CoachEra Maintainer Manual

Rayan Wali, Yann Hicke, Jayesh Paunikar, Elena Stoeva, Ashley Yu,
Hehong Li, Angela Lau, Yixin Zhang

I. Requirements Analysis and Specification

The key high-level requirement that the client had for us is to extend or modify defined functionalities of the MobileCoach chatbot system. The client provided us with the following specific functionalities that they were looking for in the final release of the system:

- A dynamic dashboard that includes a star system, a progress indicator that collectively reflect the progress of a user, a weekly task list, and an indicator of past weekly accomplishments.
- A Frequent Questions and Answers page that is focused on guiding users on how the application can help them achieve their goal.
- Conversational flows that define the interaction between the chatbot and the user.
- Push notifications that could be sent from the mobile application to the user's device.

The requirements listed above are the key requirements. There were certainly many sub-requirements within each point above that the client defined and that we resolved. However, there are also requirements that the client tentatively stated but had to let go of due to other priorities. Our expectation is that future maintainers like you can take these functionalities we implemented following the client's specifications and extend them to turn the application into one that is for all types of learners and for different courses, which is something we were not able to implement, but is something we can expect developers to implement on top of our complete package, containing the complete code and documentation, in the future.

II. Test Facilities

A. Test Plan

We test the components that we implemented in the MobileCoachApp repository using the following approaches:

a. Unit tests (including snapshot tests and functionality unit tests)
b. End-to-end tests
c. Continuous Integration through Jenkins

B. Unit Tests

We implement both functionality tests and snapshot tests using Jest - a JavaScript test runner.

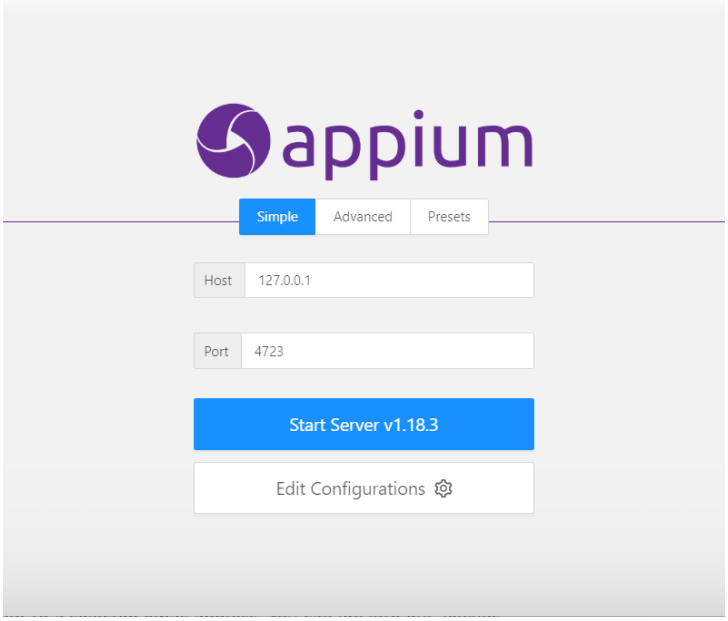
Functionality unit tests are assertion tests for the methods and classes that we implemented and snapshot tests ensure that UI does not change unexpectedly. To run

all tests inside repository, run the command 'yarn test' assuming that you have installed all dependencies with 'yarn install' Note that when you change any component in the application, you need to first run 'yarn test -u' which will update the reference snapshot for the snapshot tests.

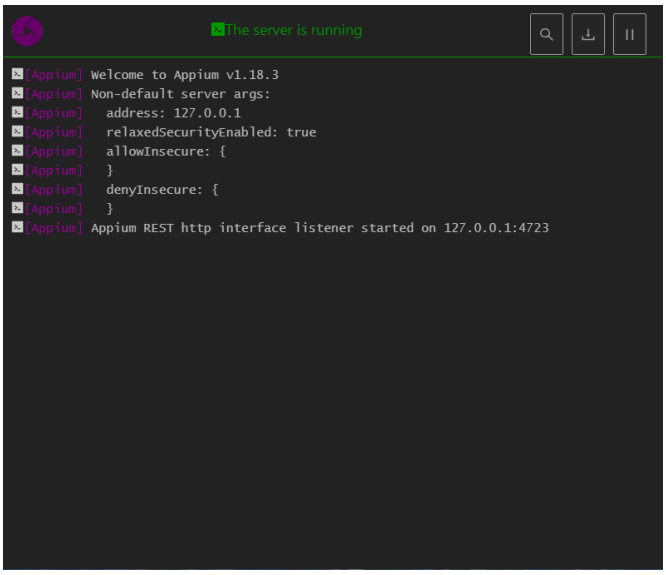
All tests for dashboard in '_test_' folder under 'containers' folder. The original mobilecoach project does not have any test, and all unit tests added by us are in that folder.

C. End-to-End Tests

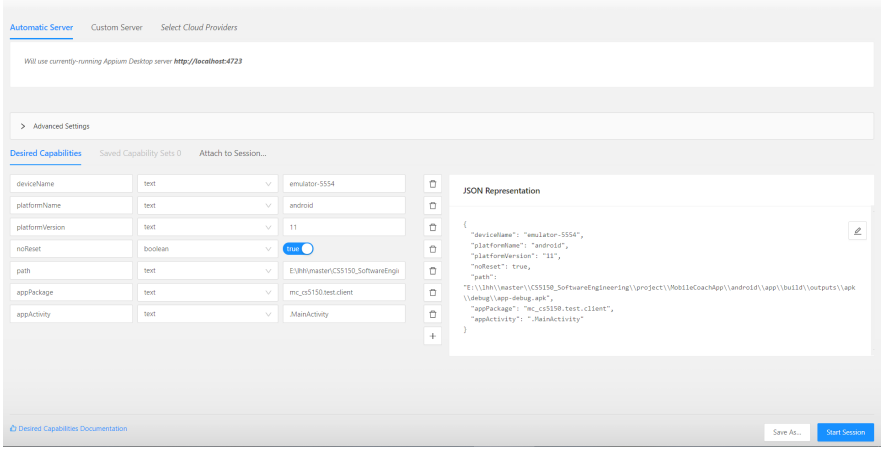
Appium and Selenium are two automated testing frameworks we use to test our application, we integrated these two with the android emulator. First, we start a local server at the url:http://127.0.0.1 using Appium and the configurations are set to the path of JAVA and Android in your own laptop.



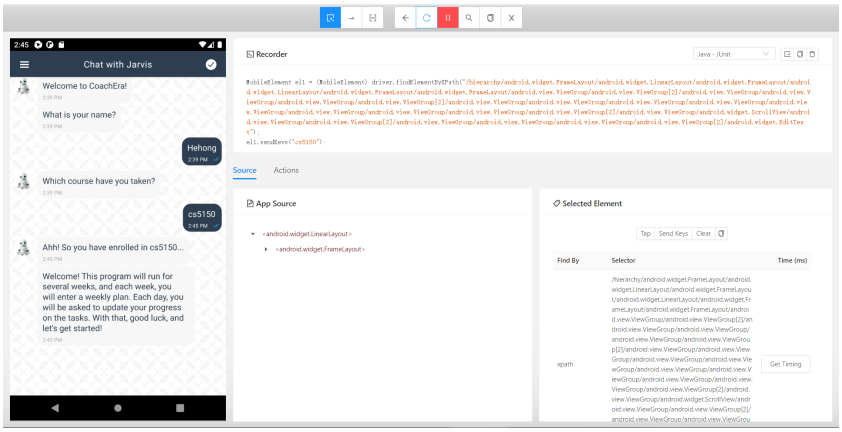
And you could also set the host address to 0.0.0.0, that will also start a local Appium Server, but you need to use different addresses which you use here for the future connection of your java program and Appium.



Then we connect the Appium server and our Android simulator by entering the parameter of the application. For example, deviceName, platformName, platformVersion, appPackage, appActivity and path (.apk). To get the values of these, you can run '*adb devices*' for the deviceName and d*umpsys window windows | grep -E 'mCurrentFocus'* under the command '*adb shell*' on your cmd if you are using a Windows system.



If it is connected successfully, you can get the following page showing the same screen with our emulator. Then we use the internal function shown at the top of the page to record the actions we would like to test based on the scenarios we planned to perform before. We added some clicking and texts which we want to chat with the e-coach.



And when we get all the functions with the element number and the actions on them and these are all added into the java program. And then you need to download 4 basic libraries containing the classes used for the program, which are *selenium-server-standalone-3.141.59*, *java-client-7.0.0*, *selenium-java-3.141.59*, *commons-lang3-3.12.0*. You may be able to do the latest version, but it may need some configuration changes, for example I was having an error for 'MobileElement' is changed to 'WebElement' in the latest java-client packages.

