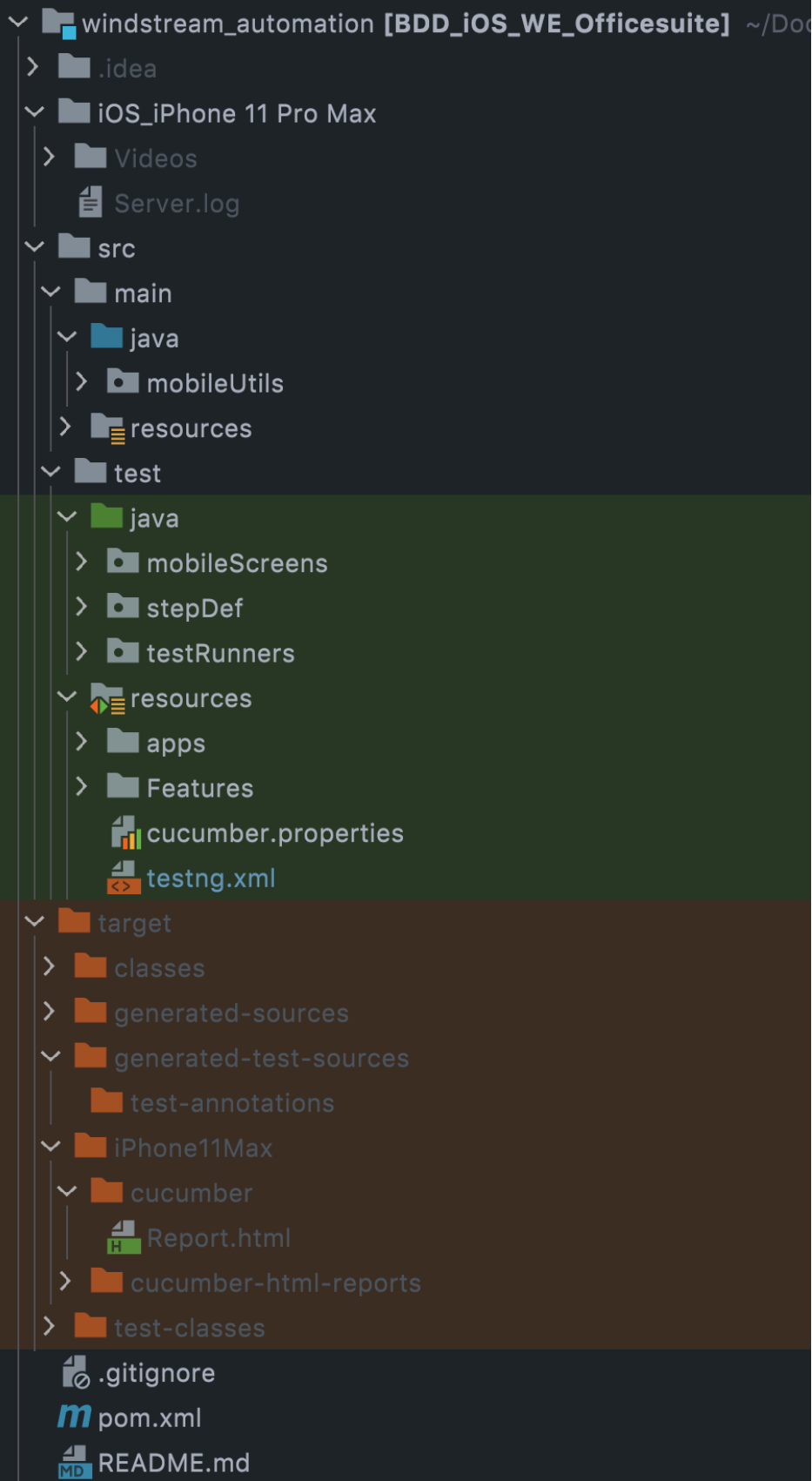
```
<dependency>
  <groupId>io.cucumber</groupId>
  <artifactId>cucumber-junit</artifactId>
  <version>7.0.0</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>io.appium</groupId>
  <artifactId>java-client</artifactId>
  <version>7.6.0</version>
</dependency>
```

## Framework Components

The framework consists of the following major components:

1. Page Objects
2. Custom Debugging Tools
3. Report Generation



---

## 1. Page Objects

This component provides the application screens in an easily accessible format. The page object is created based on the platform value passed in the Gradle environment. All action methods are accessible only through the standard interface.

