- Rohan Maheshwari

3050 Mobile Development Notes

1.001 Welcome / Introduction

- This module is very interactive and is assessed using small quizzes, marked assessment and your final project.
- These elements comprise two courseworks (CW1 and CW2). ie, CW1 (30%) ; CW2 (70%)
- Check for node.js / NPM. Install it.
- Consider using Watchman

- Key aspects covered ^
- The Course syllabus

1.002 Course syllabus

Module description

Mobile technology, including smartphones and tablets, has been a significant technology platform in recent times and the mobile app ecosystem is a significant driver of both innovation and employment. Mobile development is, therefore, a critical applied area of computer science. This course will support you in getting started in mobile development, and it builds on material such as databases, networking and web development taught elsewhere in the programme.

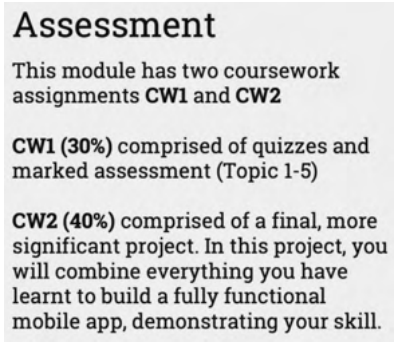
This module is designed to give you an overview of the mobile development ecosystem as well as how to design, develop and test create cross-platform mobile apps built using React Native.

Learners get to achieve the above through a heavily practical approach, using a mixture of weekly programming exercises, code demonstrations, videos and quizzes.

Module goals and objectives

After completing this module, you will be able to:

1. Design an apps user interface based on a set of requirements and goals,
2. Critically discuss UX techniques used by prominent applications,
3. Design wireframes to inform the testing and development process,
4. Replicate different design styles across devices,
5. Create accessible applications,
6. Create responsive applications that adapt to device requirements,
7. Design and create UI elements, linking them to actions,
8. Utilise navigation systems across devices,
9. Assess the use and performance of more advanced UI elements such as scroll views and table views,
10. Create precise and useful unit tests to help monitor the performance of an application,
11. Integrate API's within your application. Making authenticated requests,
12. Access and use sensors found on mobile devices, e.g. camera, accelerometer,



1.1 The app ecosystem

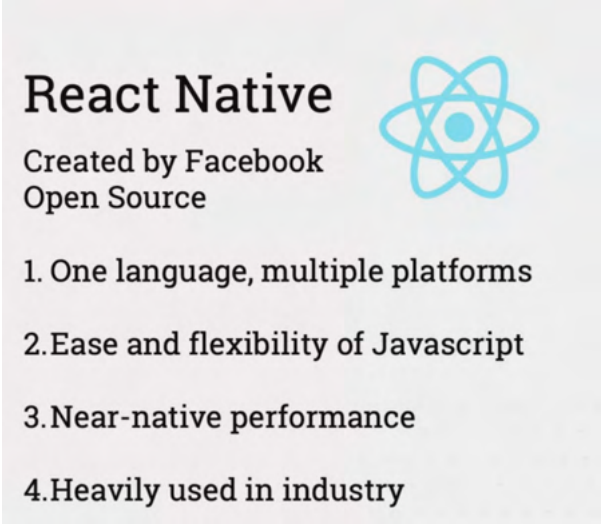
- Importance of everyday interactions with mobile apps.
- Getting started with ReactNative.
- Key aspects which need to be understood to become a mobile developer.

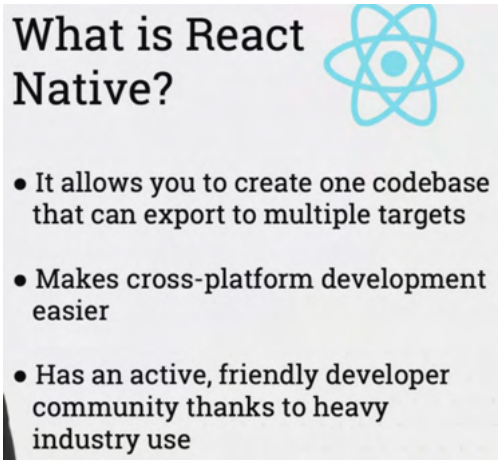
1.2 Developing for multiple platforms

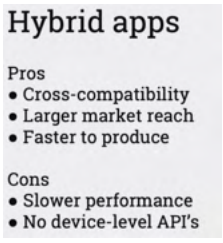
- Multiple codebases, for two key worldwide dominant mobile OS. Android has 72% while iOS has 27%.
- iOS written in Swift & Objective-C while android is written in Java & Kotlin
- Native apps & Hybrid apps; native ones are written specifically for one platform/device whereas hybrid are written in one language and are compiled for multiple devices/ platforms.
- Native apps: Made for specific Os & devices. Hybrid, work cross platform

1.3 Programming with React Native

- A framework that allows us to program one codebase in Javascript but compile and deploy apps to numerous platforms.
- Others include : Ionic, Flutter, Cordova
- Mac & Windows repos allowing us to make native desktop apps as well.







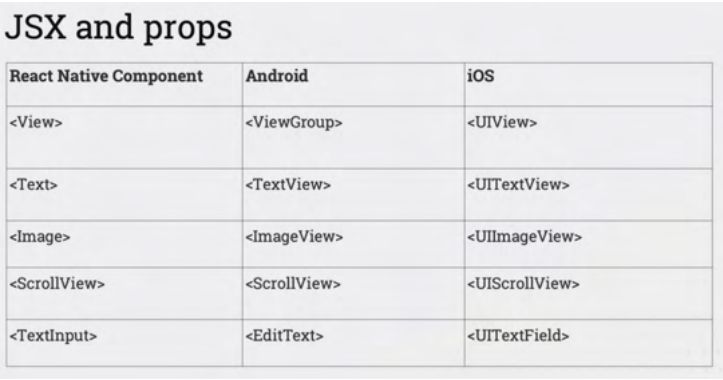
- Expo : A tool that works alongside React Native to make development easier. Allows previews for development and testing.
- Managing code signing is the process of authenticating an app before loading onto a device for testing.
- Install React Native and Expo using terminal.
- Arrow Notation: A modified way to express functions, expressed as… (a) => { *statement};
- Other variants: (a) => a*2; | a => a*2;

CLI Guide:

1. npm install -g expo-cli
2. expo init projectName
3. npm start (to begin expo)
4. Then 'i' runs the iOs simulator

1.4 ReactNative Core concepts:

- JSX(Javascript syntax extension); a key part of ReactNative, allows us to create components that appear on the screen.



