Comp 322/422 - Software Development for Wireless and Mobile Devices

Fall Semester 2019 - Week 13

Dr Nick Hayward

Cordova & React - Data - Firebase

listener events - intro

- for subscriptions and updates
  - *Firebase provides a few different events*
- for the `on()` method, we may initially consult the following documentation
- Firebase docs - `on()` events
- need to test various listeners for datastore updates

Cordova & React - Data - Firebase

listener events - `child_removed` event

- add a subscription for event updates
  - *as a child object is removed from the data store.*
- `child_removed` event may be added as follows,

```
// - listen for child_removed event relative to current ref path in DB
db.ref('egypt/ancient_sites/').on('child_removed', (snapshot) => {
  console.log('child removed = ', snapshot.key, snapshot.val());
});
```

Cordova & React - Data - Firebase

listener events - `child_changed` event

- also listen for the `child_changed` event
  - *relative to the current path passed to `ref()`*
  - e.g.

```
// - listen for child_changed event relative to current ref path in DB
db.ref('egypt/ancient_sites/').on('child_changed', (snapshot) => {
  console.log('child changed = ', snapshot.key, snapshot.val());
});
```

Cordova & React - Data - Firebase

listener events - `child_added` event

- another common event is adding a new child to the data store
  - *a user may create and add a new note or to-do item...*
  - *e.g. new child added to specified reference*

```
// - listen for child_added event relative to current ref path in DB
db.ref('egypt/ancient_sites/').on('child_added', (snapshot) => {
  console.log('child added = ', snapshot.key, snapshot.val());
});
```

Mobile Design & Development - Data Usage

Fun Exercise

A single app, multiple views

- Todo - http://linode4.cs.luc.edu/teaching/cs/demos/422/gifs/todo/

For each app, consider the following

- initial data preparation
- data loading as app starts and renders home screen
- data manipulation and updates
- data validation and integrity

~ 10 minutes

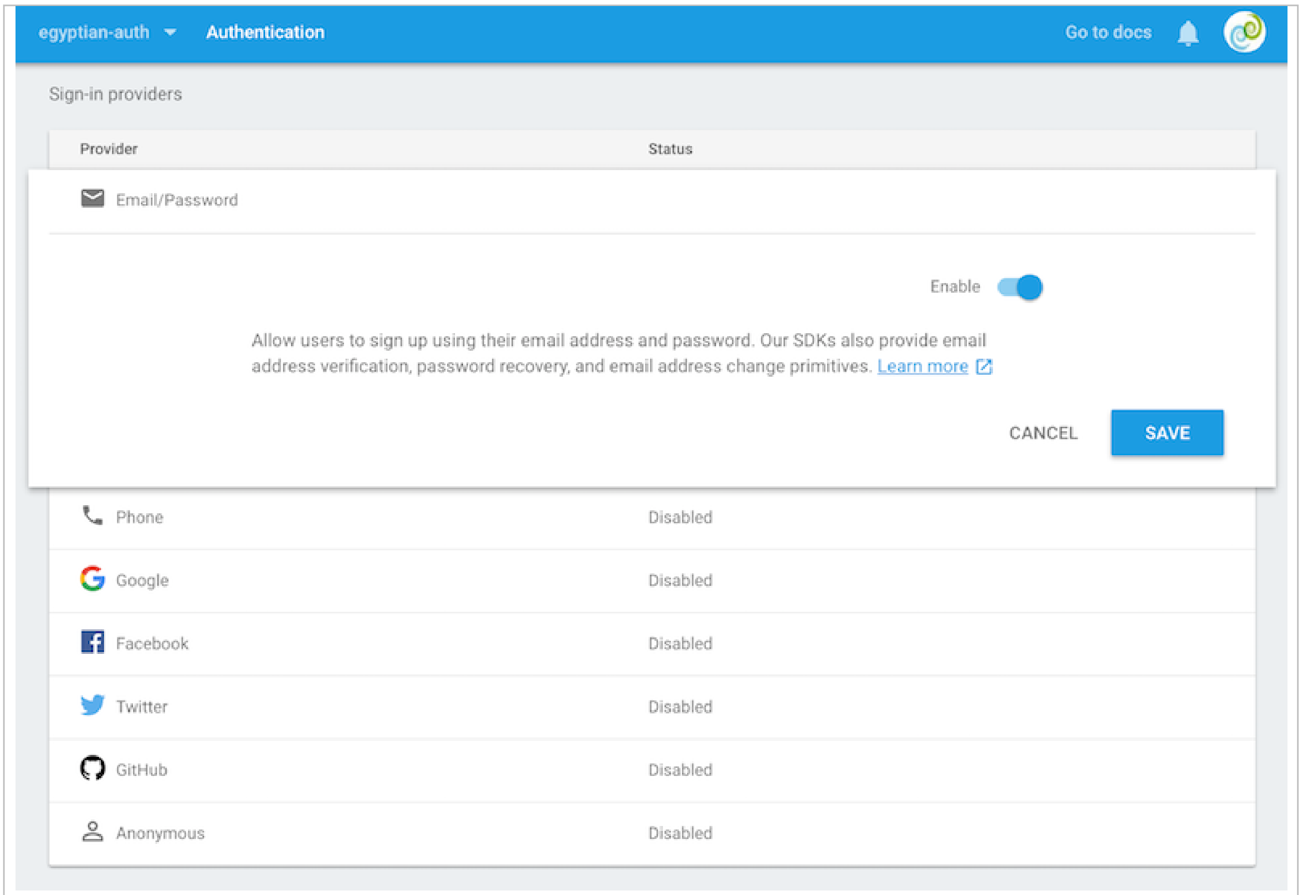
Cordova & React Native - Authentication

Firebase - setup authentication

- part of using authentication with Firebase
  - *need to explicitly configure this option in the Console Dashboard*
- need to setup the sign-in method for a particular database
- select various options and providers, including
  - *email and password*
  - *phone*
  - *Google*
  - *Facebook*
  - *Twitter*
  - *GitHub*
  - *and Anonymous*

