Building User Interfaces

React Native Advanced Concepts

Professor Yuhang Zhao

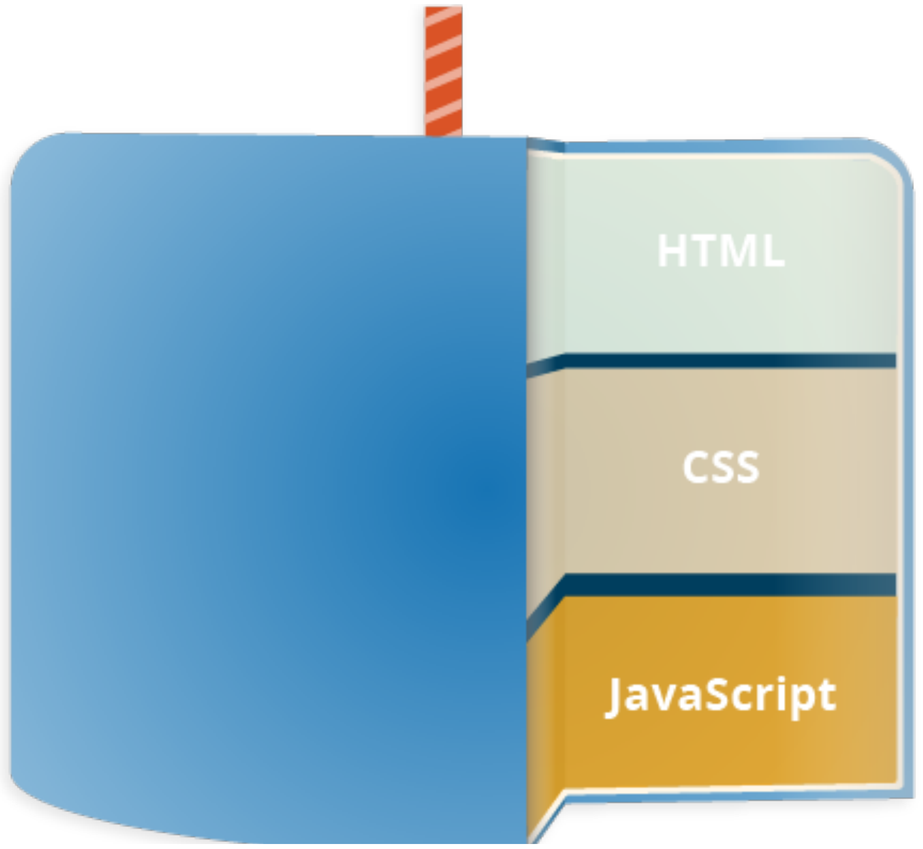
What we will learn today?