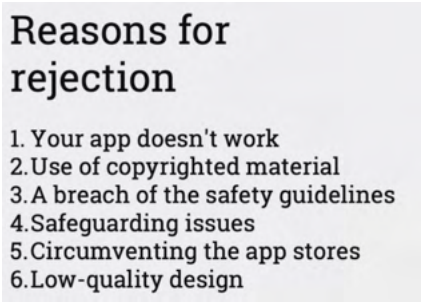
- Components must be capitalised as shown. They will translate directly into native platform code
- Props(properties). These make alterations.
- Useful: https://react.dev/learn

- Read about: https://docs.expo.dev/workflow/ios-simulator/
- Read about: https://docs.expo.dev/workflow/android-studio-emulator/

1.6 The App economy:

Guidelines to be followed:

- Apple Guidelines
- Google Guidelines



2.0 Mobile User Interface Design:

We look at the design considerations associated with creating a mobile app, including the design styles prevalent in the mobile field, choosing a colour palette and what this means for accessibility.

- There are three key design styles: Skeuomorphism (inspired by the real world), Minimalism (clean, focused designs), Neumorphism (middle ground between the two)
- The initial design were Sk. which was then followed by Minimalism. Currently, it is neumorphism which is a hybrid style, taking from the best of previous two styles.
- Colours have an impact on the mood. Refer and explore ideas around colour psychology.

2.006 Design ideology can be referred to in the following two links: Apple & Google

2.1 Dark Patterns

- Take the users prior experience of the web and subverts it to trick the user into doing something they never intended to.
- Learn more on https://www.deceptive.design

2.2 Wireframing

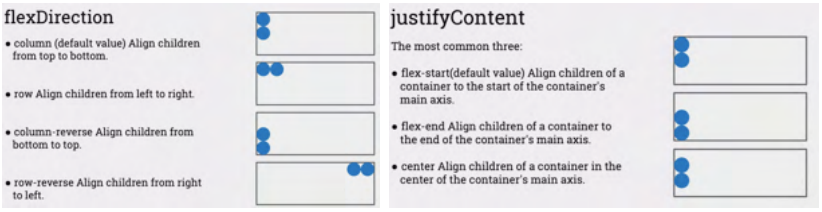
- allow you to quickly design layouts and check your designs. It sits somewhere in between rough sketches and design prototypes.
- No focus on style, colour. It is a plan for the elements.
- A mid-fidelity wireframe is one that is semi realised.
- Explore user flow diagrams as well. They describe a user uses an app and goes from one screen to another. Should also require minimum to access important areas and information. Should also be consistent.
- Wireframes allow you to create a foundation for your content.
- diagrams.net and paper used for demonstration of a user flow diagram; build them as a flow chart.
- As a practise: wireframe up an existing app.

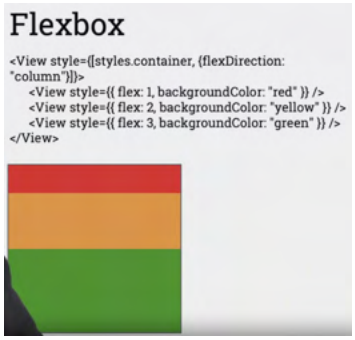
2.4 Styling in React Native

- Explored inline styling and stylesheet styling and mixed the two.
- Explored multiple different elements which are commonly used.

2.5 Responsive Design

- We don't build based on dimensions, rather proportions. ie, 100% width of device
- We use Flexbox in React Native. This defines how elements will Fill over the available space.





<- A complete guide to flexbox: linked here & here

- An apple guide to layout considerations for mobile devices.

2.6 Always design with accessibility in mind.
- Four key categories: Vision, Hearing, Physical & motor + Literacy and learning.
- Learn about the accessible, accessibilityLabel, accessibilityHint, accessibilityActions props.

3.0 Programming user interfaces

We now learn about using JSX to create basic elements on the screen, understand the need for pagination. We use components such as images, buttons and press-able elements. We learn about hooks and navigation as well.
- User interfaces are made up of components such as buttons, images & switches.
- A variety of UI elements are highlighted such as <View>, <ActivityIndicator>, <Button>, <Image>, <Text>…
- <View> Can be considered a sheet of paper. Elements within it get grouped together and can often be modified together. Use colours and borders to add elements and remove these later on. Elements added within are called child components. Views are often parent components.</View>
- Reading list for components is here

3.103 Hooks are important to modern react development. Hooks are functions that let you "hook into" React state and lifecycle features from function components. Hooks are used to add memory and start to our applications. This allows us to alter the state and change elements on the screen.

- Only call hooks at the top level. Don't call them inside loops, conditionals or nested functions.

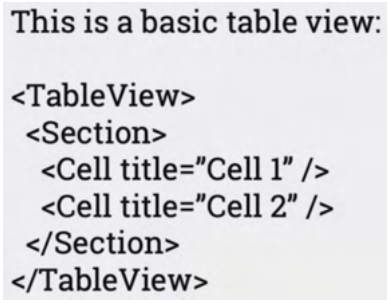
- useState and useRef are pre built hooks. There are many more, learn about them here.