Besides, in your Jjava program, you also need to set these parameters to make the connection between your program and the emulator through Appium. These are the same values with the ones you use to connect with Appium.

```
DesiredCapabilities dc= new DesiredCapabilities();
dc.setCapability(MobileCapabilityType.DEVICE_NAME, "emulator-5554");
dc.setCapability("platformName","android");
dc.setCapability("appPackage","mc_cs5150.test.client");
dc.setCapability("appActivity",".MainActivity");
dc.setCapability("app","E:\\lhh\\master\\CS5150_SoftwareEngineering\\project\\AwesomeProject\\android\\app\\build\\outputs\
//dc.setCapability("ms:waitForAppLaunch",25);
```

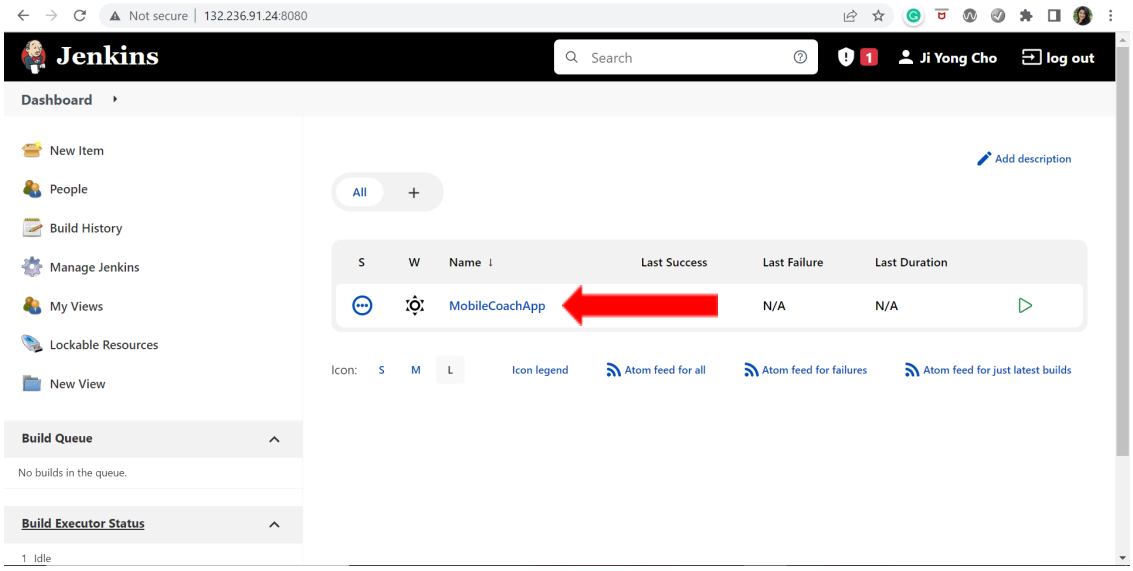
Finally, after following all of these steps, when you run your java program, you will see the application is doing all of the pre-set actions automatically. There is one thing to notice is that if your app reacts slowly or needs some more time to launch at the starting part, please add some sleep time for the thread.

III. Continuous Integration

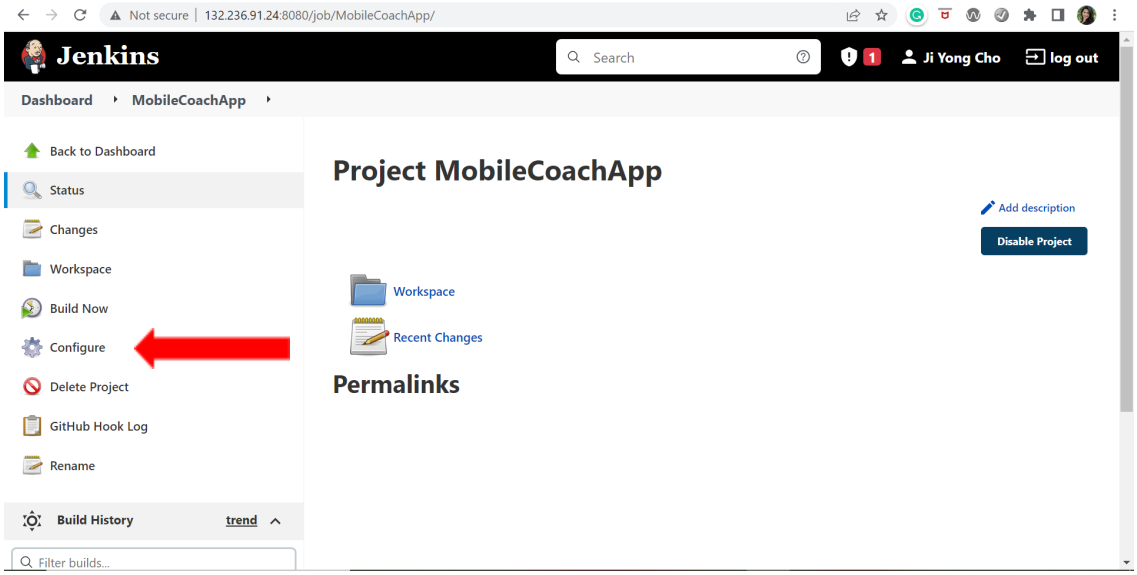
We maintain continuous integration through Jenkins which is installed and runs on our server. Every push and pull request to the MobileCoachApp repository triggers a new build in our Jenkins project which runs all unit tests in the repository. If a build fails, an email notification is sent to a selected email address. You can log into our Jenkins environment through this URL: http://132.236.91.14:8081/

The Jenkins project that creates builds that run our unit tests is called MobileCoachApp.

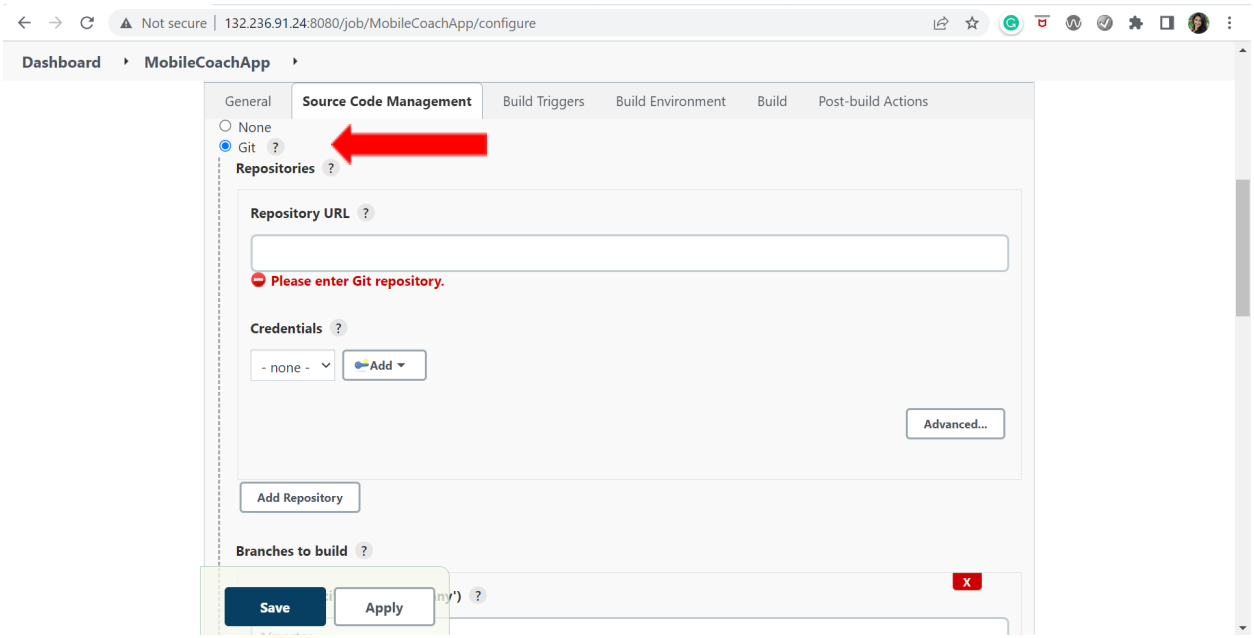
Getting Started With Jenkins:
Jenkins is installed on the production server and it is accessible at http://132.236.91.24:8080/
Please contact Ji Yong Cho for login credentials.
Once you log in, you will see that we have created a Jenkins Project called MobileCoachApp.



