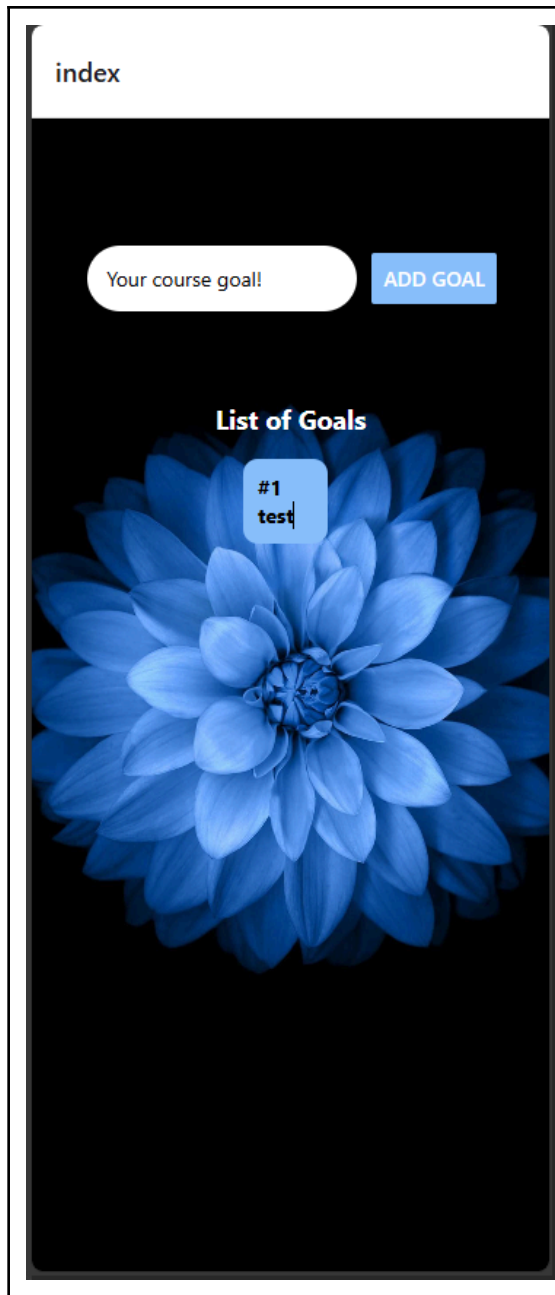


Hands-on Activity 8.1 Using Pressables	
Course Code: CPE 026	Program: Computer Engineering
Course Title: Emerging Technologies 3	Date Performed: 10/17/2024
Section: CPE41S8	Date Submitted: 10/20/2024
Name: Alferos, Joshua L. Efa, Christian Ed B.	Instructor: Engr. Roman Richard
1. Instructions	
<ol style="list-style-type: none"> 1. Perform the given tasks. 2. Follow the Hands-on Activity format from the previous activities to accomplish the given instructions below. 3. Perform the Procedures and Outputs in Section 6. 4. Answer the Supplementary Activity in Section 7. 5. Provide the following in section 8: <ol style="list-style-type: none"> A. A concise summary of this activity performed. B. Your personal conclusions/ reflections. C. An overview of the lessons you learned from this module. 6. Submission Requirements: <ol style="list-style-type: none"> A. All files must be submitted as PDF only. 	
2. Task	
<ol style="list-style-type: none"> 1. Go through the topics presented in the module: Determine the difference between a button and a pressable? 2. Create a pressable component in your goal list application instead of a button for 'Add Goal'. How is this any different? 3. Experiment with the different props of the pressable component. 	
3. Output	
<ol style="list-style-type: none"> 1. Go through the topics presented in the module: Determine the difference between a button and a pressable? <ul style="list-style-type: none"> - In React Native, a button is a basic component for rendering a button, which comes with a fixed design and limited customization