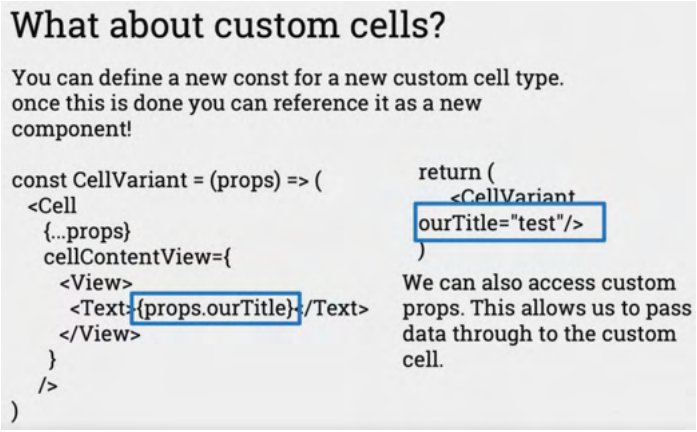
3.301 Navigation and Pages

- Navigation and pages separate our content into easily understandable and digestible chunks of information. Present just the needed information. Make use of storyboards and user flow diagrams.
- Install all the latest react navigation libraries. Learn more here

4.1 Table and Grid Views

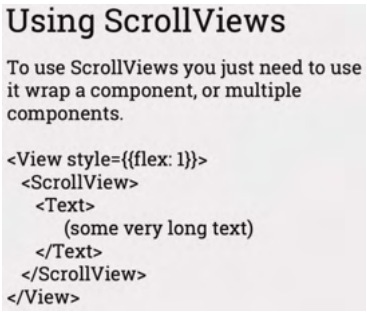
- Table Views are a key building block of mobile apps and are used extensively.
- We will be using react-native-tableview-simple by Patrick Puritscher
- Table views are comprised of sections and cells. You can consider a section to be a grouping of cells.
- Read more on - GitHub.com/Purii/react-native-tableview-simple
- One can also define custom cell - which an excellent way of creating a template for repeatable content.
- As such, Table views are most likely found within most apps. They are extremely flexible and can be customised extensively. They build upon design styles which help improve ease of use.

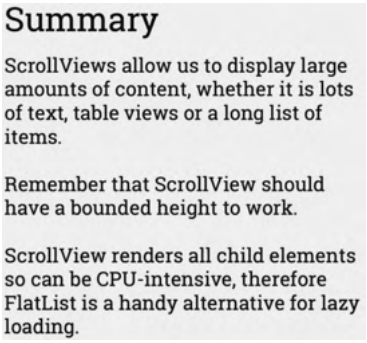




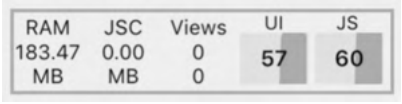
4.2 ScrollViews & performance

- Allow us to otherwise display content that would not otherwise usually fit into the screen. ScrollViews must have a bounded height to work, usually 100%
- ScrollViews often impact performance. It is also important to remember that they load/render all child elements at the same time even if not viewed.
- FlatList is the new alternative to this problem. This loads items lazily, ie, items are only loaded prior to the view and they are unloaded once they are no longer in View after a while as well.
- React Native offers near native performance. Performance monitoring is essential to create a smooth app. We monitor this using frames. However, sometimes you may spot issues such as “Jumpy” or glitchy Scroll View. These can be solved.
- It is all about Frames. Eg, of film is given. Frames are usually displayed at 30fps. However, modern devices such as iOS display at 60fps. If it cant do that in time, it will drop frames and make the animation look jumpy. Dropped frames occur when the device is under load and cannot process fast enough.





- When building in debug mode, you can shake the device to display the performance monitor. Eventually get rid of dev mode and all console.logs as they eat away at resources.



- Apps should run smoother in production mode vis a vis developer mode as they carry more overheads in dev mode.

- Javascript(JS) thread is where majority of react Native code is executed. If it takes longer than frame rate to execute, it will be a dropped frame. This renders a glitch animation.

- UI main thread executes code natively outside of React Native. We rarely use this. However, some libraries such as NavigatorIOS use real UI elements.

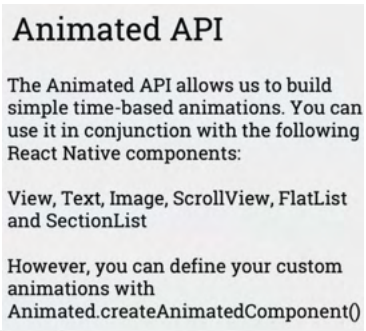
4.4 Animations

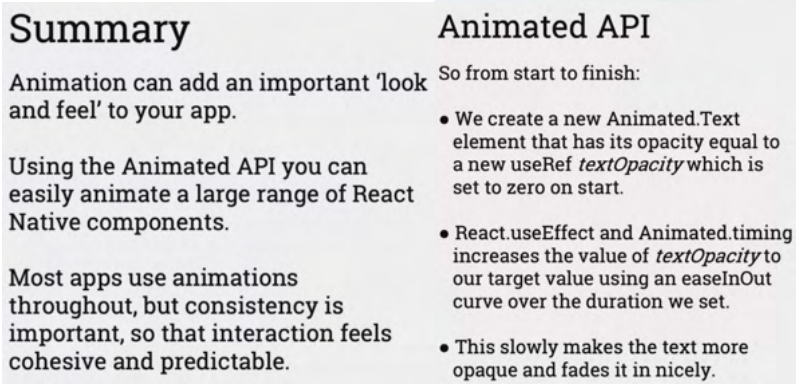
- A key part of mobile applications, they provide feedback to actions performed by users and they also improve the look and feel.

- There are many system animations as well. Such as, app launch effects, “elastic banding” and sleep/wake animations.

- However, we can also make our own animations.