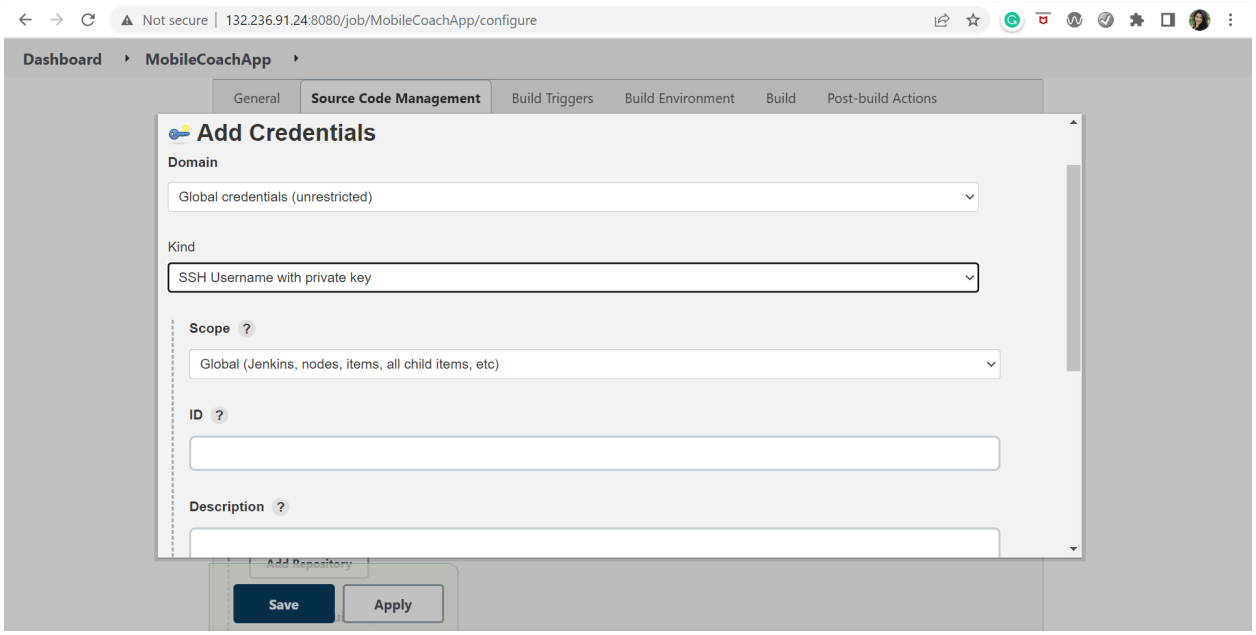
Click on the name of this project. Then select the "Configure" button in the menu to the left.



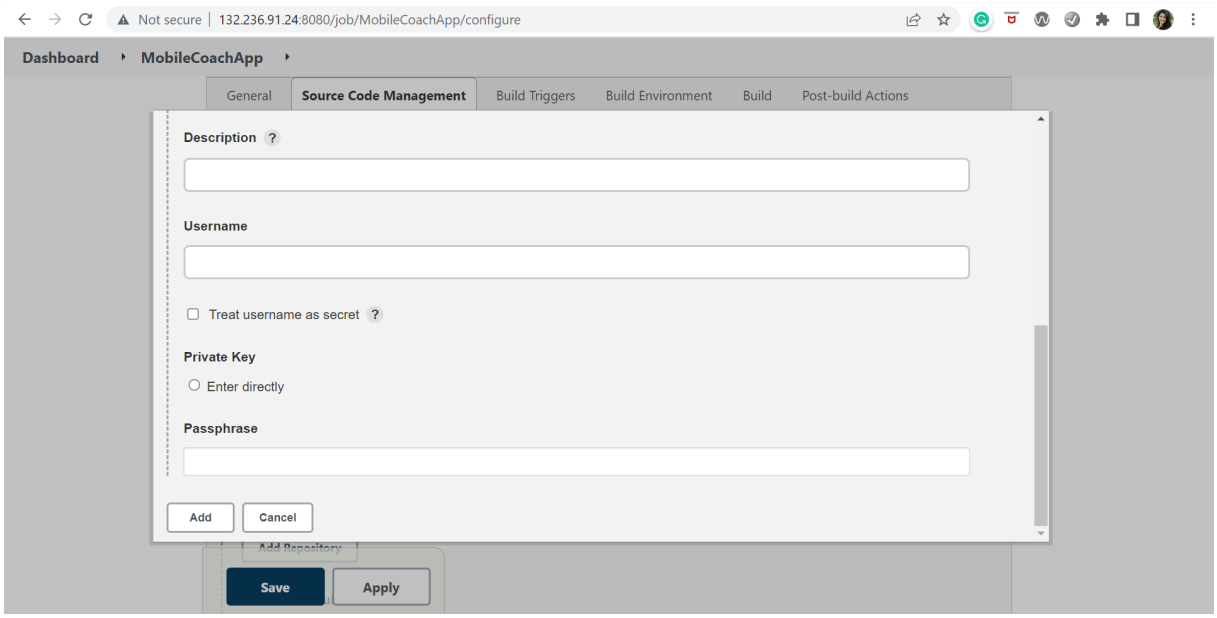
Under the “Source Code Management” tab, select the “Git” option.



If the repository that you are working on is public, just enter its http or ssh address. If it is private. You need to provide Git credentials. Click on the “Add” button under the “Credentials” tab.

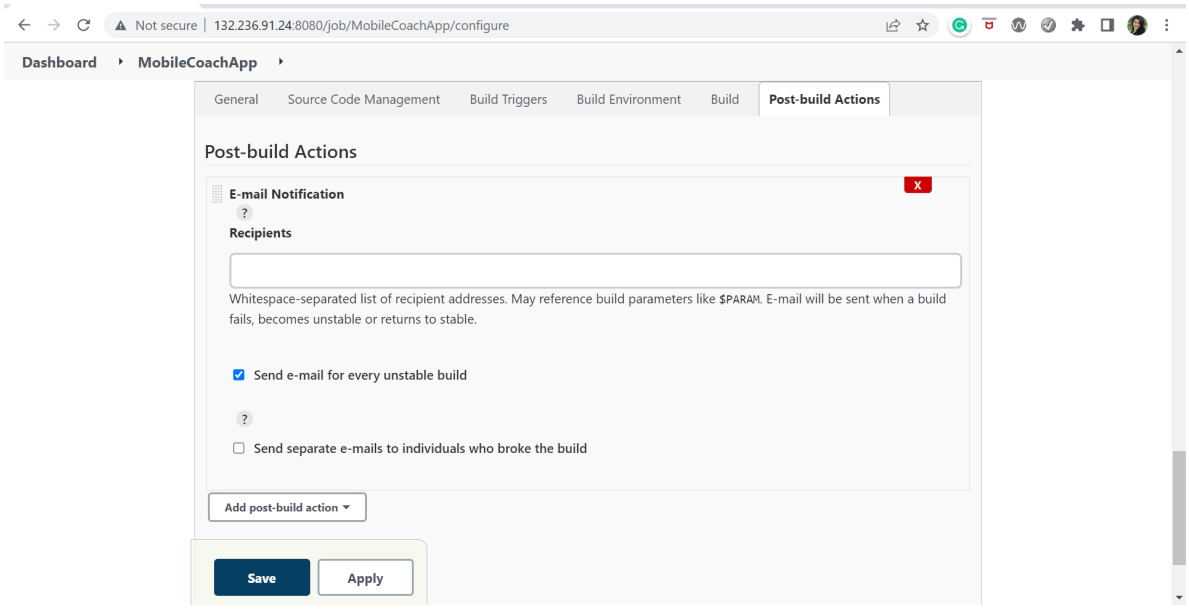


We recommend that you select SSH Username with private key under “Kind”.



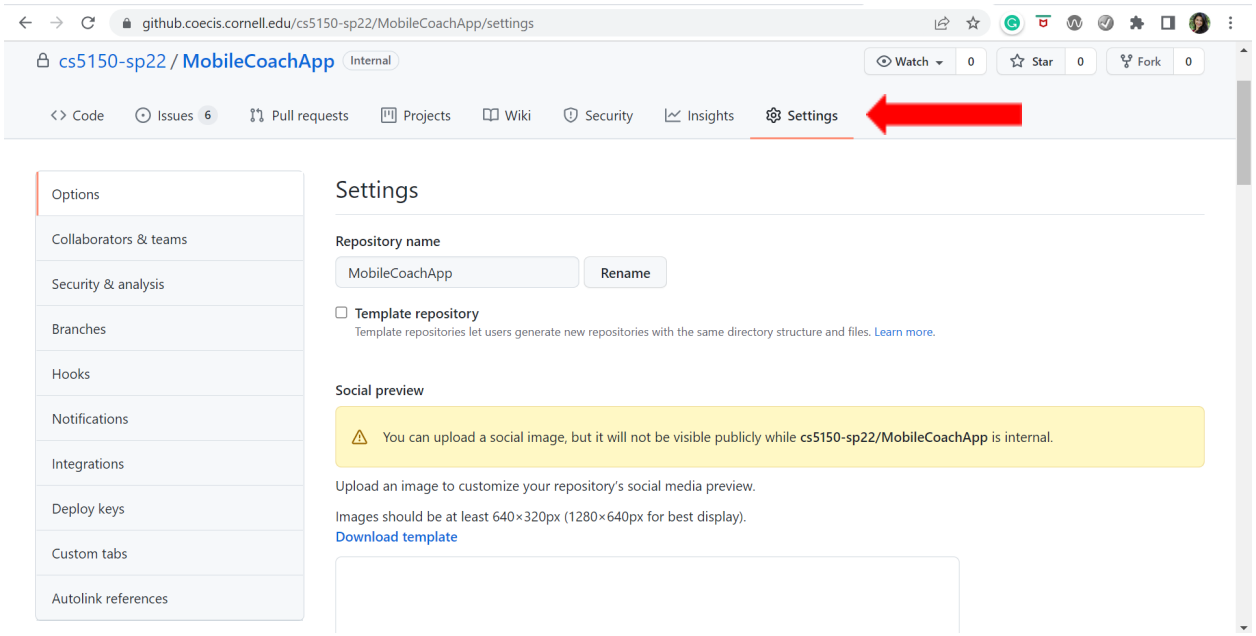
Write your GitHub username in the "Username" field. Select "Enter directly" under "Private Key", click on the "Add" button, and enter your **private** SSH key. Then click on the "Add" button to add your SSH credentials. Note: if you use SSH credentials, you need to add the **SSH URL** of your repository, not the HTTP one.