options, like onPress event. Pressable, on the other hand, is more flexible and allows the user to customize the interaction for press events. The user can define what happens on a press, hold, release etc, and style it freely, making it suitable for button-like components. 2. Create a pressable component in your goal list application instead of a button for 'Add Goal'. How is this any different? 	



```

resizeMode='cover'
style={styles.backgroundImage}
>
<View style={styles.appContainer}>
  <View style={styles.inputContainer}>
    <TextInput
      onChangeText={goalInputHandler}
      value={enteredGoalText}
      style={styles.textInput}
      placeholder='Your course goal!'
    />
    <Button title='ADD GOAL' color='#888EFC' onPress={addGoalHandle} />
  </View>
  <View style={styles.goalsContainer}>
    <Text style={styles.goalsList}>List of Goals</Text>
    <FlatList
      data={courseGoals}
      renderItem={({itemData}) => (
        <Pressable
          onPress={() => {
            console.log('Pressed:', itemData.item.text);
          }}
        >
          <View style={styles.goalBox}>
            <Text style={styles.goalText}>
              #{goalCount - (courseGoals.length - itemData.index)}
            </Text>
          </View>
        </Pressable>
      )
    />
  </View>
</View>

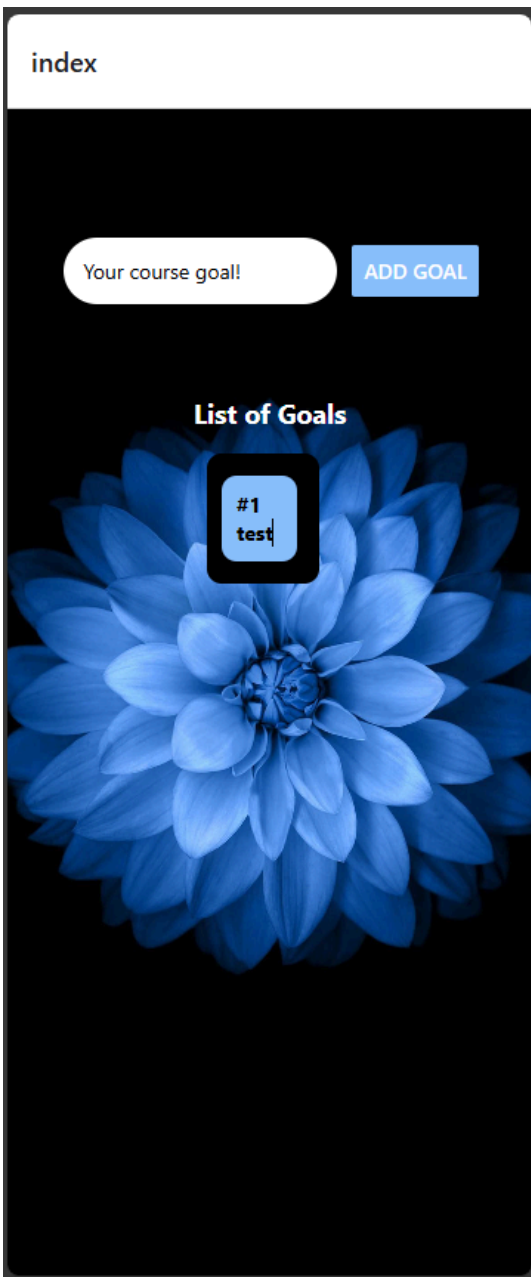
```

Server Error

Pressed: test C:\Users\tipgc\Docum-ct\apollindex.tsx:56

In here, I made each added goal in the courseGoals to be pressable. This is different compared to before because I can now add custom actions whenever the added goal is clicked..