## 2. Custom Debugging Tools

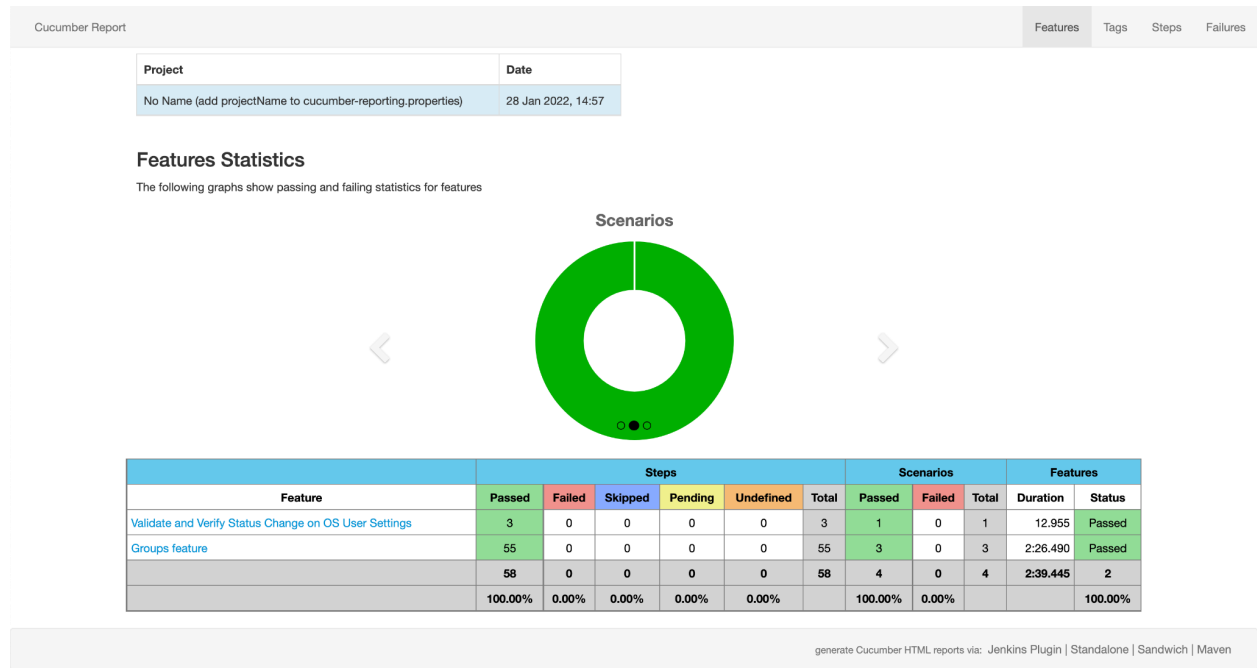
This component provides debugging tools to investigate test failures. It consists of the following sub-components:

- Screenshots
- Video capture of each test
- File logging

## 3. Report Generation

This component can generate test reports, customize using CSS, and provide the capability to add details such as:

- Tests count
- Test passed
- Test failed
- Passing percentage
- Platform information
- Scenario details
- Total time taken for execution
- And many more.



## Appium:

Appium is an open-source automation testing framework for apps supported by platforms like iOS, Android, macOS, and Windows. Similar to Selenium, it enables the tester to write test scripts in different programming languages such as JavaScript, Java, Ruby, Python, PHP, and C#, which can be used for both Android and iOS platforms.

## How does Appium iOS work?

Appium works on the principle of RESTful services by sending JSON files, which automatically interact with an iOS application using UI elements like text labels, buttons, etc. via Apple's UIAutomation API for automated app testing.

The bootstrap.js file works as a TCP server sending test commands to perform actions on the iOS device using Apple's UIAutomation API framework.

---

## Why use Appium for testing iOS devices?

Appium is an ideal choice for automated testing on iOS devices because of the following features:

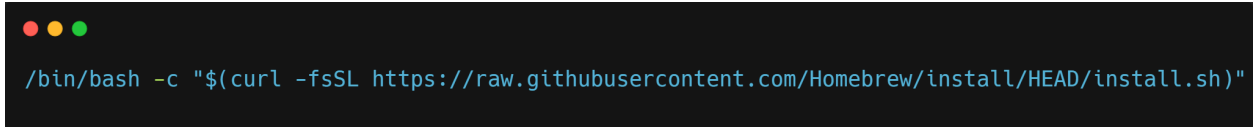
- It is an open-source framework, which means one can save on hefty licensing costs of tools and frameworks.
- With an in-built XCUITest driver, Appium is capable of producing information logs with a detailed reporting structure. This makes analysis of the test results better and improves debugging efforts.
- It offers reusability of the same code that is written for iOS devices for Android devices, saving time and effort for the hybrid apps.
- It allows users to test the application on various versions of iOS devices. This ensures cross platform compatibility for apps.
- It offers real-time monitoring of the tests on real devices which provides greater reliability.