(Optional) Under "Post-build Actions" you can choose "Email Notification" and provide an email address to which an email notification will be sent every time when a build fails (this is the approach that we used in our CS5150 team). If you decide to have this functionality, you also need to perform some additional setup in order to enable Jenkins to send emails to your email address. Please follow the instructions on this page.

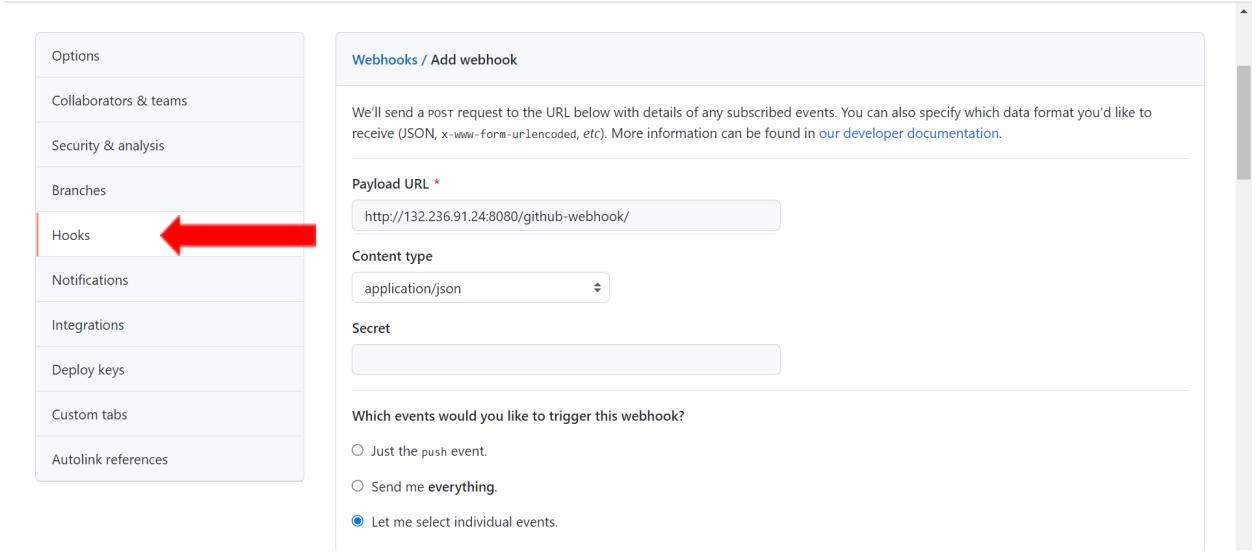


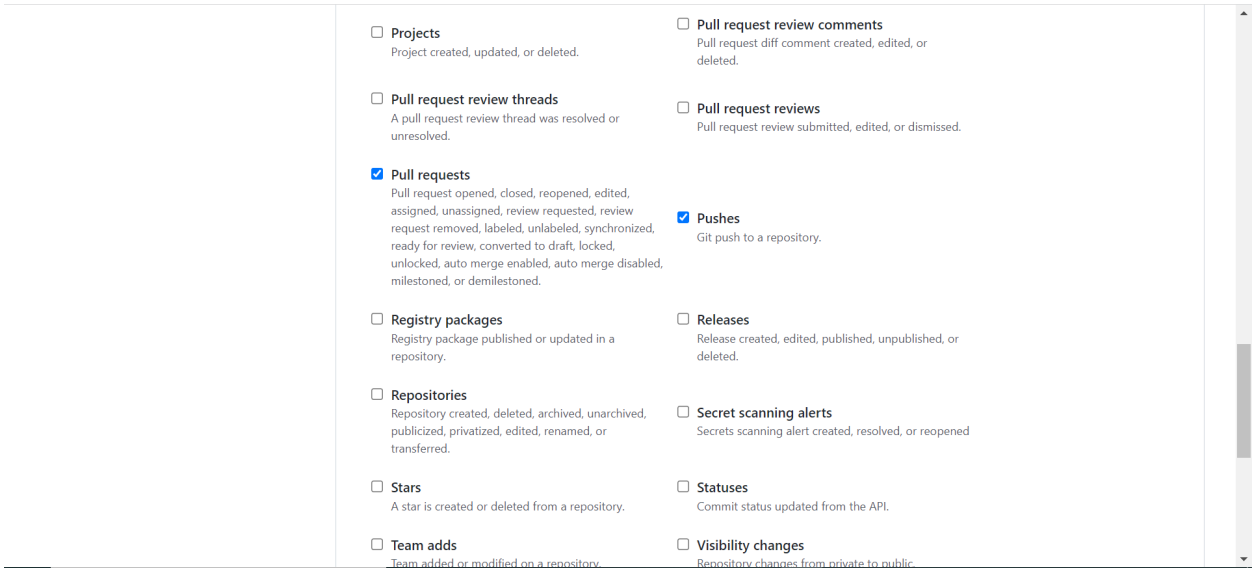
Finally, click on the "Save" button. Your Jenkins project is now configured.

Now, head over to your repository and click on the "Settings" tab.



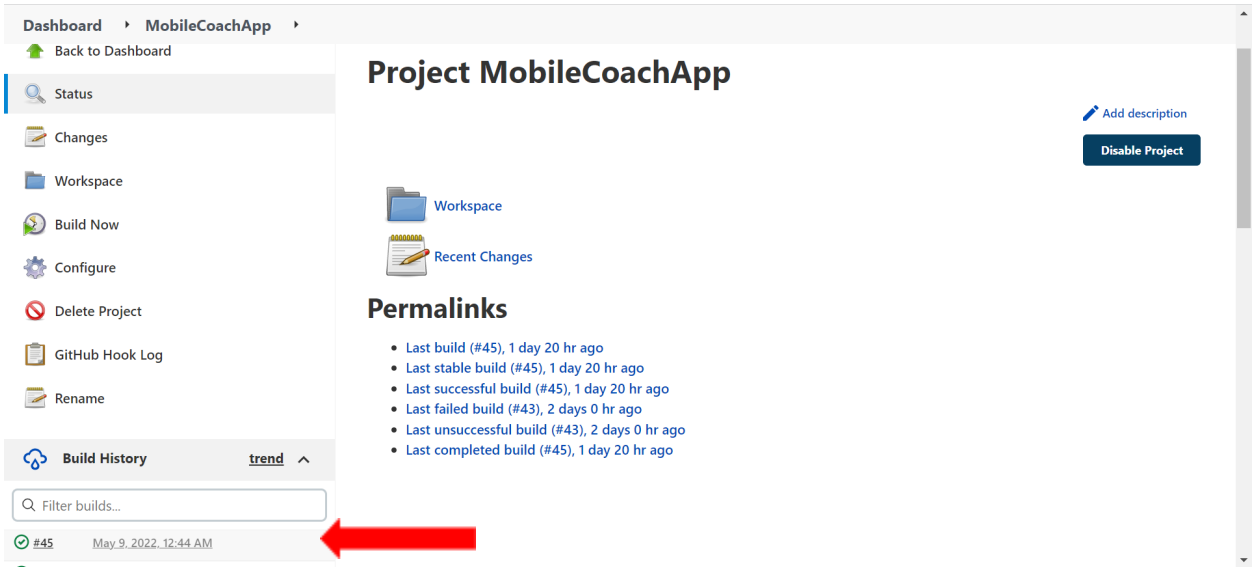
Click on the "Hooks" tab in the menu to the left. Then click on the "Add webhook" button. Under "Payload URL" type in http://132.236.91.24:8080/github-webhook/. Choose "application\json" under "Content type". Under the question "Which events you like to trigger this webhook?", select "Let me select individual events." From the events, select "Pushes" and "Pull requests".