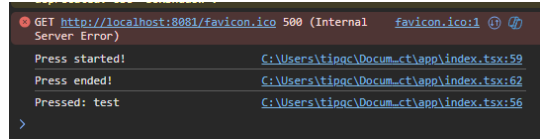
3. Experiment with the different props of the pressable component.



```

View style={styles.goalsContainer}>
  <Text style={styles.goalsList}>List of Goals</Text>
  <FlatList
    data={courseGoals}
    renderItem={({itemData}) => (
      <Pressable
        onPress={() => {
          console.log('Pressed:', itemData.item.text);
        }}
        onPressIn={() => {
          console.log('Press started!');
        }}
        onPressOut={() => {
          console.log('Press ended!');
        }}
        android_ripple={{ color: '#0000' }}
        style={({pressed}) => [
          styles.goalBox,
          pressed && { backgroundColor: '#000' },
        ]}
      >
        <View style={styles.goalBox}>
          <Text style={styles.goalText}>
            #{goalCount - (courseGoals.length - itemData

```



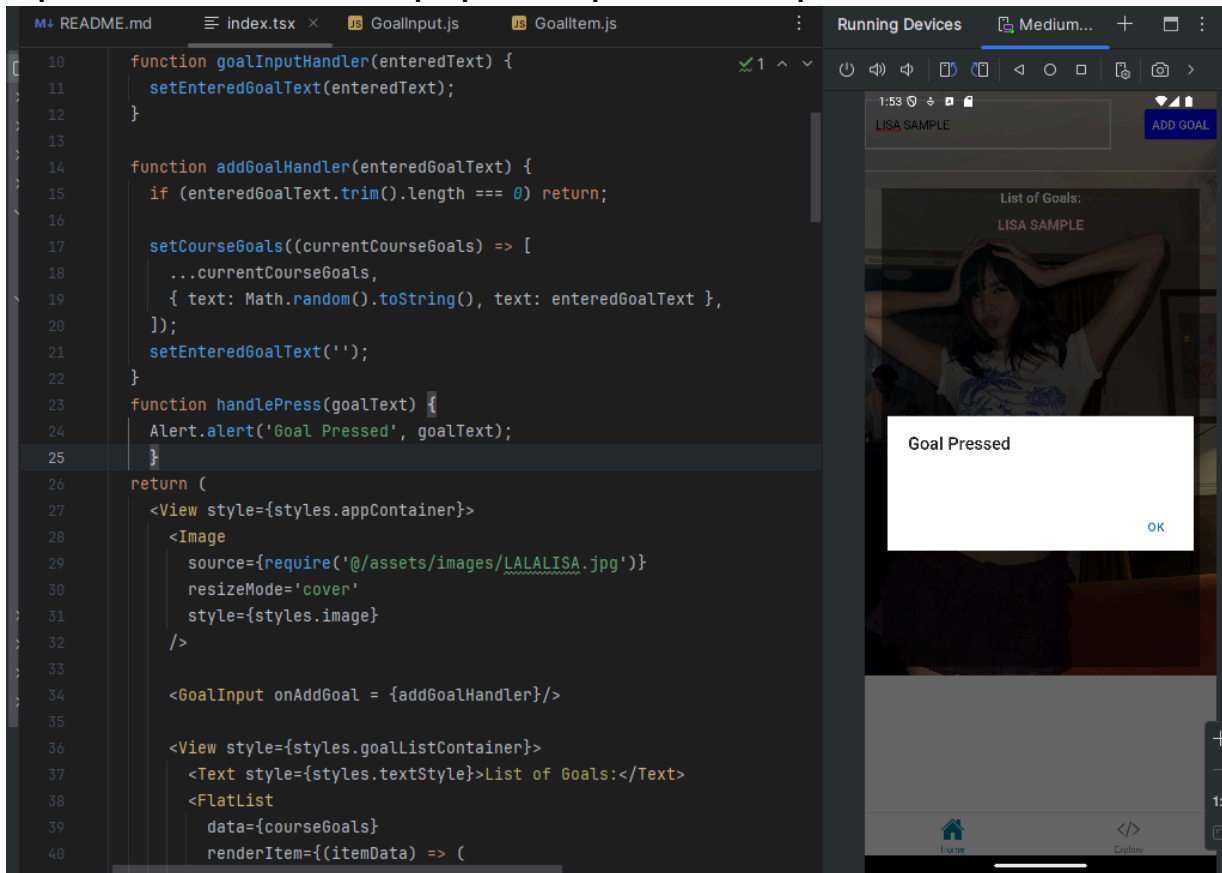
In here, we put onpress in and out to monitor the pressing action on the listed goal. Upon holding the mouse down, it is shown on the console the string 'Press started!', and upon releasing the mouse, the console will show 'Press ended!'. The android_ripple effect is a prop that shows a border to the pressed element, as shown on the output.