- Animated API can be used in such instances.
- Read more here



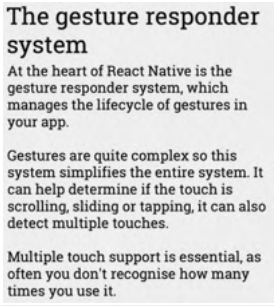


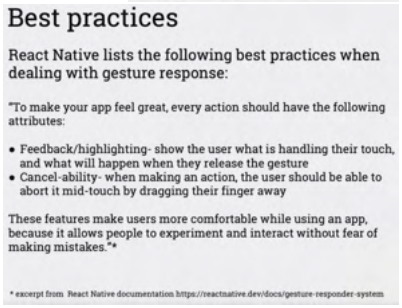
Gestures:

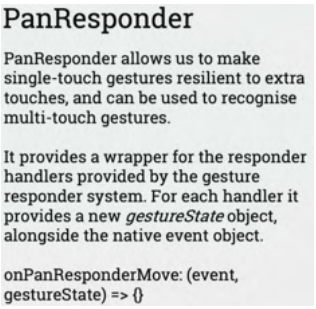
-An integral part of the mobile experience. They are the reason why an app feels smooth and fluid.

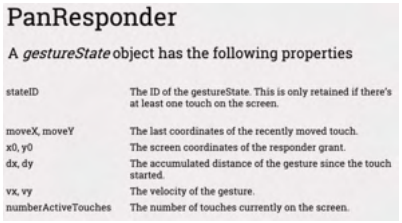
-At the heart is the gesture responder system which manages the lifecycle of gestures within your app.

Gestures are complex and this system simplifies them.



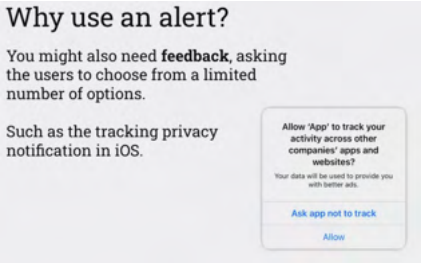






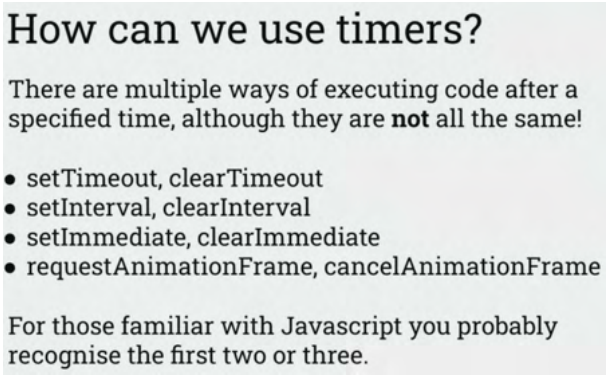
4.6 Alerts

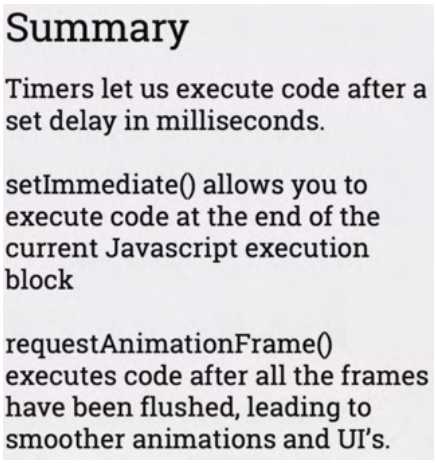
- Alerts are a common component of mobile applications. They are used to deliver critical information to the user, or provide immediate feedback.
- You can customise the title, message and buttons shown within alerts. There are small changes in options available basis the operating system.



4.7 Timers

- Timers let us execute code after a set delay and are very important for mobile development. Read more here



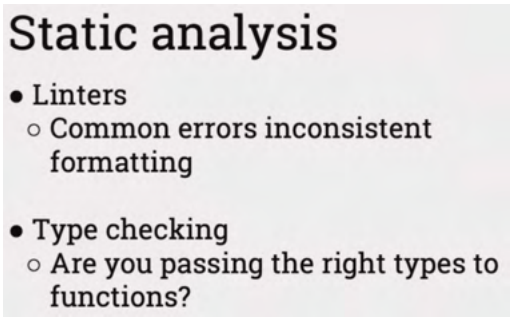


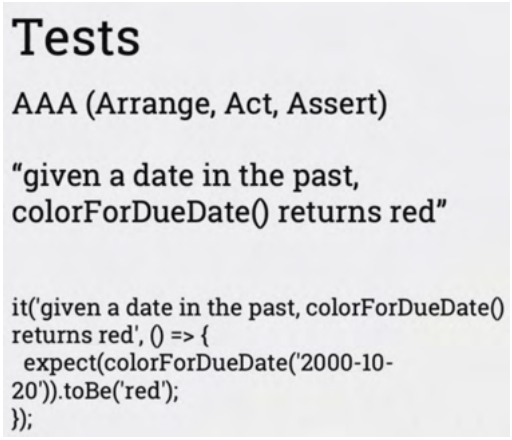
5.0 Efficient programming

- Write clean, understandable code for long term benefits.
- Make sure to comment your code. This is key.
- Write modular code. Only.
- Write them in multiple files. Import them into app.js
- console.log() should be used, then removed. Use a common naming convention, such as camelCasing.

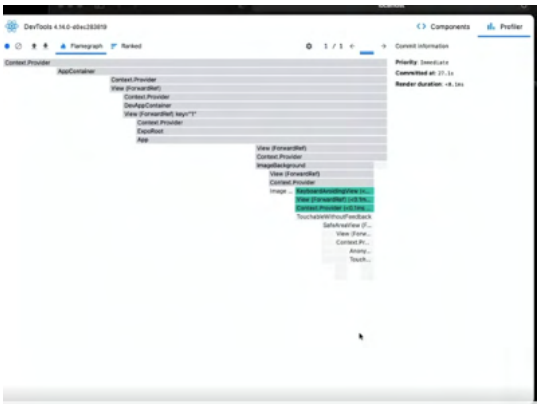
5.1 Unit Testing and Debugging

- Testing is particularly important as your applications code base grows as small changes might cascade and impact your code negatively in all sorts of edge cases. Automated testing can be used to counter this. There are two types of testing, Static and dynamic.
- Large code-bases are particularly susceptible to errors as multiple developers would work on them and they would not be familiar with all the intricate details.
- Modular code is essential as it can be tested. For instance, think of modular code as a lego bricks. If a part broke it can be replaced. If your code is not made as such, and is instead a large unified brick. A small break may render the whole brick unusable.
- Unit tests test the smallest parts of your code. So, individual functions or classes.
- Mocking allows us to override variable dependencies and provide a constant state.
- Integration testing implements testing functions that should work in parallel with each other.
- Component testing: Looking at our UI, does it interact and work correctly?