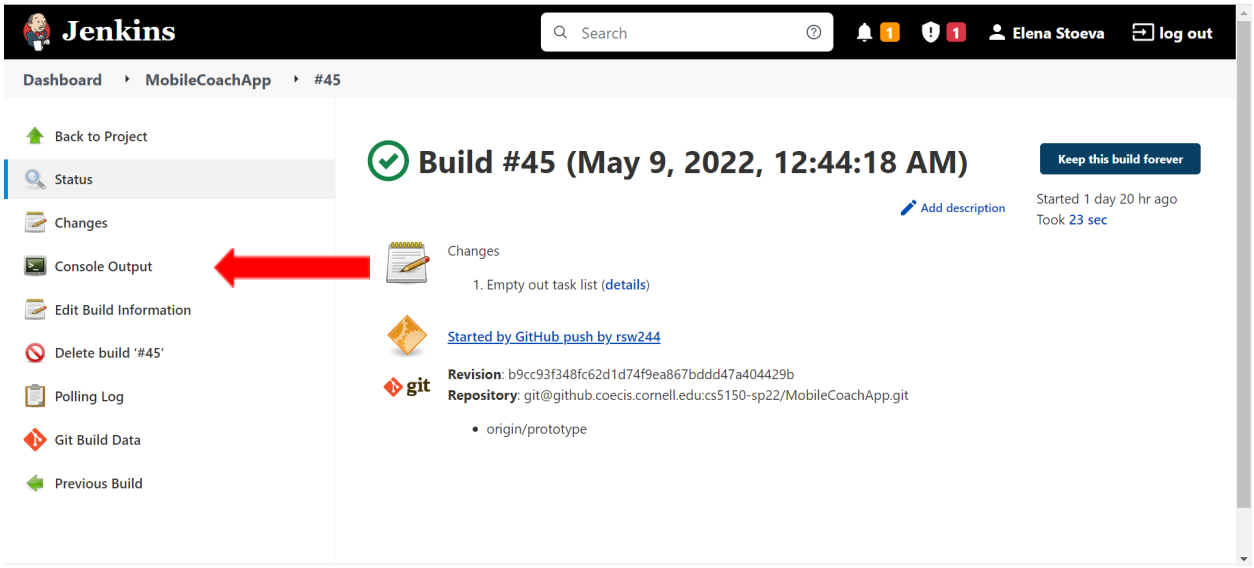


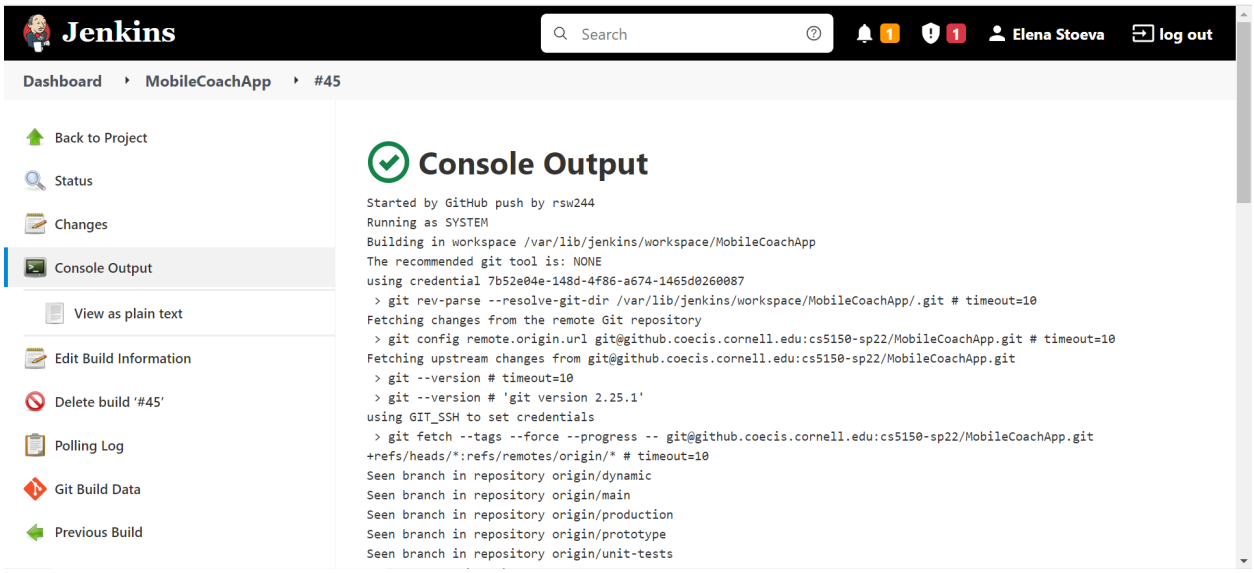


Finally, click on the "Add Webhook" button.

Now, your Jenkins CI should be set up. You can test it by making a small change in the repository (e.g. changing the README file) and pushing it. Then go to the Jenkins project that we configured above and look at the "Build History" section on the left. There should be a new build. You can click on the build to see more details. For example, after you click on the build, you can click on the "Console" button to see all commands that were executed and their logs.







IV. Revised System Design

For our project, we adopted an iterative refinement methodology, where we would iterate upon the features that we developed. Combined with a greater knowledge of our system components and interactions, we would like to produce a revised version of our system design, which we hope is helpful for maintainers of this system for getting a grasp of the system in a simpler manner than what was provided to us. In Figure 1 (UML Deployment Diagram), we identify the different components of the architecture of the system we deployed: apps on phones, Web app on a computer, four docker images on a Linux server (tomcat, deepstream, nginx and mongodb). All these services communicate with one another

over HTTPS to both render an app on a phone and a controlling web app for interventions on a laptop.

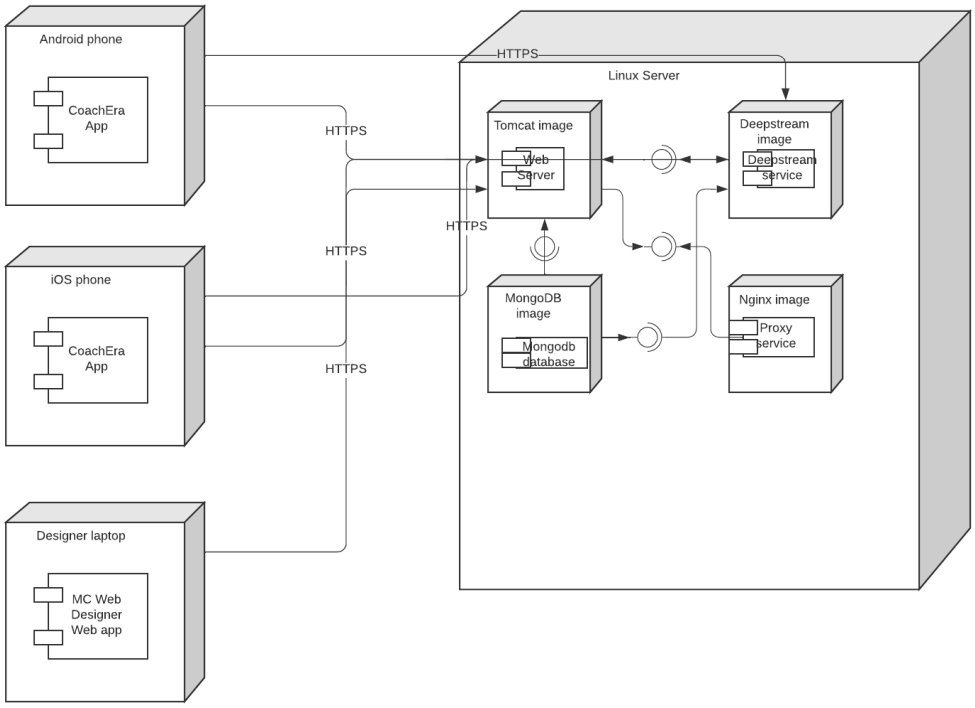


Figure 1: UML Deployment Diagram

Note: the capacity of the MongoDB is limited. After 500 new participants are added the server will start to slow down. Some support from Cornell IT is required.

V. Style Guide

A *style guide* is a documentation of standards that should be followed if chosen to be applied to a codebase. The MobileCoach codebase, particularly the code for the newly implemented features (Dashboard, Chat, FAQ), follows the Google JavaScript style guide, which can be found at https://google.github.io/styleguide/jsguide.html. This style guide contains different conventions that programmers like you should follow if they plan on writing additional JavaScript code to extend existing features made by our team.

For example, Section (3.4.1) of the style guide corresponds to following a certain convention when writing import statements. The style guide specifies that a `.js` extension *must* be added to the end of the file path in order to be compliant with the style guide. It is thus good practice to review the style guide before you begin writing code so that you do not have to make significant style modifications when you are done writing code, and particularly when you anticipate on writing lots of lines of code.

Please note that the conventions in this style guide are only applied to the modifications that were made to the initial state of the MobileCoach repository. If you plan on applying the rules defined in the style guide across all components of the codebase, please ensure that all the code that you plan on modifying is consistent with the other parts of the codebase.

VI. Future UI Designs

We do foresee potential changes that could be made to improve the user interface of our system. For example, there are improvements we thought about for the Dashboard page, including layout of different components on the screen, accessibility, and usefulness.

