What we will learn today?

» Accessible Building
» Storing data using AsyncStorage
» Theming Libraries
» Accessing and Using Sensor Data
» App Lifecycle using AppState
» Assignment Preview
TopHat Attendance
TopHat Questions
Accessible Building
Accessibility in Web Technologies

From the *three-layered cake* to the *Peanut M&M*:

1 Image source
Accessible Rich Internet Applications (ARIA)²

aria is a set of HTML attributes that make web components available to assistive technologies.

```html
<div id="percent-loaded" role="progressbar" aria-valuenow="75" aria-valuemin="0" aria-valuemax="100">
</div>
```

² MDN Web Docs: ARIA

© Building User Interfaces | Professor Mutlu | Week 11: React Native — 3
Accessibility in React Native

RN provides us with access to assistive technologies that mobile platforms provide (e.g., VoiceOver on iOS or TalkBack on Android) through component attributes.

```
<View accessible={true}>
  <Text>List item one</Text>
  <Text>List item two</Text>
</View>
```

3 React Native Accessibility
React Native Accessibility Properties

accessible attribute indicates whether the component is an accessibility element and, if so, groups its children in a single selectable component.

accessibilityLabel attribute defines screen reader descriptions of components. doesn’t have to be visible on screen

accessibilityHint attribute helps users understand what will happen if they perform the action on the accessibility element.
React Native Accessibility Actions

Standard, e.g., magicTap, escape, activate, increment, decrement, longpress, or custom actions, handled by onAccessibilityAction.

```javascript
onAccessibilityAction={(event) => {
    switch (event.nativeEvent.actionName) {
    case 'longpress':
        // take action
        ...
    }
}}
```
AsyncStorage
What is AsyncStorage?

AsyncStorage is a simple, unencrypted, persistent, key-value storage system that is global to the app.

Four key features:

1. **Simple**: Core functionality involves `set` and `get` methods.
2. **Unencrypted**: Access is controlled by location access.
3. **Persistent**: Data is saved until it is explicitly deleted.
4. **Global**: Saved data is global to the app.
How does it work?

We use the AsyncStorage JS library:

```javascript
import AsyncStorage from '@react-native-community/async-storage';
```

Through RN Bridge, the corresponding native code library will store the data in an appropriate format, in a dictionary or files in iOS and in a database in Android.

All AsyncStorage operations are asynchronous and therefore return a Promise.
Saving Data

storeData = async () => {
  try {
    await AsyncStorage.setItem('@storage_Key', 'stored value')
  } catch (e) {
    // saving error
  }
}
Retrieving Data

```javascript
getData = async () => {
    try {
        const value = await AsyncStorage.getItem('@storage_Key')
        if(value !== null) {
            // value previously stored
        }
    } catch(e) {
        // error reading value
    }
}
```
Other operations

`removeItem(key)` removes the item that corresponds to a key.
`mergeItem(key)` merges an existing key value with an input value.
`clear()` erases all `AsyncStorage`.
`getAllKeys()` retrieves all keys for your app.
`multiGet(keys), multiSet(keys,values), multiRemove(keys), multiMerge(keys,values)` are batch operations for array data.

---

4 More information on RN `AsyncStorage`
TopHat Quiz
Theming in React Native
Popular Theme Libraries and Toolkits

» NativeBase

» React Native Elements
NativeBase

For iOS and Android.

Customized using NativeBase Customizer.

Different themes using StyleProvider.

```react
<Button light style={{borderRadius:8}}>
  <Text>Contact Us</Text>
</Button>
```

5 Image source

6 See example in Expo
Importing themes:

```javascript
import getTheme from './native-base-theme/components';
import material from './native-base-theme/variables/material';
```

Applying themes using `getTheme()`:

```javascript
<StyleProvider style={getTheme(material)}>
  <Container>
    <Content>
      ...
    </Content>
  </Container>
</StyleProvider>
```
Sensors
Sensor Libraries

Two options:

1. React Native sensors library: react-native-sensors

2. Expo sensors library: expo-sensors
Expo Sensors Library

Provides access to device sensors through specific components:

Accelerometer: provides access to the accelerometer sensor, which captures displacement in 3D.

Barometer: provides access the device barometer sensor, which captures changes in air pressure.

Gyroscope: provides access the device gyroscope sensor, which captures changes in rotation in 3D space.
Magnetometer: provides access the device magnetometer sensor, which measures changes in the magnetic field.
MagnetometerUncalibrated: provides access to uncalibrated raw values from the magnetometer.

Pedometer: Provides step count from the native sensor libraries.
How to Access Sensor Data

Install the sensor library:

```javascript
expo install expo-sensors
```

Import the sensor component:

```javascript
import { Accelerometer } from 'expo-sensors';
```

Check if the sensor is available:

```javascript
Accelerometer.isAvailableAsync() // returns true or false
```

Create listener for sensor events:

```javascript
Accelerometer.addListener(listener)
```

Best practice is to create `subscribe` and `unsubscribe` functions:

```javascript
_subsubscribe = () => {
    this._subscription = Accelerometer.addListener(accelerometerData => {
        this.setState({ accelerometerData });
    });
};
```
To remove listeners for sensor events:

```javascript
Accelerometer.removeAllListeners()
```

To subscribe to updates to the sensor data at specified intervals:

```javascript
Accelerometer.setUpdateInterval(intervalMs)
```
Access to Other Hardware

Camera using expo-camera renders a preview of the front or the back camera.

Battery using expo-battery provides battery information.

Haptics using expo-haptics provides haptic feedback using the Taptic Engine on iOS and Vibrator system service on Android.

Audio using expo-av provides basic audio playback and recording.

Brightness using expo-brightness allows getting and setting screen brightness.
App Lifecycle Using AppState
The Problem

Everything we have been doing so far assumes that our app is loaded on the screen and is running as a foreground process.

We need to be able to perform background processes or safely save the user's data in case the OS suspends it or the user quits it.
The Solution

AppState provides information on the current state of the app:

» active indicates that the app is running in the foreground

» background indicates that the app is running in the background

» inactive indicates that the app is transitioning between foreground and background
import {AppState} from 'react-native';

state = { appState: AppState.currentState };

componentDidMount() {
    AppState.addEventListener('change', this._handleAppStateChange);
}

_handlesAppStateChange = (nextAppState) => {
    if (this.state.appState.match(/inactive|background/) && nextAppState === 'active') {
        // Do something
    }
    this.setState({appState: nextAppState});
};
Example Background Process

BackgroundFetch from expo-background-fetch allows performing background fetch tasks using the TaskManager Native API.

`BackgroundFetch.registerTaskAsync(taskName, options)`
Assignment Preview
React Native 3 Requirements

» Be able to add a meal to a day
» Be able to add foods to meals
» Summarize (aggregate) stats for meals based on foods
» Hook up stats in day view for nutritional data (aggregated over meals/foods from that day)
» Allow users to track their stats over a the past 7 days
» Clean/Clear/Attractive interface
Some Notes

1. Using endpoint for foods (not limited to these):
   https://mysqlcs639.cs.wisc.edu/foods/

1. `pluralize` package to properly format food items:
   ```
   pluralize('test', 1, true)  //=> "1 test"
   pluralize('test', 5, true)  //=> "5 tests"
   ```
What did we learn today?

» Accessible Building
» Storing data using AsyncStorage
» Theming Libraries
» Accessing and Using Sensor Data
» App Lifecycle using AppState
» Assignment Preview