5.1 (cont.) Advanced Debugging



- This is apart from console.log() and alerts.
- Npm install -g react-devtools can be launched with react-devtools once installed via the terminal (We are interested in React-Native)
- Eventually we will need to go over/compile the app in production mode which will get rid of the dev tools. This will give a top down view of all the <components>.
- <profiler> Can be used to debug glitchy, slow apps. What specific elements are under stress can be clarified from here, after clicking the record button.

5.2 Functional components

- A component that we created using a function. Should be used over class components.
- Functional component simplify the development process vis a vis class components. Has an impact with state management. Learn more here

Functional components

```
function FunctionalComponent() {
    return (
        <View>
            <Text>Testing 123</Text>
        </View>
    );
}
```

```
class ClassComponent extends Component {
   render() {
    return (
        <View>
            <Text>Testing 123</Text>
        </View>
    );
   }
```

5.3 Advanced Javascript - Syntax Transformers

- So far we've been using relatively centered JavaScript language with some small ES6 Additions signing such as array syntax or destructuring. But, as React Native ships with the Babel JavaScript compiler, we actually have a lot more options, including ES6, ES8, and even stage 3 features. When I'm talking about ES6 or ES8, what I technically mean is ECMAScript.
- Block Scoping - var vis a vis let. Var uses function scope and let uses block scope along with const. Use let exclusively. Read.

for of

```
let grades = [23, 43, 77, 56];

for (let i=0;  i<grades.length; i++) {
    console.log(grades[i]);
}

// 23
// 43
// 77
// 56
```

Destructuring

```
let person = {
    name:"Joe",
    location:"London",
    device:"iPhone"
}

let {name, device} = person;

console.log(name); // Joe
console.log(device); // iPhone
```

App Performance

- Expensive operations are ones which consume a lot of resources. Also, we should keep in mind that in React Native we have one JS thread. This is a significant disadvantage.

- However, you can use interaction manager to make sure any long running work is scheduled to start after any interactions/ animations have been completed. Interactions are active touches.

InteractionManager

```
InteractionManager.runAfterInteractions(() => {
  // a synchronous task

});
```

- requestAnimationFrame():
- setImmediate/setTimeout/setInterval()
- runAfterInteractions():

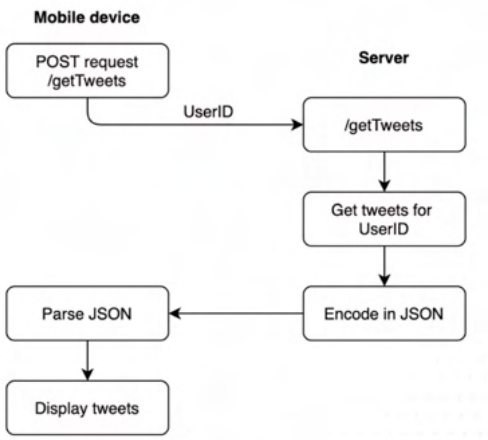
6.0 Data Sources & Ethics

- Now, we will learn how to handle and manipulate data. We will delve into data ethics and understand various data sources.

- Data sources are a significant portion of apps.

- API - Application programming interface is a piece of software that allows two applications to talk to each other.

- JSON is a way of encoding data in text sources.



POST and GET

POST
- Used to write/modify data

- You can attach parameters e.g user identifiers, customisations...

GET
- Used for viewing information
- Can have parameters e.g. https://example.com/?id=testid

Ethics of Data collection

- Don't collect data unless its essential. Treat customer data as you would your own.
- HTTPS and Strong SSL Certs are bade minimum. On device encryption should also be considered.
- Learn about API security here

6.1 JSON and XML

- JSON and XML are the most popular for encoding data apart from CSV. These are optimised to be parsable, which means they are to easily converted into associated objects within our own programming language.

JSON

```
{
    "name":"Joe",
  "skills":["running", "coffee", "code"],
  "programming":{
    "preferred_language":"Javascript",
    "preferred_IDE":"Atom"
  }
}
```

- JSON [Javascript object notation] ; Often used for transmitting data between servers and mobile apps. It is a lightweight data interchange written in plain text.
- XML has a different syntax, similar to html. Stands for Extensible Markup Language. Everything is encapsulated in tags.
- Loading JSON from a file is fairly straightforward. Add a .json file and import it within the app to reference it in the code by working your way down the file tree.

JSON

```
let meString =
"{"name":"Joe","skills":["running","coffee"
,"code"],"programming":{"preferred_lang
uage":"Javascript","preferred_IDE":"Ato
m"}}"

let parsedString =
JSON.parse(meString);
let myName = parsedString.name;
let secondSkill = parsedString.skills[1]

let newJSONString =
JSON.stringify(object);
```

XML

```
<root>
  <name>Joe</name>
  <programming>
    <preferred_IDE>Atom</preferred_IDE>
    <preferred_language>Javascript</preferred_language>
  </programming>
  <skills>
    <element>running</element>
    <element>coffee</element>
    <element>code</element>
  </skills>
</root>
```

6.2 Saving Data

- Memory is important within a mobile app. Whether its remembering a user who is logged in to perhaps saving data that has been inputted by the user.
- There are two libraries for this, namely, AsyncStorage & SecureStorage.
- Read the documents here, here and here.
- Learn about javascript promises.
- Saving data can reduce cognitive overload for users.