EFA

Go through the topics presented in the module: Determine the difference between a button and a pressable?

- Button is Simple and easy to use with limited styling options and Best for basic, standard actions. while Pressable is more flexible, allowing full control over styling and interactions and Ideal for custom or complex touch interactions.

Create a pressable component in your goal list application instead of a button for 'Add Goal'. How is this any different?
Experiment with the different props of the pressable component.



4. Supplementary Activity

1. Create a table and show the different props of the pressable component.
2. Create a column for the description of each prop.
3. Create a column that shows the syntax of each prop.
4. Create a column that shows your demonstration of each prop's use.

ALFEROS

Prop	Description	Syntax	Demonstration
android_disableSound	If true, don't play Android system sound on press.	android_disableSound={true}	<Pressable android_disableSound={true} onPress={() => console.log('Pressed')

			<pre> item: \${itemData.item.text} `)}} style={styles.pressableItem} > </pre>
android_ripple	Enables the Android ripple effect and configures its properties.	android_ripple={{color: 'blue' }}	<pre> <Pressable android_ripple={{ color: 'lightblue', radius: 50 }} onPress={() => console.log('Pressed with ripple!')} > <Text>Press with ripple effect</Text> </Pressable> </pre>
children	Either children or a function that receives a boolean reflecting whether the component is currently pressed.	<Pressable>{children}</Pressable>	<pre> <Pressable onPress={() => console.log('Pressed !')}> <Text>Pressable with children text</Text> </Pressable> </pre>
unstable_pressDelay	Duration (in milliseconds) to wait after press down before calling onPressIn.	unstable_pressDelay={100}	<pre> <Pressable unstable_pressDelay ={500} onPress={() => console.log('Press delayed by 500ms')} > <Text>Press with delay</Text> </Pressable> </pre>

delayLongPress	Duration (in milliseconds) from onPressIn before onLongPress is called.	delayLongPress={300}	<pre> <Pressable delayLongPress={800} onLongPress={() => console.log('Long press after 800ms')} > <Text>Press and hold</Text> </Pressable> </pre>
disabled	Whether the press behavior is disabled.	disabled={true}	<pre> <Pressable disabled={true} onPress={() => console.log('This will not fire')} > <Text>Disabled Pressable</Text> </Pressable> </pre>
hitSlop	Sets additional distance outside of element in which a press can be detected.	hitSlop={{ top: 10, bottom: 10, left: 10, right: 10 }}	<pre> <Pressable hitSlop={20} onPress={() => console.log('Pressed with extended area')} > <Text>Pressable with extended hit area</Text> </Pressable> </pre>
onHoverIn	Called when the hover is activated to provide visual feedback.	onHoverIn={() => console.log('Hovered In')}	<pre> <Pressable onHoverIn={() => console.log('Hovered in')} > <Text>Hover over me</Text> </Pressable> </pre>