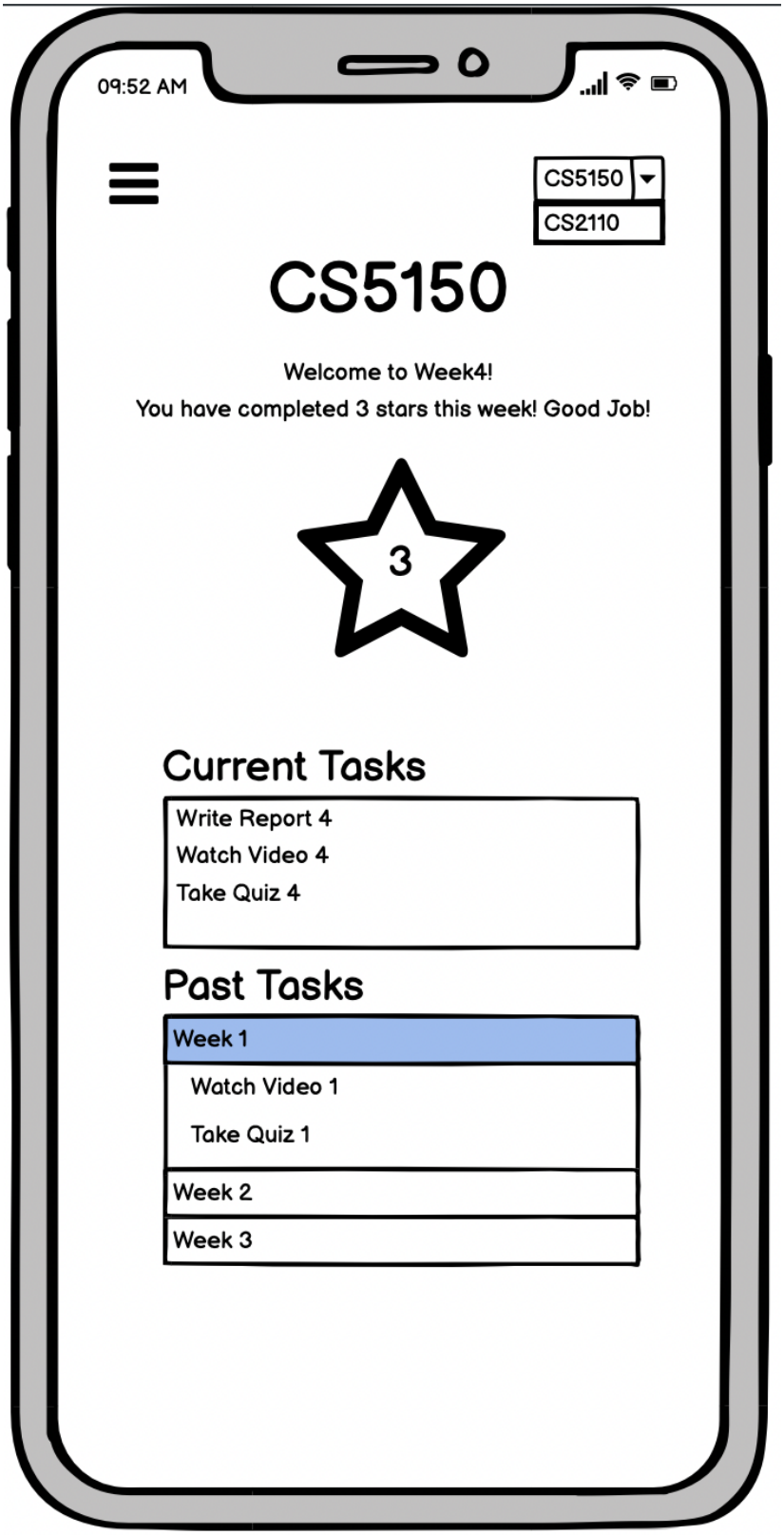
In this section, we propose some of the improvements that we had in mind regarding the user interface of the application.

A. Chat

The chat screen is mostly provided by the original MobileCoach app. Current screen conforms to the convention for chatbot interface and is functioning as we intended. We do not propose a new interface for this screen to fit our users' existing mental model as much as possible. If there are new features added in the future, corresponding changes can be made to this screen to accommodate the new functionalities.

B. Dashboard

The dashboard screen is primarily designed and implemented by our team. It has been through several iterations. We propose a new iteration for the layout according to our latest user feedback, as shown below.


09:52 AM
CS5150
CS2110
CS5150
Welcome to Week4!
You have completed 3 stars this week! Good Job!
3
Current Tasks
Write Report 4
Watch Video 4
Take Quiz 4
Past Tasks
Week 1
Watch Video 1
Take Quiz 1
Week 2
Week 3

In this new design, we added the course choosing screen so it supports tracking of multiple courses at the same time. The stars are tracked using a big star icon instead of a line of stars to provide a clearer representation of achievements, which also gamify the star earning process. Current tasks and past tasks are wrapped in blocks for a better appearance. Also for past tasks, users can now check specifically what they have accomplished with the accordion instead of just the number of stars.

VII. Deployment Procedure

Terminology

Term	Definition
Coach/Agent	The chatbot system interacting with the user of the MobileCoach application.
User	The individual interacting with the chatbot system by entering queries.
Digital Intervention	A series of communication messages between the user and the chatbot coach.
MobileCoach Web Designer	The server platform we hosted on the Cornell server. Controls the interaction between the user and the chatbot.
Configuration File	A .json file that contains a list of initial parameters and settings a project will need to get started.
Yarn	A powerful package manager used to install dependencies for building the mobile applications.

Dependency	An essential functionality, library or piece of code that is essential for a specific part of the code to work.
Metro Bundler	A packager that starts up when building and running React Native project. It combines all the JavaScript files into a single file, translates it for the device, and converts assets into objects that can be displayed by an image component.

(A.1) Interacting with the Staging Environment [iOS & Android Apps via Emulator]

1. Downloading the Repository

(1.1) Clone the GitHub repository containing the MobileCoach application files by running the following command,

git clone https://bitbucket.org/mobilecoach/mobilecoach-mobile-app.git,

in a Terminal window (MacOS) or in a Command Prompt window (Windows OS).

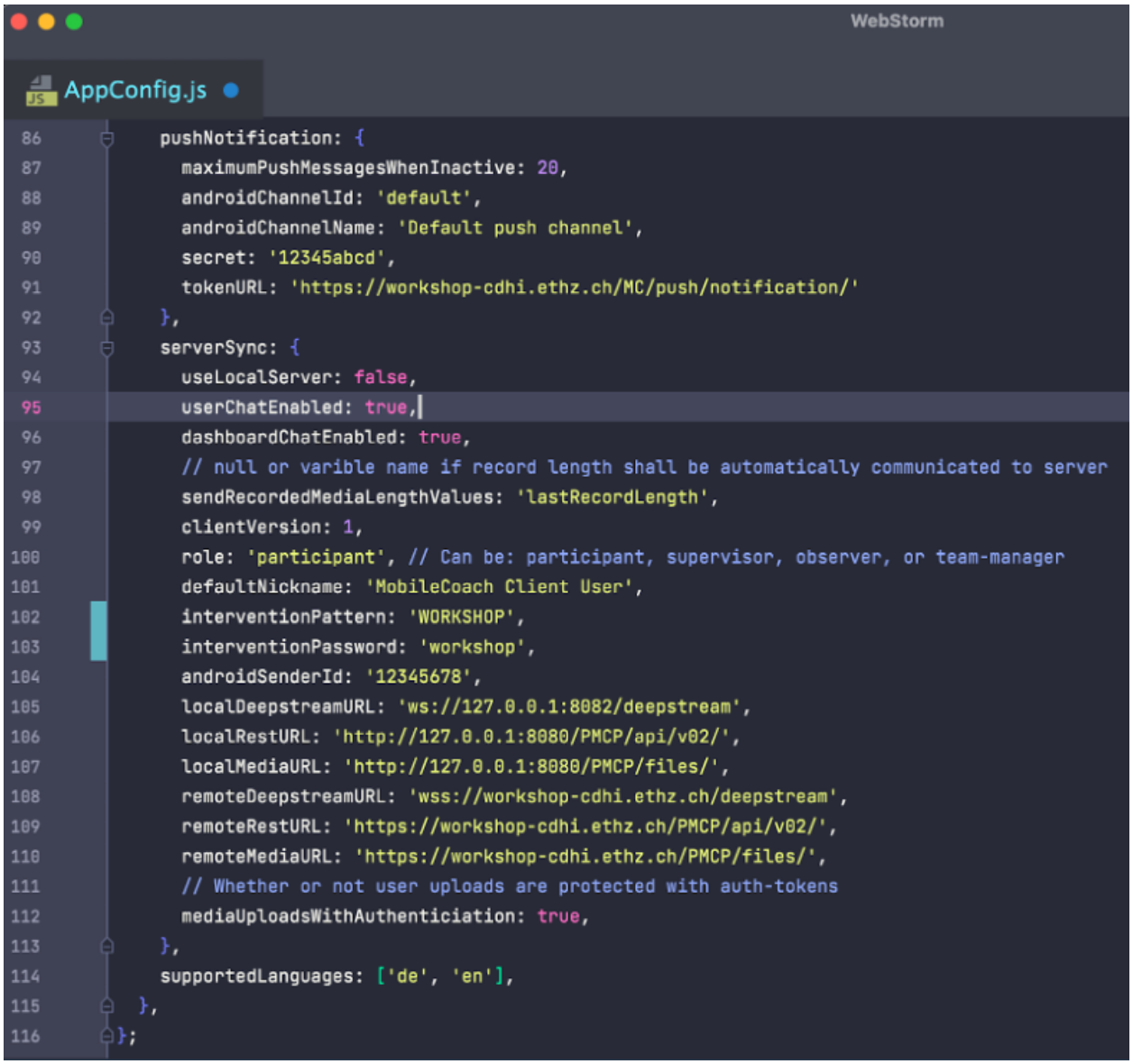
(1.2) Now, you should have a folder named mobilecoach-mobile-app in your home directory.