XML

```
npm i fast-xml-parser
...

import { parse } from 'fast-xml-parser';

let parsedString = parse(textResponse);
let myName = parsedString.root.name;
let secondSkill =
parsedString.root.skills[1]
```

7 Introduction to cloud services

- A wide range of services that are delivered on demand over the internet. They provide easy access to applications and resources without the need for internal infrastructure or hardware. Removes the need for us to personally host, difficult or just cumbersome application on our own infrastructure.

- Gives ability to scale and pay as you go and as you use. Can be the cheaper option and also allows for flexibility.

- SaaS anything that provides a lot of different digital services. IaaS provides access to hardware, application firewalls. PaaS platform as a service allows developers to build apps within a web interface w/t a need for databases, Os's as they are all provided by the service provider.

Types of cloud

"A public cloud is like renting an apartment, while a private cloud is like renting a similarly sized house. The house is more private, but it also typically costs more to rent, and it's not the most efficient use of resources. Maintenance in the apartment is handled by the building supervisor, but it's harder to get a contractor out to fix the house (sometimes, the tenant may have to do it themselves)."

(Cloudflare, 2021)

- A key guide to the cloud is here. Read it.

Reachability

- Reachability Detection - this is a key implementation.
- Use: expo install expo-network
- Use: import * as Network from ‘expo-network’;
- Can be used for notifications etc. Read all about it here
- Use: expo install @react-native-community/netinfo

Network connection

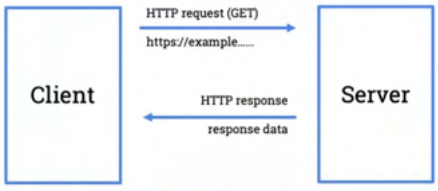
```
await Network.getNetworkStateAsync();


{
  type: NetworkStateType.CELLULAR,
  isConnected: true,
  isInternetReachable: true,
}
```

Dealing with unstable internet connections

- Network library allows us to get network configuration details. So we can determine whether or not they are connected to a specific type of internet connection. Ie, wifi, cellular etc.
- When using await use a try-catch block.
- Event listeners listen in the background for any particular event happening.
- Netinfo is similar but generalised to determining network data.
- NetInfo.addEventListener(state => { })
- SSID - Adding an entitlements request, can be accessed for identification etc
- Learn about expo documentation here

7.2 HTTP GET ; Retrieving data from the internet

- GET provides the most useful and basic way of getting information from a remote server. You can imagine get requests to be as simple as shouting to a server. “Hey server, I want the contents of whatever is at URL”.



fetch API

```
fetch("https://archive.org/metadata/principleofrelat00eins")
.then((response) => response.json())
.then((json) => {
    console.log(json);
})
.catch((err) => {
    console.log(`An error: ${err}`);
})
```

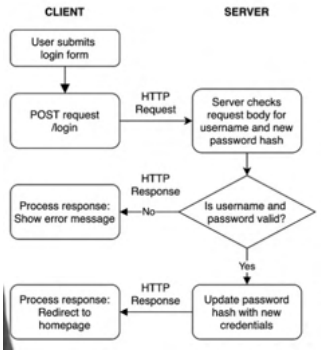
7.3 HTTP POST ; Changing information over the internet

- HTTP POST is similar to the GET, together they form the very foundations of the internet.
- GET is used for viewing information, POST is used for changing information. POST Example, changing one’s password over a web application.
- 7.303 is an important video which demonstrates

fetch API

```
fetch('https://example.com/something',
{
 method: 'POST',
 headers: {
  Accept: 'application/json',
  'Content-Type': 'application/json'
 },
 body: JSON.stringify({
  param1: 'value',
  param2: 'value'
 })
});
```

POST requests



how to setup and query a node.js server with express.js

7.4 What is authentication?

- We use authentication for multiple reasons, such as limiting access to specific APIs, securing sensitive URL endpoints or perhaps making request auditable. This allows access to be linked back to a specific user.
- There are 4 typical authentication methods available to us: Auth, Bearer Token, API Key and OAuth.
- Be sure to seek out and refer to appropriate documentation for the above.
- OAuth allows integration with third part authentication providers, eg, Facebook and twitter. This is often a complex setup but docs are available.

- Basic Auth

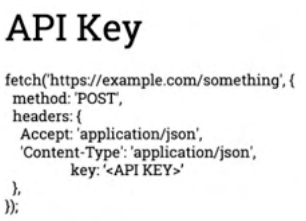
Authorization: Basic <credentials>

- Bearer token

Authorization: Bearer <token>

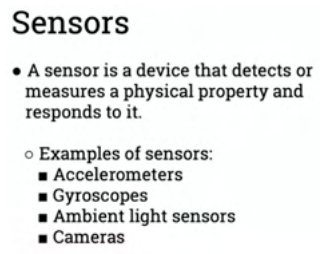
API Key

- key: <token>

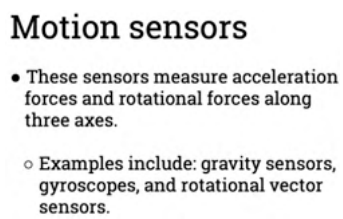


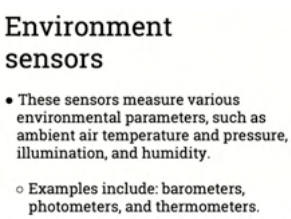
- Implementation of Basic Auth is demonstrated. Have a look.
- Implement: npm i express-basic-auth. Find and read the docs.
- Implement: npm i react-native-base64. Find and read the docs.

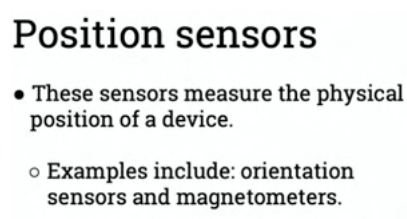
8.0 Sensors



- A sensor is a device that detects or measures a physical property and responds to it.
- FInd and read the android documentation describing and discussing the various types of sensors.
- Often the best use of sensors comes from adding quick shortcuts which improve overall accessibility.
- Consider backup sensors as well.