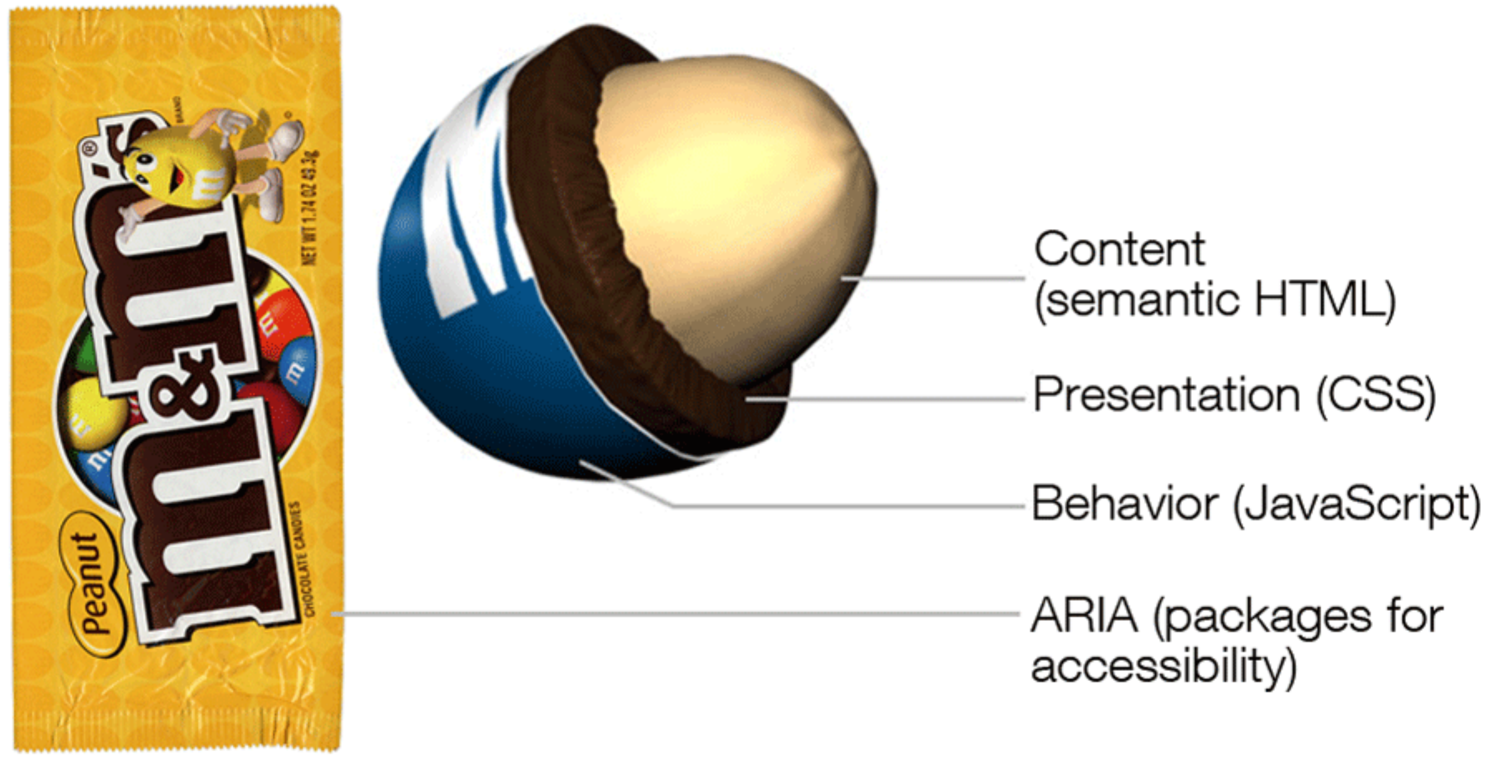
- Accessible Building
- Storing data using `AsyncStorage`
- Theming Libraries
- Accessing and Using Sensor Data
- App Lifecycle using `AppState`
- Assignment Preview

Accessible Building

Accessibility in Web Technologies[1]

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





By Dennis Lembree based on illustration by Dave Stewart

[1] Image sources: left, right

Accessible Rich Internet Applications (ARIA)[2]

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

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

[2] MDN Web Docs: ARIA

29

VoiceOver speaks items on the screen:
- Tap once to select an item.
- Double-tap to activate the selected item.

Learn more...

[29] Video source

Accessibility in React Native[3]

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.

`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`.

```
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?

```
npm install @react-native-async-storage/async-storage
```

```
import AsyncStorage from '@react-native-async-storage/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 [10]

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

[10] Example code

Other operations[5]

— `removeItem(key)` removes the item that corresponds to a `key`.

— `mergeItem(key, value)` 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.