onHoverOut	Called when the hover is deactivated to undo visual feedback.	onHoverOut={() => console.log('Hovered Out')}	<pre> <Pressable onHoverOut={() => console.log('Hovered out')} > <Text>Hover away from me</Text> </Pressable> </pre>
onLongPress	Called if the time after onPressIn lasts longer than 500 milliseconds. This time period can be customized with delayLongPress.	onLongPress={() => console.log('Long Pressed')}	<pre> <Pressable onLongPress={() => console.log('Long press detected')} > <Text>Press and hold me</Text> </Pressable> </pre>
onPress	Called after onPressOut.	onPress={() => console.log('Pressed')}	<pre> <Pressable onPress={() => console.log('Press detected')} > <Text>Tap me!</Text> </Pressable> </pre>
onPressIn	Called immediately when a touch is engaged, before onPressOut and onPress.	onPressIn={() => console.log('Pressed In')}	<pre> <Pressable onPressIn={() => console.log('Press in detected')} > <Text>Press in me</Text> </Pressable> </pre>
onPressOut	Called when a touch is released.	onPressOut={() => console.log('Pressed Out')}	<pre> <Pressable onPressOut={() => console.log('Press out detected')} > </pre>

			<pre><Text>Press and release me</Text> </Pressable></pre>
pressRetentionOffset	Additional distance outside of this view in which a touch is considered a press before onPressOut is triggered.	<pre>pressRetentionOffset ={{ top: 10, bottom: 10, left: 10, right: 10 }}</pre>	<pre><Pressable pressRetentionOffset ={30} onPress={() => console.log('Press retained')} > <Text>Retain press within margin</Text> </Pressable></pre>
style	Either view styles or a function that receives a boolean reflecting whether the component is currently pressed and returns view styles.	<pre>style={({ pressed }) => [pressed ? styles.pressed : styles.default]}</pre>	<pre><Pressable style={({ pressed }) => [{ backgroundColor: pressed ? 'lightgray' : 'white', padding: 10, borderRadius: 5, },]} onPress={() => console.log('Styled press')} > <Text>Styled Pressable</Text> </Pressable></pre>
testOnly_pressed	Used only for documentation or testing (e.g. snapshot testing).	<pre>testOnly_pressed={tr ue}</pre>	<pre><Pressable testOnly_pressed={tr ue} > <Text>Simulated pressed state (for testing)</Text></pre>

			</Pressable>
EFA			
Types	Description	Syntax	Demonstation
android_disableSound	If true, don't play Android system sound on press.	android_disableSound={true}	<pre><Pressable android_disableSound={true} onPress={() => console.log('Pressed without sound')}> <Text>Press Me (No Sound)</Text> </Pressable></pre>
android_ripple	Enables the Android ripple effect and configures its properties.	android_ripple={{color: 'blue' }}	<pre><Pressable android_ripple={{color: 'blue' }} onPress={() => console.log('Pressed with ripple')}> <Text>Press Me (Ripple Effect)</Text> </Pressable></pre>
children	Either children or a function that receives a boolean reflecting whether the component is currently pressed.	<Pressable>{children}</Pressable>	<pre><Pressable> <Text>Press Me (Static Content)</Text> </Pressable> <Pressable> ({ pressed }) => (<Text style={{ color: pressed ? 'red' : 'black' }}> {pressed ? 'Pressed!' : 'Press Me'} </Text>) </Pressable></pre>

			</Pressable>
unstable_pressDelay	Duration (in milliseconds) to wait after press down before calling onPressIn.	unstable_pressDelay={100}	<pre> <Pressable unstable_pressDelay ={100} onPressIn={() => console.log('Press In after delay')}> <Text>Press Me (With Delay)</Text> </Pressable> </pre>
delayLongPress	Duration (in milliseconds) from onPressIn before onLongPress is called.	delayLongPress={300}	<pre> <Pressable delayLongPress={300} onPressIn={() => console.log('Long Pressed')}> <Text>Press Me (Long Press)</Text> </Pressable> </pre>
disabled	Whether the press behavior is disabled.	disabled={true}	<pre> <Pressable disabled={true} onPress={() => console.log('This will not trigger')}> <Text>Press Me (Disabled)</Text> </Pressable> </pre>
hitSlop	Sets additional distance outside of element in which a press can be detected.	hitSlop={{ top: 10, bottom: 10, left: 10, right: 10 }}	<pre> <Pressable hitSlop={{ top: 10, bottom: 10, left: 10, right: 10 }} onPress={() => console.log('Pressed with hitSlop')}> <Text>Press Me (With Hit Slop)</Text> </Pressable> </pre>