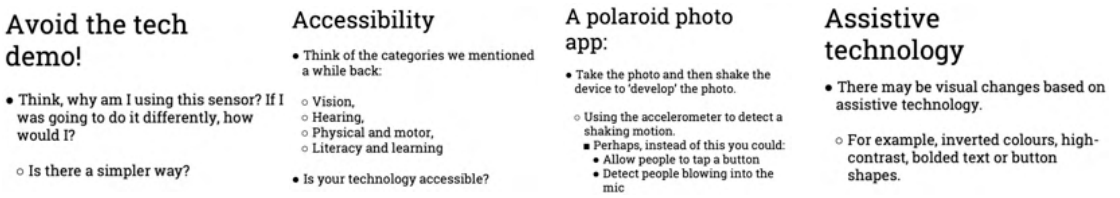




- LiDAR : Is a method for determining ranges (Ex, how far an object is) by targeting an object with a laser and measuring the time for the reflected light to return to the receiver.

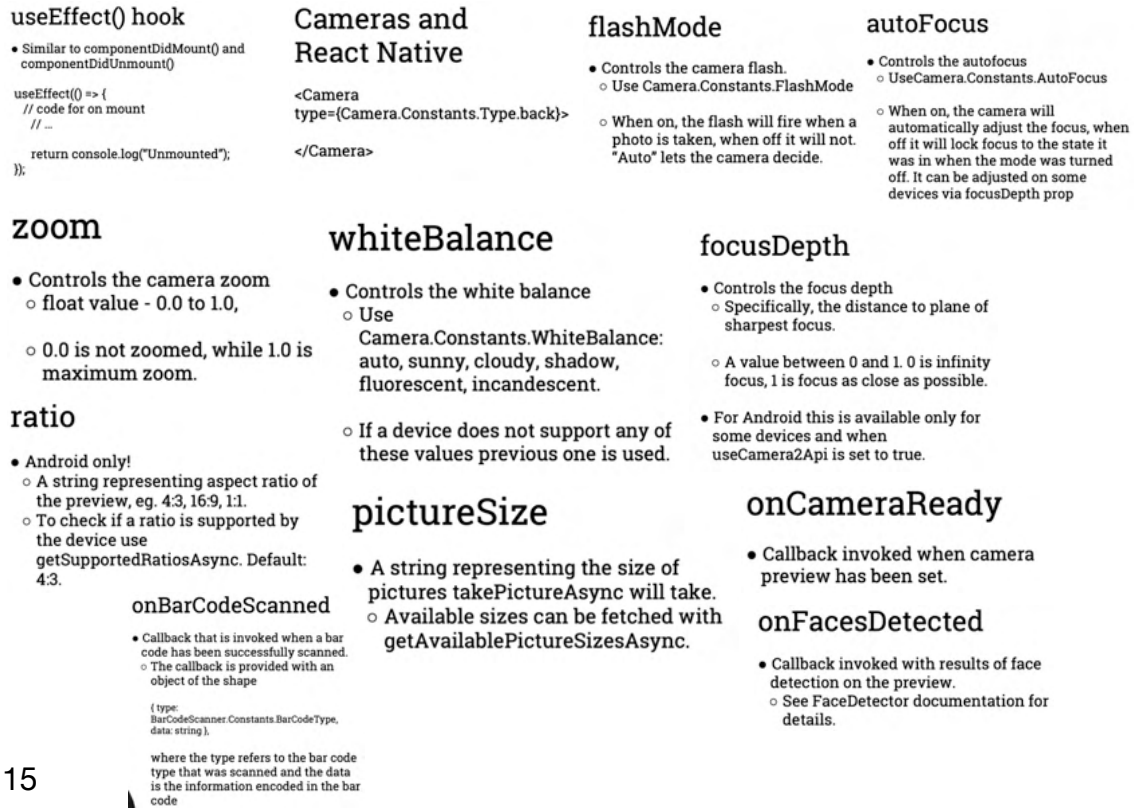
8.3 How to best use sensors?

- Always ask yourself, how can I do this without a sensor? Is there a simpler way? Is it accessible?
- Expect all types of devices to use your technology. Account for this in the development of your application.
- Dev toolkits can virtualise these various different elements so that you can test your app under different use cases and circumstances.
- Consider a public beta test as well.
- Different Components example: ‘P1’ New Android phone with new gyro with higher precision versus a ‘P2’ Older IPhone with old gyro and low precision



8.1 Mobile Cameras:

- Photos are everywhere: 1.2 trillion photos in 2017. With 82% coming from mobile devices.
- Very high market interest.
- Cant use cameras on simulators
- Remember to always ask for permission to access the camera.
- React Native greatly simplifies the process for us. First, install the library expo install expo-camera. Ask for permission to access the camera Camera.requestPermissionsAsync() within useEffect().
- Again, seek and read up on the dev docs.



8.2 Haptics and Vibrations

- They provide immediate physical feedback. Build a clear, causal relationship between each haptic and its trigger.

- Ideally, people always know why your app plays a haptic pattern. If this cause effect relationship is not maintained it can be confusing to the user. Use it in a complementary capacity. Also use it judiciously.

- In general, avoid significant haptic feedback. Make it optional and don't make it an integral part of your app.

- Use: expo install expo-haptics

- Read the developer document here

expo-haptics

```
import * as Haptics from 'expo-haptics';

Haptics.impactAsync(style);

// has the following options
Haptics.ImpactFeedbackStyle.{Light, Medium, Heavy}

Haptics.notificationAsync(type)

// has the following options
Haptics.NotificationFeedbackType.{Success, Warning, Error}
```

8.3 GPS

- Stands for global positioning system.
- Satellites act like stars in the constellation. NASA says we know where they are at any times so we can make calculations.
- Formed of 30+ satellites circling the Earth.
- Requires a lot of power. This is a big downside. Requires consent as well.
- iOS lets you work with approx. Locations.
- Use: expo-location
- Get the users current position.
- Location.getCurrentPositionAsync({});
- Location.getLastKnownPositionAsync({});
- Location.geocodeAsync(address)
- Location.reverseGeocodeAsync(location)
- Location.startLocationUpdatesAsync(taskName, options)
- Location.startGeofencingAsync(taskName, regions)
- Read the notable functions on dev docs. Here.