[5] More information on RN `AsyncStorage`

Theming in React Native

Popular Theme Libraries and Toolkits

- NativeBase
- React Native Elements

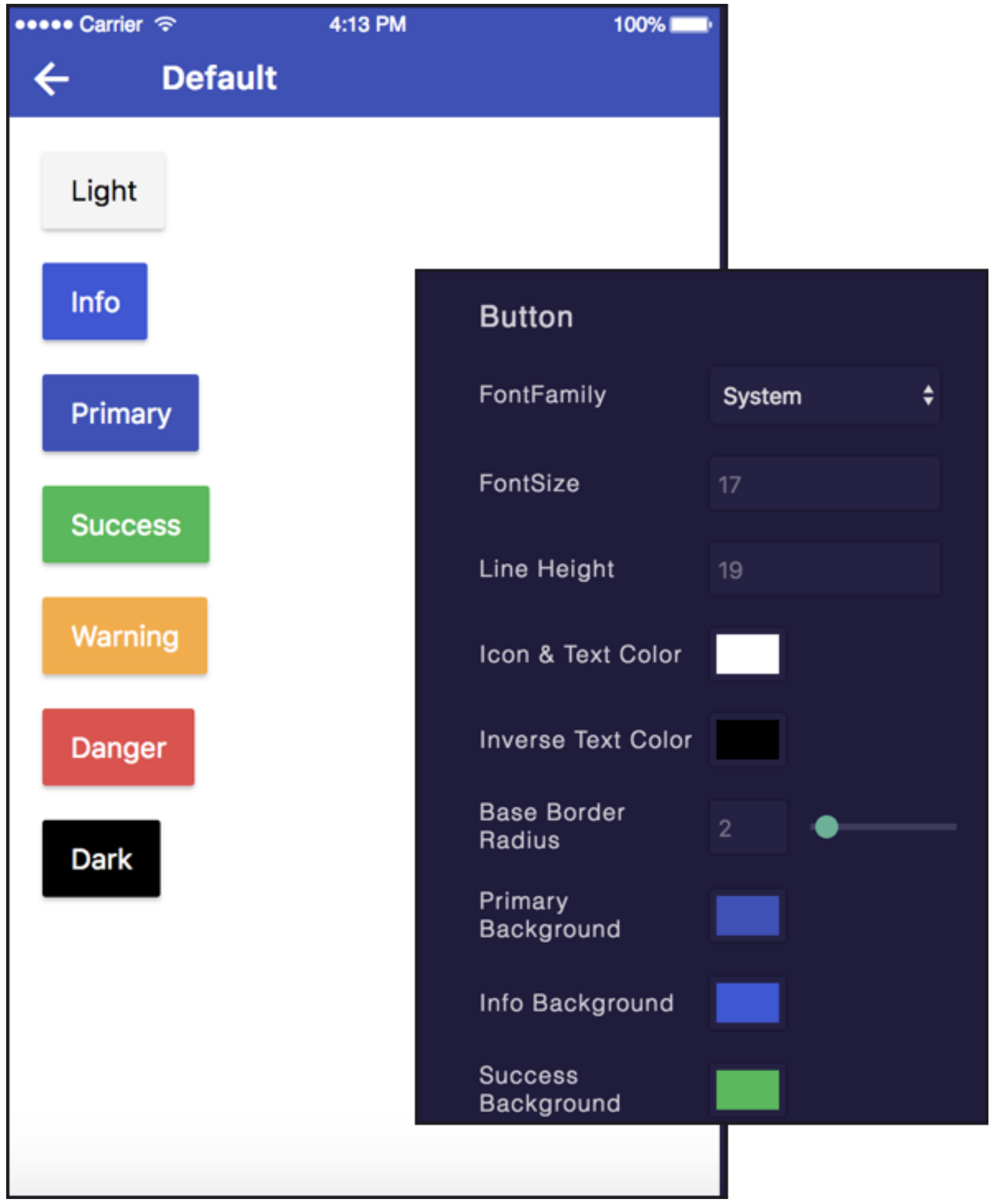
NativeBase[6] [7]

For iOS and Android.

Enable themes using `NativeBaseProvider`.

```
<NativeBaseProvider>
  <View>
      ...
  </View>
</NativeBaseProvider>
```

```
<Button variant="subtle" colorScheme="secondary">
  <Text>Contact Us</Text>
</Button>
```



Button

FontFamily System

FontSize 17

Line Height 19

Icon & Text Color

Inverse Text Color

Base Border Radius 2

Primary Background

Info Background

Success Background

[6] button example

[7] `NativeBaseProvider`

Cutomizing themes *:

```
import {extendTheme, NativeBaseProvider} from 'native-base';
```

Customizing themes using `extendTheme()`:

```
const theme = extendTheme({
    colors: {
      // Add new color
      primary: {
        50: '#E3F2F9',
        100: '#C5E4F3',
        200: '#A2D4EC',
        ...
      },
  });

<NativeBaseProvider theme={theme}>
  <Container>
    <Content>
      ...
    </Content>
  </Container>
</NativeBaseProvider>
```

* example code

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

How to Access Sensor Data

Install the sensor library:

```
expo install expo-sensors
```

Import the sensor component:

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

Check if the sensor is avialable:

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

Create listener for sensor events:

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

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

```
_subscribe = () => {
    this._subscription = Accelerometer.addListener(accelerometerData => {
        this.setState({ accelerometerData });
    });
};
```

To remove listeners for sensor events:

```
Accelerometer.removeAllListeners()
```

```
or
```

```
this._subscription.remove()
```

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

```
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.

Demos

- Accelerometer
- Step Counter

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);
}

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

example code

Assignment Preview

React Native 1 β: Prototyping

Designing/prototyping screens, navigation to support the capabilities:

— Creating a day view that shows user meals and exercises and make it the default view,

— Providing the ability to add a meal to a day and foods to meals,

— Creating a section of the day view that allows the user to compare their goals versus the current day's stats (e.g., total calories consumed),

— Developing a view that allows the user to add/edit/remove exercises to the current day.

In three parts:

1. **Part 1:** Paper Prototyping — using ... paper!
   — **Deliverable:** photos of paper prototypes
2. **Part 2:** Visual & Interaction Design — using Adobe XD
   — **Deliverable:** screenshots of static screens
3. **Part 3:** Interactive Prototyping, using Adobe XD
   — **Deliverable:** interactive prototype, video demonstration

What did we learn today?

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