## React Native

* Framework for building apps using JavaScript/TypeScript for multiple platfoms - Android/Web/IOS
* Uses Components/States and hooks

## Expo

* Two ways to build React Native apps

1. Expo CLI

* Set of tools and frameworks that sit on top of React Native and hide complexity from us. Makes it fast and easy to build the apps
* install globally by running the code npm install -g expo on cmd

1. React Native CLI

* Needs prior knowledge with Android and IOS programming which you only have Android atp.

## Setting Up Development Environment

* install node.js
* install expocli by running the following command on CMD
  + npm i -g expo-cli
* install Expo Client on the phone
* make sure the node installed is 20+
* Code editor should be VSCode
* Inside VS Code
  + Search for react native under extensions:
  + install "React Native Tools" - for code debugging
  + install "React Native/React/Redux Snippet" - generates code using shortcuts
  + Install "Prettier" for code formatting
  + Install "Material Icon Theme" - gives the project files icons
  + In setting search for "formatonsave" and check it

## Creating the App

* ***npx create-expo-app*** ***NameOfTheApp*** - comes with extra boiler plate code
* code . to open the project in VS Code
* ***npx create-expo-app@latest NameoftheAp***p --template blank
* delete the default index and app files
* create an app folder inside the root folder which is where all the components will be
* create an index.jsx in the app folder – represents the home page
* install expo router - ***npx expo install expo-router react-native-safe-area-context react-native-screens expo-linking expo-constants expo-status-bar***
* update the package.json file main to "main": "expo-router/entry", -> makes the index.jsx under apps the starting point
* update the app.json file and add “scheme”: “namepoftheapp”

## Errors

* *Powershell Script Error fix*
* *Open Powershel and paste the following command:****Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned***
* *When running the project on VSCode and the error about no web dependencies installed just install them using the following:* ***npx expo install react-dom react-native-web @expo/metro-runtime***

## Metro Bundler

* responsible for compiling all the JS files into a single file

## Building the App

build instructions: <https://docs.expo.dev/build/setup/>

## Building as a website

* ***npx expo export***
* go to vercel
* connect GitHub
* import the repository
* ***npx expo export --platform web***

## Starting the app

* ***npx expo start***

## Logging

* For dubugging.
  + use ***console.log("");*** - use this only during development and testing since it can create a lot of negative impacts on the app perfomance in production

## Changing the Icon and Splash Screen

* navigate to the app.json file
  + Locate the "Icon" attribute and change the icon directory to one of your choice
  + Locate the "splash" attribute and also change the directory as you wish

## Authentication and Database

* Postgres
* App write

## Basic Components

* React apps are built using components
  + Full page screen components that cover the screen
  + Custom components - reusable UI components built by a programmer e.g. ThemedView
  + Native components like Text and View that react native provides
* Components that we make in React native are just functions that return a UI template and should be exported at the end in order to be referenced on another page

*const ComponentName = () => {*

*return ()*

*}*

*export default ComponentName*

* + this is equivalent to a body tag in html and a composable in kotlin
* React Native uses Native components like Text and View because Native devices like Android and IOS don’t render components using HTML but use their platform specific elements so the built in native components in React Native represent and are translated to those required Native elements in Android and IOS

1. View Component
   1. Used to group other native components and provide structure and layout. Can also be used to make Cards
   2. Uses Rows and Columns to arrange and align components
   * flex: 1;
   * flexDirection: 'column'
2. Text Component
   1. For rendering text
   2. You can have as much styling reference on one text component as follows (applies to other components too):
      1. *<Text style = {[ styles.title, { margin: 5 } ]}> </Text>*
         1. Basically separate them with a comma
3. Image Component
   1. To get/reference an image from the asset folder
      1. *<Image source = { require(“…/assets/img.jpeg”) } />*
4. List/Picker Component
   1. npm install @react-native-picker/picker

*import {Picker} from '@react-native-picker/picker'*

*<Picker style = {styles.picker}>*

*<Picker.Item label='Tea' value='tea'/>*

*<Picker.Item label='Coffee' value='coffee'/>*

*<Picker.Item label='MilkShake' value='milkshake'/>*

*</Picker>*

1. Scroll View
2. FlatList

## File Base Navigation.

* Each file you make represents a different page in the app
* Navigating to index.jsx -> “/”
* Navigating from one page to another
  + Use the Link component: ***<Link href={ "/about" }>About</Link>***

## Layouts and Stack

* Layout file: component and a template that wraps all of our pages. You can use to render UI that should appear by default on all your pages
  + Create a new file called \_layout.jsx inside your “*app*” folder
  + Name the layout RootLayout
  + When there’s no layout file, the app will render the index.jsx but if you do it’ll render the layout file
* Slot navigation component
  + Wraps the content of all your pages inside it. Basically in the layout it represents your pages ***<Slot/>***
* Stack navigation Component
  + Adds the back button. Works similar to Slot
  + It basically stacks your pages on top of each other as you navigate through them