GPS

Global Positioning System

- 30+ satellites circling Earth
  - Our phones detect signals from orbiting satellites
  - Calculate our position based on the response time of at least 4 satellites

expo-location

```
useEffect(() => {
  (async () => {
    let { status } = await Location.requestForegroundPermissionsAsync();
    if (status !== 'granted') {
      console.log("Permission was denied!");
      return;
    }

    let location = await Location.getCurrentPositionAsync({});
    console.log(`Their location is ${location}`);
  })();
}, []);
```

8.4 Accelerometers

- A sensor that measures the acceleration forces acting on an object.
- It monitors the rate of change in the velocity, velocity being the displacement of the object divided by the change in time.
- Often used with a gyroscope.
- Use: expo install expo-sensors

expo-sensors, accelerometer

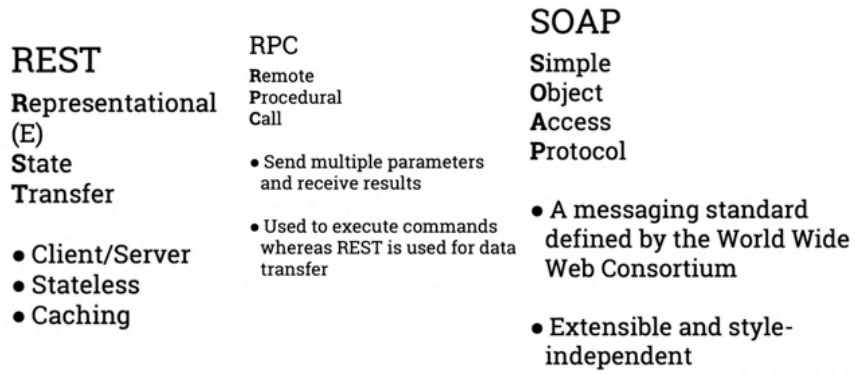
```
const [subscription, setSubscription] = useState(null);
const _subscribe = () => {
  setSubscription(
    Accelerometer.addListener(accelerometerData => {
      console.log(accelerometerData);
    })
  );
};

const _unsubscribe = () => {
  subscription && subscription.remove();
  setSubscription(null);
};

useEffect(() => {
  _subscribe();
  return () => _unsubscribe();
}, []);
```

accelerometerData contains our x, y, z values

_unsubscribe() is called when the component unmounts

9.0 Introduction to APIs

- APIs are a type of software interface, offering a service to other software services. Basically, letting computers communicate with each other.They provide us with a fantastic set of functionality and are often the backbone of mobile apps.
- API Jargon worth knowing:



9.1 Push Notifications

- Push notifications are mobile elements that let users know about something that may interest them. They are proven to help boost user interaction.

- Avoid sending multiple notifications for the same thing. Respect user privacy. Remember that users can and will mute/cancel your notifications if not followed.

- On most platforms you can also send badges which are small notices with numbers on apps. Badges are a number to indicate something, such as missed messages etc.
- Be sure to read the developer guidance for push notifications here

Get permission

```
useEffect(() => {

  // register for the push notifications
 token = (await Notifications.getExpoPushTokenAsync()).data;
  console.log(token);

}, []);
```

Make POST request to expo

```
fetch('https://exp.host/--/api/v2/push/send', {
  method: 'POST',
  headers: {
    Accept: 'application/json',
    'Accept-encoding': 'gzip, deflate',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(
    {
      to: expoPushToken,
      sound: 'default',
      title: 'Notification title',
      body: 'Notification body',
      data: { },
    }),
```

Typical flow

1. Ask for permission,
2. Save the 'token' server-side,
3. Make post request to expo, containing the notification "payload".

10 Deployment

- Learn and follow app store guidelines so that your app meets those guidelines.

- You will have to buy developer licenses before we can submit an app to the app store.

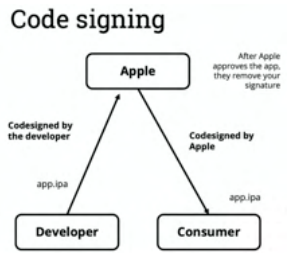
- Apple's Guidelines
- Google's Guidelines

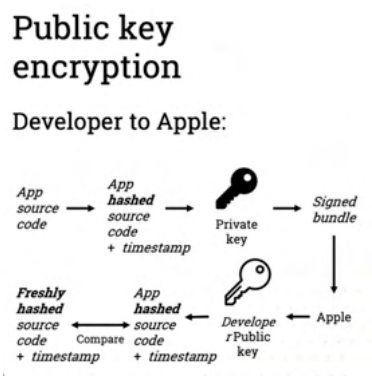
10.1 Code Signing

- This is an important part of creating a distributable app.

- Uses public key encryption; but is the inverse process of the usual public key encryption. Never expose your private key.

- Code signing documentation can be found here and here





10.2 Exporting a build

- Expo builds app bundles in the cloud. PaaS is used to build the app.

- Run either: expo build:android or expo build:is
- The above will also run expo publish!

- Let expo deal with code signing for android

- For iOS you will need to share the developer credentials.

- Once done, you will get a link to your downloadable bundle. Upload this to the play store for android

- Read the expo developer guidelines: Here

- The newer version is here

app.json

```
{
  "expo": {
   "name": "Your App Name",
   "icon": "./path/to/your/app-icon.png",
   "version": "1.0.0",
   "slug": "your-app-slug",
   "ios": {
    "bundleIdentifier": "com.yourcompany.yourappname",
    "buildNumber": "1.0.0"
   },
   "android": {
    "package": "com.yourcompany.yourappname",
    "versionCode": 1
   }
  }
}
```