2. Installing all necessary dependencies
   (2.1) Navigate inside the repository, and execute yarn install to install the Node dependencies.

3. Running the current state of the application

4. Changing the Conversational Flow in MobileCoach Web Designer
   Currently, the client has been connected to the PROTOTYPE intervention. This intervention is for the client to add their own conversational flow(s).

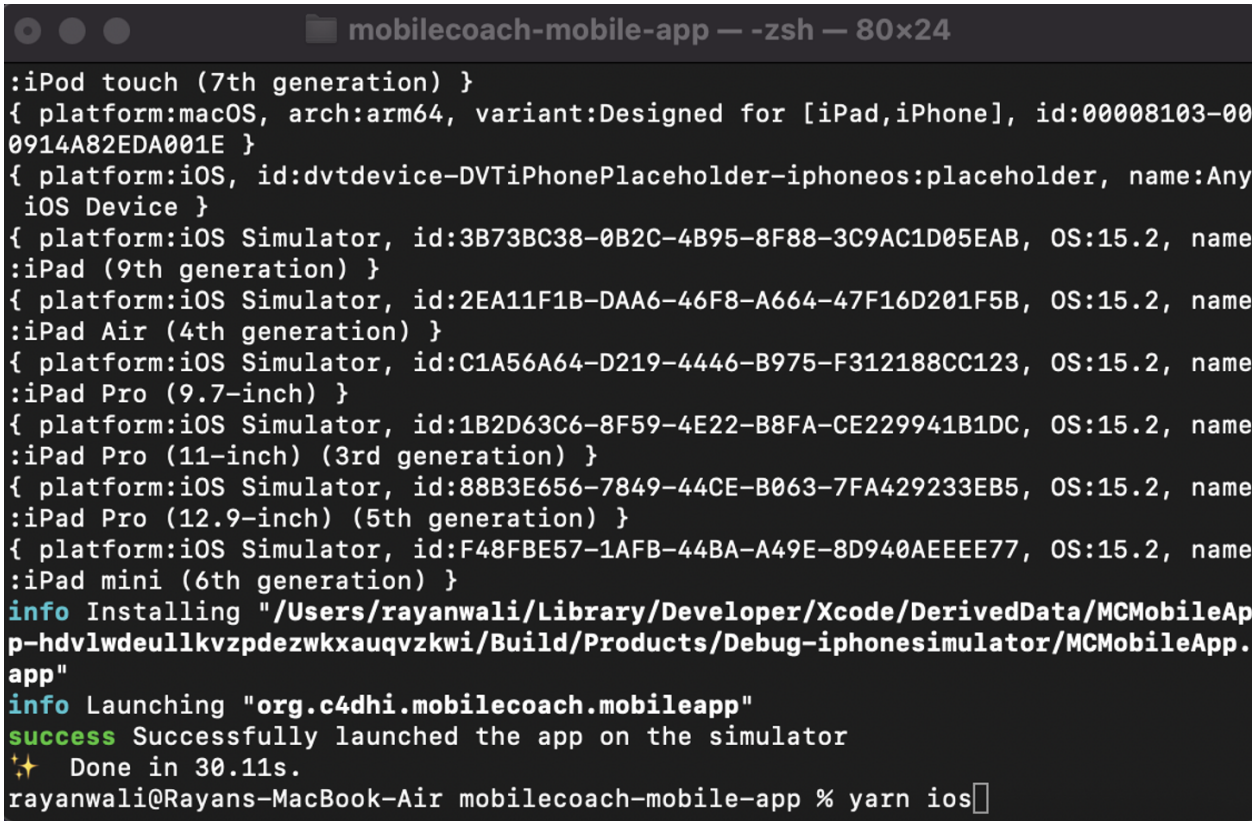
5. Changing the interventionPattern variable in AppConfig.js from PROTOTYPE



6. Re-executing the iOS/Android application

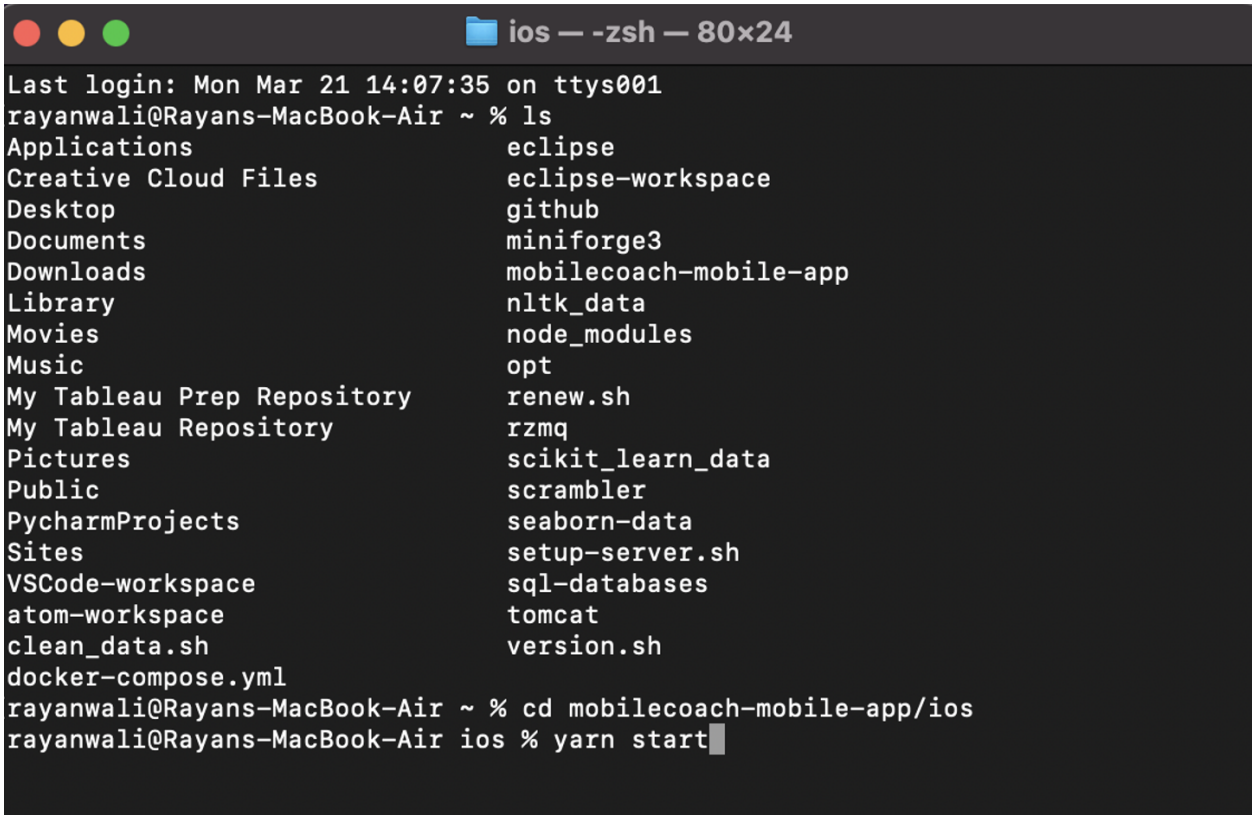
(6.1) The iOS Application
Prerequisites: For this step, you will need to have the XCode application installed and an emulator running iOS set up.

(6.1.1) Once inside the mobilecoach-mobile-app repository, run yarn ios.



(6.1.2) Run yarn start to allow the application to connect to the application Bundle.
Warning: If you do not perform this step correctly, you will run into a bundle

error when opening the application.

