**CS624 Full Stack Development – Mobile App (React Native)**

**HOS03A: Building React Native App – Part 1**

Created by Veerendra Jagatha on March 4, 2023

Reviewed by Anahita Raeiszadeh on March 9, 2023

Revised by Sam Chung on March 30, 2023

Reviewed by Anahita Raeiszadeh on September 22, 2023

School of Technology & Computing (STC)

City University of Seattle (CityU)

**Before You Start**

* Screenshots may be different from your environment.
* The directory path shown in screenshots may be different from yours.
* Version numbers may not match the most current version at the time of writing. If given the option to choose between the stable release (long-term support) or the most recent, please select the **stable release** rather than the beta-testing version.
* There might be subtle discrepancies along with the steps. Please **use your best judgment** while going through this cookbook-style tutorial to complete each step.
* If you are not familiar with a terminal, command line, and bash scripts, check out this video: <https://youtu.be/Dp7uw9c6QH8>
* All the steps and concepts in this tutorial are from the textbook, so if you encounter problems, please **try to read and compare the textbook to solve the problem**. If you still can't solve the problem, don't hesitate to contact your course TA.
* **Avoid copy-pasting code from the book or the GitHub repository**. Instead, type out the code yourself as much as you can. Resort to copy-pasting only when you are stuck and find things not working as expected.
* Some steps may not be explained in detail. If you are not sure what to do:

1. Consult the resources from the course.
2. If you cannot solve the problem after a few tries, ask a TA for help.

#### **Readings and Examples:**

* Visit [CS 624 Repository for Examples](https://github.com/samchung0117/cs624-examples).
  + Select the related module.
  + Visit the README.md file.
  + Find examples for your practices.
* Dabit, N. (2019). [React Native in Action](https://learning.oreilly.com/library/view/react-native-in/9781617294051/). Manning Publications. (ISBN 9781617294051)
  + Chapter 3: *Building your first React Native app  
    (You can find all source codes in Chapter 3. But it needs careful tracking for file structure.)*

**Learning Outcomes**

* Section 1: Accessing GitHub Codespaces
* Section 2: Creating a mobile app
* Section 3: Understanding the layout of the todos app
* Section 4: Creating the Heading user interface
* Section 5: Creating the Input user interface
* Section 6: Pushing your work to GitHub

**Section 1: Accessing GitHub Codespaces**

It would be best to have a GitHub account and an active internet connection to access the Codespaces environment.

If you have questions about this section, please revisit HOS01 Section 1.  
You will repeat the same steps for each HOS.

1. From your course shell, visit the HOS.   
   Visit the HOS GitHub Classroom.
2. Go to your repository created from Brightspace on the GitHub classroom.
3. Download the given HOS guidelines that you will use for your hands-on practices.  
   Also, check the README.md for your HOS.
4. Click the "<>Code" dropdown menu in the top-right corner.
5. Select "Create a codespace on main."

Graphical user interface, text, application, chat or text message

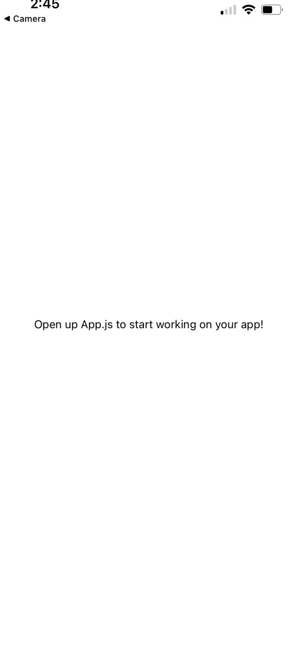
Description automatically generated

Wait for the Codespaces environment to load. Once loaded, you can access the terminal, file explorer, and other tools to start working on your project.

Once your codespace is created, the template repository will be automatically cloned into it. Now you can run the application and launch it in a browser called cloud IDE (Integrated Development Environment). We use Visual Studio Code for the web.

**Section 2: Creating a mobile app**

1. Suppose you are under the current HOS directory, HOS02.  
   Run the command “**expo init todos”** to create a new Expo project.  
   Then Select a blank template for your mobile app.
2. You can see a directory called “todos” was created.
3. Change the current directory to the newly created project using the command “**cd todos**.”
4. Run the command “**expo start --tunnel”** to start the development server.
5. Scan the QR code and you can have the following screen below.



**Section 3: Understanding the layout of the todos app**

The finished app is depicted in the figure below so you can visualize the components you'll require and their organizational structure. The application is divided between components and container components.

**Diagram

Description automatically generated**

A heading, a text entry, a button, and a tab bar will all be visible in the app. The app will display the new todo below the input when you enter a task and add it to the list of tasks. There will be two buttons for each to-do: Done and Delete. It will be marked as completed when you click the Done button, and it will be deleted when you click the Delete button. The tab bar at the bottom of the screen filters the tasks based on whether they are finished or not.