### Framework setup prerequisites:

Before exploring how to run Appium tests on iOS devices, here are the prerequisites that have to be fulfilled in order to conduct those tests:

- A Mac computer with macOS 10.11 or 10.12, and a device with iOS 9.3 or higher
- Clone the automation WE Connect [project repository](#)
- [Install Homebrew](#) (for managing missing packages)

**Cmd:-** `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`

A screenshot of a macOS terminal window with a dark background and light green text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The command being entered is `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

- [Install Carthage](#) (for managing dependencies)

**Cmd:-** `brew install carthage`



```
brew install carthage
```

- [Install Node](#) & NPM

**Cmd:-** brew install node



```
brew install node
```

- [Install Maven](#)

**Cmd:-** brew install maven



```
brew install maven
```

- [Install Appium](#)

**Cmd:-** npm install -g appium



```
npm install -g appium
```

- Install ios-deploy

**Cmd:-** `npm install -g ios-deploy`



```
npm install -g ios-deploy
```

- [Install XCode](#) 12 or higher
- [Install Java](#) and set up the [environment variables](#)
- [Install Eclipse IDE for Java](#)
- [Install XCUITest Driver](#)
- [Install TestNG](#)

**Framework setup for running the test suite :**

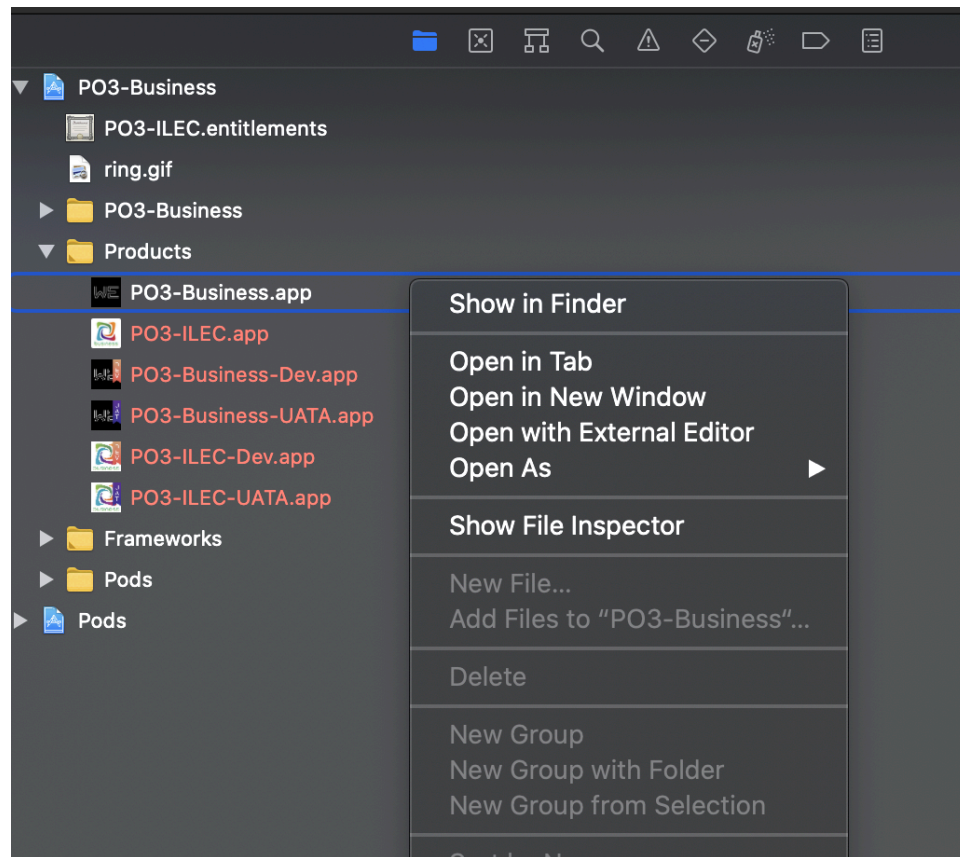
**Build the latest WE Connect iOS app:**

- Clone the [WE Connect App Repository](#) (ADO repository)
- Navigate to cloned directory
- Run pod install
- Open **PO3-Business.xcworkspace** file in Xcode
- Build the application

**Run the test suite**



- After build is complete navigate to the **PO3-Business.app** in Finder



- Copy the location of **PO3-Business.app**
- Navigate to automation project **BDD\_iOS\_WE\_Officesuite**
- Inside **src/main/resources/config.properties** file, paste the full path of **PO3-Business.app** for the **iOSAppLocation** key.
- Run the test suite using **testng.xml**