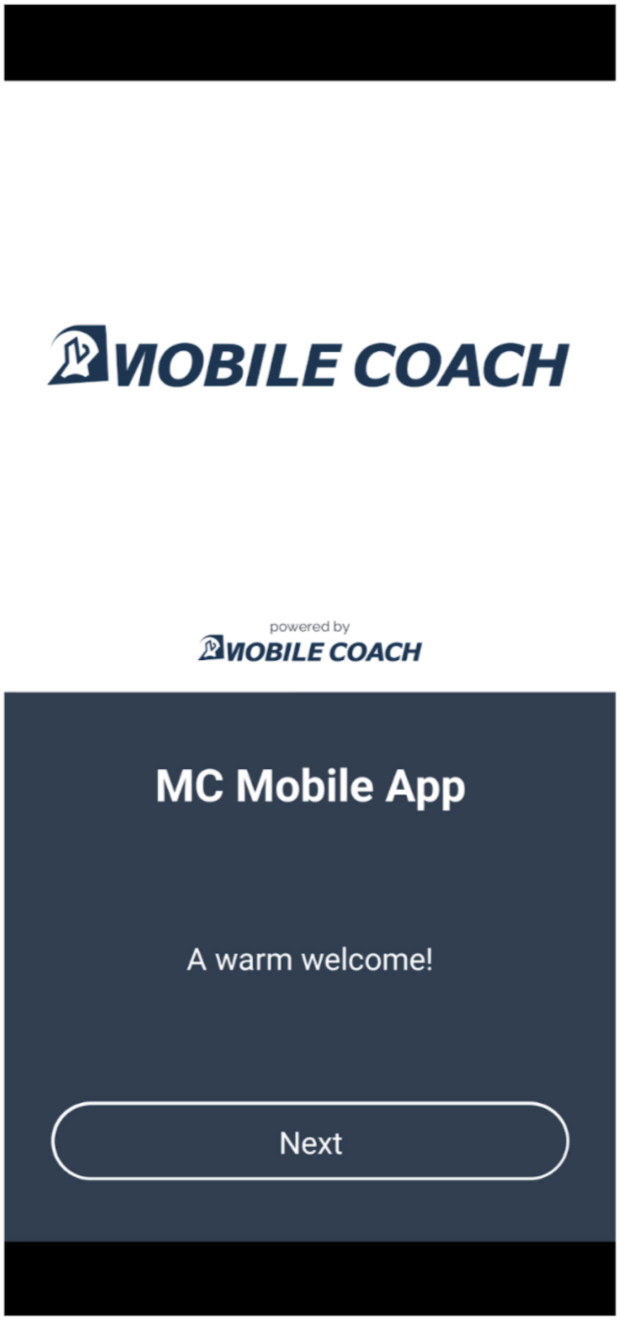


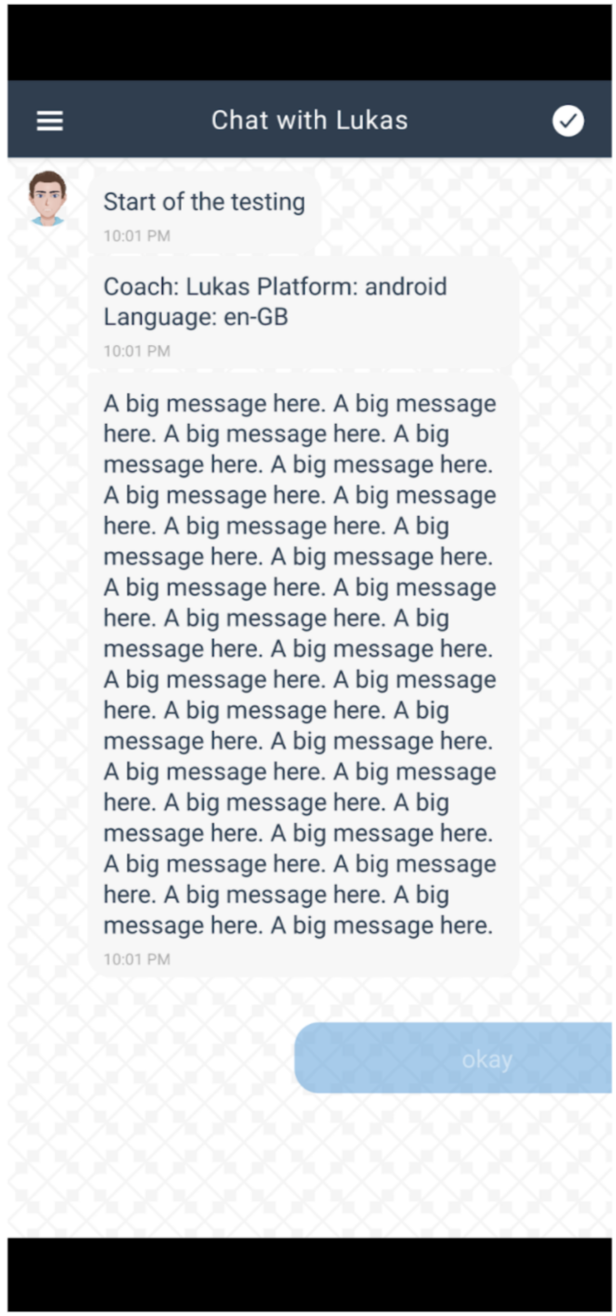
(6.2) The Android Application

Prerequisites: For this step, you will need to have the Android Studio application installed and an emulator running Android OS set up.

(6.2.1) Once inside the mobilecoach-mobile-app repository, run yarn android.

(6.2.2) Run yarn start to allow the application to connect to the application bundle.





(A.2) Interacting with the Testing Environment [native Android application]

The testing environment, unlike the staging environment, does not require the client to install any dependencies. Rather, they can simply download the testing version of the app on Google Play Store and test the application on an Android device.

1. Visit https://play.google.com/apps/internaltest/4701582881664393291, which when clicked on, will download and install the MobileCoach application.
2. Open the installed MobileCoach application and you may begin using it!

(A.3) Releasing an Android app on the Google Play Store

1. Generating an upload key; execute the following command:
   keytool -genkeypair -v -storetype PKCS12 -keystore my-upload-key.keystore -alias my-key-alias -keyalg RSA -keysize 2048 -validity 10000

2. Setting up gradle variables
   (2.1) Place the my-upload-key.keystore in the android/app folder.
   (2.2) Edit the file ~/.gradle/gradle.properties or android/gradle.properties, and add the following:

```
MYAPP_UPLOAD_STORE_FILE=my-upload-key.keystore
MYAPP_UPLOAD_KEY_ALIAS=my-key-alias
MYAPP_UPLOAD_STORE_PASSWORD=*****
MYAPP_UPLOAD_KEY_PASSWORD=*****
```

3. Adding signing config to the app's Gradle config
   (3.1) Edit the file android/app/build.gradle in your project folder, and add the signing config:

```
...
android {
    ...
    defaultConfig { ... }
    signingConfigs {
        release {
            if (project.hasProperty('MYAPP_UPLOAD_STORE_FILE')) {
                storeFile file(MYAPP_UPLOAD_STORE_FILE)
                storePassword MYAPP_UPLOAD_STORE_PASSWORD
                keyAlias MYAPP_UPLOAD_KEY_ALIAS
                keyPassword MYAPP_UPLOAD_KEY_PASSWORD
            }
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release
        }
    }
```

```
}
...
```

4. Generating the release AAB

(4.1) Run the following commands in a Terminal window:

```
cd android
 ./gradlew bundleRelease
```

(4.2) The generated AAB can be found under

android/app/build/outputs/bundle/release/app-release.aab and is ready to be uploaded to Google Play.

5. Testing the release build of the app

(5.1) Before uploading the release build to the Play Store, make sure you test it thoroughly. First uninstall any previous version of the app you already have installed. Install it on the device using the following command in the project root:

```
npx react-native run-android --variant=release
```

6. Creating a developer account on the Google Play console

   (6.1) Navigate to https://play.google.com/.

7. Creating the app and follow the steps on the Google Play console

   (7.1) Navigate to the "All Apps" tab.

Development Tools

As convention we always use yarn (you can also use npm, but just don't mix both and makes sure to never have both lockfiles).
Just install yarn as a global npm module (https://classic.yarnpkg.com/en/docs/install/#mac-stable))

Install Node Dependencies

yarn install or just yarn

Run

yarn android

or

yarn ios

Clean Native Projects

yarn clean:android

or

yarn clean:ios

App Configuration:

Mainly in these files:

- App/Config/AppConfig.js
- App/Themes/Brand.js
- App/Themes/Colors.js
- App/Themes/Fonts.js
- App/Themes/Images.js

Building keys (Android only):

android/app/build.gradle

Versions:

Version and build in android/app/build.gradle and in ios/APP_FOLDER/Info.plist

Automatic formatting in VSCode:

Add the following snippet to your user settings:

```
  "saveAndRunExt": {
      "commands": [
```

```
    {
      "match": "\\.js[x]?$",
      "isShellCommand" : true,
      "cmd": "yarn fixfile '${file}'"
    },
    {
      "match": "\\.json?$",
      "isShellCommand" : true,
      "cmd": "yarn fixfile '${file}'"
    },
    {
      "match": "\\.htm[l]?$",
      "isShellCommand" : true,
      "cmd": "yarn fixfile '${file}'"
    },
    {
      "match": "\\.css$",
      "isShellCommand" : true,
      "cmd": "yarn fixfile '${file}'"
    },
    {
      "match": "\\.md$",
      "isShellCommand" : true,
      "cmd": "yarn fixfile '${file}'"
    }

  ]
}
```