The following table shows what user interfaces will be covered in HOS03A, PE03 Part 1, HOS04, and PE03 Part 2.

|  |  |  |
| --- | --- | --- |
| **HOS or PE** | **User Interface** | **Required** |
| **HOS03A** | **Heading, Input** |  |
| PE03 Part 1 | Button | Copy HOS03A |
| HOS04A | Todo List, Todo, Todo Button | Copy PE03 Part 1 |
| PE03 Part 2 | Tab bar and Tab bar item | Copy HOS04A |

The screenshots below show the screen before and after HOS03A and PE03 Part 1, respectively.

**HOS3A**

|  |  |
| --- | --- |
| **Before** | **After** |
|  | **Graphical user interface, text, application  Description automatically generated** |

**PE03 Part 1**

|  |  |
| --- | --- |
| **Before** | **After** |
| **Graphical user interface, text, application  Description automatically generated** | **A screenshot of a computer  Description automatically generated with low confidence** |
|  |  |

**Section 4: Creating the Heading user interface**

Let us now create the App component. In App.js, replace the existing code with the following. You import a new component named ScrollView, which is essentially a scrollable View component and wraps the platform ScrollView. The keyboard will be closed if it is open, and any onPress events can be handled by the UI thanks to the keyboardShouldPersistTaps prop, which now has the value always. Make sure that the ScrollView and its parent View both have a value of flex:1. A style setting called flex:1 causes the component to occupy the entire area of its parent container.

Text

Description automatically generated

Create an initial state right away for a few of the values you'll use later. Add a constructor and an initial state to the class in App.js before the render method and initialize these values in the state.

Text

Description automatically generated

The Heading component should then be created and styled. Create a file called Heading.js, and type the code as shown in the image and place it in the root folder. This will be a stateless component.

Text

Description automatically generated

Return to App.js, import the Heading component, and replace the empty View that was previously there with the Heading component.

Text

Description automatically generated

If you check the simulator using the expo go client, you will see the screen as below. Capture the complete simulator screenshot and save it in the module03 folder. Steps to verify the changes using the simulator are mentioned at the end of the document.

A picture containing shape

Description automatically generated

What is the purpose of this section? i.e., why did you practice this section?

**Section 5: Creating the Input user interface**

The TextInput component should then be created and styled. Make a file called Input.js and place it in the root folder.

Text

Description automatically generated

Let us now create an inputChange function in App.js and import the Input component. We pass the inputValue stored on the state to Input as a prop. inputChange takes one argument, the value of the TextInput, and updates the inputValue in the state with the returned value from the TextInput.

**Text

Description automatically generated**

Now let us update the Input.js component with the new inputChange function and the inputValue property. The inputChange function is called whenever the value of the TextInput changes, and the new value is then provided to the parent component to set the state of inputValue. Additionally, you set the TextInput's value to inputValue so that you may later control and reset it. Every time the value of the TextInput component is changed, the method onChangeText is called and is supplied the new value.

Text

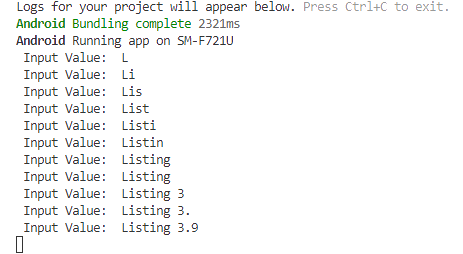
Description automatically generatedr

Now save the changes and see that the simulator window should be as shown in the image below. Type r in the console to reload.

**Graphical user interface, text, application

Description automatically generated**

Start typing something in the textbox and capture the console output and save it in the module03 folder. The console output in the browser should show something like the image below.

****

What is the purpose of this section? i.e., why did you practice this section?

**(The source code of HOS03A will be used for your PE03 Part 1 later.)**

**Steps to verify the changes using the simulator “Expo Go”:**

1. Open the terminal in the GitHub Codespaces environment.
2. Type **expo start --tunnel** and press Enter to start the expo development server.
3. Wait for the development server to load and show the QR code.
4. Open the "Expo Go" app from your mobile device.
5. Scan the QR code shown in the terminal with the "Expo Go" app or the Camera app (iOS).
6. Wait for the app to load on the mobile device.
7. If you are already connected and want to reload, you can press r in the terminal.

**Section 6: Pushing your work to GitHub**

* 1. Go to Source Control on your GitHub codespace and observe the pending changes.

Graphical user interface, text, application

Description automatically generated

* 1. Type the Message for your changes on the Message box on the top. Example, “**Submission for Module03 – Your Name**”
  2. Click on the dropdown beside commit button and select **Commit & Push** to update the changes to your repository main branch.
  3. Select **Yes** when prompted.

Graphical user interface, application

Description automatically generated