**CS624 Full Stack Development – Mobile App**

**HOS05A: React Native Styling I**

Revised by Sam Chung on April 26, 2023

Revised by Clark Ngo January 30, 2025

Reviewed by Naveena Moddu April 29, 2025

School of Technology & Computing (STC)

City University of Seattle (CityU)

**Introduction:**

To create an engaging mobile application using Reactive Native, understanding the fundamentals of component styling is crucial. In this hands-on session (HOS), we will briefly introduce concepts with a few examples and then walk through the process of styling a real object - a Profile Card. You will study styling using JavaScript, applying and organizing styles, and applying styles to View components.

**Before You Start**

* **Screenshots may be different from your environment.**
* The directory path shown in screenshots may be different from yours.
* 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.
* 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 (usually 15 -30 minutes), ask a TA for help.

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

* Visit the [CS624 Repository for Examples.](https://github.com/cityuseattle/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 4: *4 Introduction to styling*

**Learning Outcomes**

* Section 1: Accessing GitHub Codespaces
* Section 2: Creating a mobile app
* Section 3: Applying styles in applications
* Section 4: Organizing styles
* Section 5: Application that toggles between light and dark themes
* Section 6: Styling View component
* Section 7: Pushing your work to GitHub

**Section 1: Accessing GitHub Codespaces**

As we learned in the previous modules, access the GitHub Codespaces for your HOS.

**Section 2: Creating a mobile app**

Follow the steps outlined in the previous modules and set up React Native Environment with Expo Go and then create an application named “Styling.”

Launch a new terminal window in the Codespaces and then navigate to the Styling folder using the below command

**>> npx create-expo-app Styling**

**>> cd Styling**

**>> npx expo start --tunnel**

Follow the steps mentioned at the end of this document (“**Steps to verify the changes using the Simulator”**)to launch the app on your mobile device using “Expo Go.” If everything works, you will see the basic template working.

A screenshot of a cell phone

Description automatically generated

**Section 3: Applying styles in applications**

**We assume that you update the “index.tsx” and invoke the App.js component. (No more will be explained for your HOS).**

import App from './App';

export default function HomeScreen() {

  return (

    <App />

  );

}

There are several ways to add styles to elements in React Native.

**Using inline styles**

Replace the following code (Listing 4.1 Using inline styles) in App.js and observe the output in the simulator. Type ‘r’ in the terminal to reload the application anytime.

**Graphical user interface, text

Description automatically generated**

**A white background with a black and white flag

Description automatically generated with medium confidence**

**Referencing styles defined in a StyleSheet**

Now let’s change the code (Listing 4.2 Referencing styles defined in a StyleSheet) in App.js to match the following and save it. Observe the changes in the simulator.

**Graphical user interface, text

Description automatically generated**

Functionally, there’s no difference between using an inline style versus referencing a style defined in a StyleSheet. You can build a style object in StyleSheet and refer to each style separately. The code is simpler to comprehend and encourages the reuse of styles across components when the styles and render method are separated.

**Section 4: Organizing styles**

If you’re used to writing CSS, putting your styles into a separate file might seem like a better approach and feel more familiar. The stylesheet definitions are created in a separate file. You can name it whatever you want (styles.js is typical) but be sure the extension is **.js**; it’s JavaScript, after all. The stylesheet file and component file are saved in the same folder.

**Externalizing a component’s stylesheets**

Create a file styles.js and save it in the same folder as App.js, as shown below.  
Check Listing 4.3 Externalizing a component’s stylesheets.

**Text, timeline

Description automatically generated**

**Importing external stylesheets**

Now, let us import styles and make changes to App.js to match the following.

Check Listing 4.4 Importing external stylesheets.  
(Because we use the external stylesheet, you need to delete the block of const styles and containers in the App.js.)

**A screenshot of a computer program

Description automatically generated**

Observe the output below in the simulator and capture the complete screenshot of the simulator. Save it in the Module05 folder.

A screenshot of a phone

Description automatically generated

**Section 5: Application that toggles between light and dark themes**

Profiting from the fact that we're using JavaScript, let's begin to view styles as code. You'll create a straightforward application with a button that allows the user to switch the theme from light to dark.

Update “styles.js” to match the following code.

Check Listing 4.5 Dynamic stylesheets extracted from the main component file.

**A picture containing timeline

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

Update App.js to match the following code.

Check **Listing 4.6** Application that toggles between light and dark themes.

**Text

Description automatically generated**

Reload the application in the simulator and observe the theme changing by clicking on the button in the center of the screen. Save the complete simulator screenshots in the Module05 folder.

A screenshot of a white screen

Description automatically generated A screenshot of a cell phone

Description automatically generated

## **Section 6: Styling view component**

One of the most crucial components to understanding to achieve accurate styling is the View component, which serves as the foundation of a UI. Remember that a View element is comparable to an HTML “div” tag in that you can use it to enclose other components and create UI code building blocks inside of it. As you move through the HOS, you will apply everything you have learned to construct an actual component, a Profile Card. The way to put things together will be demonstrated by building the profile card.

In the process of creating this component, you will learn how to do the following:

* Create a border around the profile container using **borderWidth**
* Round the corners of that border with **borderRadius**
* Create a border that looks like a circle by using a **borderRadius** half the size of the component’s width
* Position everything using **margin** and **padding** properties

**The Initial framework for the Profile Card component**

Update the code in “App.js” with the following**.** Take a screenshot of the output.

Check **Listing 4.7** Initial framework for the Profile Card component.

Timeline

Description automatically generated

**A blue rectangle on a white background

Description automatically generated**

**Adding borders to your Profile Card component**

Download a “user.png” from <https://icons8.com/icons/set/user>.

A screenshot of a computer

Description automatically generated

Store the “user.png” under “app/assets/images/user.png.”

A screenshot of a computer

Description automatically generated

Because of the image location, you need to update the path inside the “source={require('./assets/user.png')}’ in App.js into:



Check **Listing 4.10** incorporating border properties into the Profile Card.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

The simulator should be showing the following output.

**Shape

Description automatically generated**

There are some layout issues with the Profile Card. We will discuss adjusting the layout using the margin and padding styles in the next.

**Specifying margins, padding, and positioning the Profile Card**

To understand more about setting margins, padding, and positioning, please refer to Chapter 4, Section 4.2.3 in the textbook.

Update the styles code in “App.js” to match the following.

Check Listing 4.14 Modifying Profile Card styles to fix the layout.

* Aligns the circle in the horizontal center of the Profile Card.
* Aligns the user image in the horizontal center of the circle.
* Provides space between the top of the circle and the top of the Profile Card.
* Provides padding between the inner part of the circle and the contained image

Text

Description automatically generated

If you save the changes, you should see the profile card positioned in the image below. Take the screenshot of your simulator and save it in the Module05 folder.

Shape, square

Description automatically generated

**Steps to verify the changes using the Simulator:**

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

**Section 7: Pushing your work to GitHub**

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

Graphical user interface, text, application

Description automatically generated

* 1. Type the Message for your changes in the Message box on the top. For example,” **Submission for Module05 – Your Name**.”
  2. Click on the dropdown beside the 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