<Stack screenOptions={{

headerStyle: {backgroundColor: '#ff5555ff'}, //color of the header

headerTintColor: '#6c82ffff' //color of the text

}}>

<Stack.Screen name = "index" options={{

title: 'Home', headerShown: false

}}/>

<Stack.Screen name = "about" options={{

title: 'About'

}}/>

<Stack.Screen name = "contact" options={{

title: 'Contact'

}}/>

</Stack>

## StyleSheet

import {StyleSheet} from 'react-native'

const styles = StyleSheet.create({

text: {},

container: {},

image: {},

... })

## Themes. Dark and Light Mode

* Use the useColorScheme hook provided by react native
* Create a new varible called colorScheme inside your layout: ***const colorScheme = useColorScheme();*** -> should be imported
* Create a new folder called constants
  + Create a new file called Colors.js to store all the theme colors
  + Add the colors you wish to use: below is a template:

***export const Colors = {***

***//constant colors regarless of theme***

***primary: '#fa8e00ff', // Vibrant orange – good for action buttons***

***warning: '#8f0000ff', // Strong warning red***

***accent: '#00bcd4', // Cyan – modern and techy***

***success: '#22c55e', // Green – if you want success indicators***

***dark: {***

***text: '#e0e0e0', // Soft white for text***

***title: '#ffffff', // Bright white for headings***

***background: '#0d1117', // Deep slate gray (GitHub dark)***

***navBackground: '#161b22', // Slightly lighter gray for navs***

***uiBackground: '#1f2937' // Neutral dark blue-gray for UI elements***

***},***

***light: {***

***text: '#1f2937', // Dark gray for good contrast***

***title: '#111827', // Even darker for headings***

***background: '#f9fafb', // Light grayish background***

***navBackground: '#ffffff', // Pure white for navs***

***uiBackground: '#e5e7eb' // Soft gray for cards, inputs, etc.***

***}***

***}***

* + Create a constant variable called theme: ***const theme = Colors[colorScheme] ?? Colors.light***
  + You can ow use the theme to edit the header’s color and bg color
  + You can also copy the theme and colorScheme functionality and use it in other pages
* To make the status bar change between light and dark themes:
  + Add a fragment ( ***<></>*** ) that will wrap your Stack
  + Add <***StatusBar value = “auto”/*** > above the Stack

## Fonts

* navigate to the layout file and create a constant named loaded:

***const [loaded] = useFonts({***

***FontName: require(“../assets/fonts/fontName”),***

***…})***

## Reusable Themed Components. Themed UI

* Encapsulates Styling and theming into reusable UI Components
  + Make a new folder in the root folder called ***Components***
  + Create a new file for your Theme e.g. ThemedView.jsx
  + Rnfe
  + Remove the unwanted components
  + Add the colorScheme and theme constants
  + Add styling to your component***. style = {[ ]}*** -> [] represents an array
  + Example below:

***import { useColorScheme, View } from 'react-native'***

***import { Colors } from '../constants/Colors'***

***const ThemedView = ({style, ...props}) => {***

***const colorScheme = useColorScheme()***

***const theme = Colors[colorScheme] ?? Colors.light***

***return (***

***<View style = {[{ backgroundColor: theme.background}, style]}***

***{...props}***

***/>***

***)***

***}***

***export default ThemedView***

## States

* Allows for user input

***Import {React} from ‘react’***

***const [varName, onVarChange] = React.useState(‘ ’)***

* Inside the Text input (***import TextInput from ‘react-natve’***) pass the ***varName*** variable by adding ***value = {varName}.*** Also add ***onChangeText = { onVarChange }***
* In your button parameters add ***onPress = { () => { logic }}***
* If you wish to return a Text or view after the logic is done:
  + Create a variable to store the logic outcome
    - const [logicOutcome, setOutcome] = React.useState(‘ ’)
  + { logicOutcome !== ‘’ && <Text> {logicOutcome} </Text>}

## Colors and Themes

* Use React Native Paper Themes
  + Watch: <https://www.youtube.com/watch?v=JkepeUIrwUs>

## Setting up custom dev client

* npm install -g eas-cli
* eas login -> log in to expo
* eas build:configure
* eas build --profile development --platform android -> building the app for android platform
* npx expo start --dev-client -> starting the server
* eas build:list -> check the list of builds

## Connecting MongoDB

* NodeJS – Environment that allows us to create a server
* Make sure you have 2 files backend and frontend
* ***npm init***
* ***npm i express nodemon***
* create a file inside backend called ***server.js***

***const express = require("express"); //initialise express***

***const app = express();***

***//assigning a pool***

***app.listen(5000, () => {***

***console.log("Server Has Started") //message that displays when the server is started***

***})***

* Open package.json and change test to start: “node server”
* Use ***nodemon server*** in the terminal to start the server
* Login to <https://cloud.mongodb.com/v2/68c57f32947b201f06c303ec> and create a project
  + Then create a cluster. Choose the free one
  + Under quick start
    - Provide user name and password
      * glenmpofu
      * tshepolovesrea0625
    - add 0.0.0.0 as the IP address since we want access from anywhere
  + go back to overview
    - click connect
    - click drivers
    - npm install mongodb
    - npm i install mongoose
  + create a new file called UserData to create the user schema