onHoverIn	Called when the hover is activated to provide visual feedback.	onHoverIn={() => console.log('Hovered In')}	<Pressable onHoverIn={() => console.log('Hovered In')} onHoverOut={() => console.log('Hovered Out')}> <Text>Hover Over Me</Text> </Pressable>
onHoverOut	Called when the hover is deactivated to undo visual feedback.	onHoverOut={() => console.log('Hovered Out')}	<Pressable onHoverIn={() => console.log('Hovered In')} onHoverOut={() => console.log('Hovered Out')}> <Text>Hover Over Me</Text> </Pressable>
onLongPress	Called if the time after onPressIn lasts longer than 500 milliseconds. This time period can be customized with delayLongPress.	onLongPress={() => console.log('Long Pressed')}	<Pressable onLongPress={() => console.log('Long Pressed')}> <Text>Press Me (Long Press)</Text> </Pressable>
onPress	Called after onPressIn.	onPress={() => console.log('Pressed')}	<Pressable onPress={() => console.log('Pressed')}> <Text>Press Me</Text> </Pressable>
onPressIn	Called immediately when a touch is engaged, before	onPressIn={() => console.log('Pressed In')}	<Pressable onPressIn={() => console.log('Pressed

	onPressOut and onPress.		In'}} onPressOut={() => console.log('Pressed Out')}> <Text>Press Me</Text> </Pressable>
onPressOut	Called when a touch is released.	onPressOut={() => console.log('Pressed Out')}	<Pressable onPressIn={() => console.log('Pressed In')} onPressOut={() => console.log('Pressed Out')}> <Text>Press Me</Text> </Pressable>
pressRetentionOffset	Additional distance outside of this view in which a touch is considered a press before onPressOut is triggered.	pressRetentionOffset={{ top: 10, bottom: 10, left: 10, right: 10 }}	<Pressable pressRetentionOffset={{ top: 10, bottom: 10, left: 10, right: 10 }}> <Text>Press Me (Press Retention Offset)</Text> </Pressable>
style	Either view styles or a function that receives a boolean reflecting whether the component is currently pressed and returns view styles.	style={({ pressed }) => [pressed ? styles.pressed : styles.default]}	const styles = StyleSheet.create({ default: { backgroundColor: 'blue' }, pressed: { backgroundColor: 'red' } }); <Pressable style={({ pressed }) => [pressed ? styles.pressed :

			<code>styles.default}}> <Text>Press Me</Text> </Pressable></code>
testOnly_pressed	Used only for documentation or testing (e.g. snapshot testing).	testOnly_pressed={true}	<code><Pressable testOnly_pressed={true}> <Text>Press Me (Testing)</Text> </Pressable></code>

5. Conclusion

ALFEROS

- In this activity, we explored the Pressable component on React Native. It is a versatile tool that enhances user interaction by providing a customizable way to handle touch events. Pressable allows us to create an engaging user interface. By incorporating props like onPress, onLongPress, android_disableSound, and the like, we can fine-tune the interaction experience of the users. Overall, Pressable shows the responsive UI elements of React Native, contributing to a more dynamic experience in mobile applications.

EFA

- The Pressable component in React Native offers a versatile way to handle touch interactions, providing extensive customization over behavior and appearance compared to the basic Button component. With Pressable, you can create interactive elements that meet complex requirements, making it a powerful choice for building custom, responsive, and engaging UIs in mobile applications.