Image - Firebase

authentication options



egyptian-auth Authentication Go to docs

Sign-in providers

Provider	Status
Email/Password	
Phone	Disabled
Google	Disabled
Facebook	Disabled
Twitter	Disabled
GitHub	Disabled
Anonymous	Disabled

Enable

Allow users to sign up using their email address and password. Our SDKs also provide email address verification, password recovery, and email address change primitives. Learn more

CANCEL SAVE

Firebase - auth options

Cordova & React Native - Firebase Auth

Cordova Login Form

- HTML for this type of form might be as follows,

```html
<form id="fb-login">
    <input type="text" value="add your username" />
    <input type="password" />
    <button id="submit-login">login</button>
</form>
```

- for single sign-in, e.g. Google, add a login button

```html
<button id="submit-login">login</button>
```

Cordova & React Native - Firebase Auth

Cordova - test form logic

- then add some initial JavaScript logic
  - *test the form and the submit login button*
- need to test a click event listener for the button
- define a callback for successful login and error handling

```
document.getElementById('submit-login').addEventListener('click', () => {
    console.log('login button clicked');
});
```

Cordova & React Native - Firebase Auth

Auth routing

- another requirement for authentication
  - *correct routing of the authentication request*
- if a user's login is successful
  - *they need to be redirected back to the app*
- if a user's login is unsuccessful
  - *user may be redirected back to the login page*
  - *user may be shown an appropriate error message*
- another consideration for routing
  - *authenticated access to an app's content*
- user should be able to view all public material
  - *plus any material appropriate to their authenticated status*

Cordova & React Native - Firebase Auth

Firebase Auth - sign-in method

- Firebase authentication requires initial configuration of settings
  - *configure and update using online console*
  - *plus various properties defined in the host app*
- add required *sign-in* methods
  - *need to modify the default config to enable this feature*
  - *in the Firebase console*
    - select *Authentication* in left menu for required database
    - select tab for *Sign-in Method* - provides various options for user authentication
  - *start by selecting Google authentication*
    - enable authentication service

Cordova & React Native - Firebase Auth

Firebase Auth - provider

- in JavaScript config file for Firebase
  - *need to define the required provider for our app*
- e.g. for *Google* sign-in method on the Firebase console
  - *need to define a provider for this service in our app*

```
// AUTH - define provider
const googleProvider = new firebase.auth.GoogleAuthProvider();
```

- usage is defined in the Firebase docs,
  - *Firebase Auth docs - Google Provider*
- we may also see similar examples for Facebook, GitHub, Twitter, &c.

Cordova & React Native - Firebase Auth

Firebase Auth - auth state change

- Firebase provides various methods for working with authentication
- relative to `firebase` object in our app's JavaScript
  - *we may call `auth()` with various additional methods*
  - *allows us to check a user's login state*
- e.g. check state of a user's authentication request and return

```
// provides listener for user authenication
// checks if a user is logged in or not...
firebase.auth().onAuthStateChanged((user) => {
    if (user) {
        console.log('user logged in');
    } else {
        console.log('user logged out');
    }
});
```

- this example provides a listener
  - *logs to the console the state of user logins to the application*
- as the app starts
  - *initially see result of query to Firebase for current user's login state*
  - *prevents unauthorised access to restricted data &c.*
  - *helps reduce login requests to remote service...*

Cordova & React Native - Firebase Auth

Firebase Auth - auth login

- then call the following function
  - *starts login process for Google authentication*

```
// start login call to return  sign-in...
const startLogin = () => {
    return firebase.auth().signInWithRedirect(googleProvider);
};
```

- if a user is not currently logged in
  - *function will show a screen with option to login*
  - *e.g. with Google account...*
- the app's auth state will again be checked
- test this login call with the login button in our app
  - e.g.

```
document.getElementById('submit-login').addEventListener('click', startLogin);
```

Cordova & React Native - Firebase Auth

Firebase Auth - auth login

- a successful login will redirect the user back to the app
  - *auth state listener will be updated, e.g. log to console*
- successful login may be persisted as needed
- we may also check authenticated users in app's Firebase console
  - *select the Authentication option*
  - *then the Users tab*
- lists all of the currently authenticated users for the app
  - *e.g. successful user logins*

Cordova & React Native - Firebase Auth

Firebase Auth - auth logout

- to allow a user to logout, start by adding an explicit logout button
  - e.g.

```html
<button id="submit-logout">logout</button>
```

- we may set this button to only show when a user is logged in
  - *then hide after logging out...*
- button will be called with a standard event listener
  - *executes a logout function*

```javascript
// start logout call to return sign-out...
const startLogout = () => {
    return firebase.auth().signOut();
};
```

Cordova & React Native - Firebase Auth

Cordova app usage

- we may now allow a user to login and logout of the application
  - *e.g. with the Google provider service with Firebase*
- we need to setup our example app to use the authenticated status
  - *e.g. authentication relative to permissions and access*
- define what an authenticated user may access and view within the app
- need to define and setup specific requirements for Cordova app
  - *e.g. plugins, config, app usage...*

Cordova & React Native - Firebase Auth

Cordova app usage - initial Firebase setup

- after creating a Firebase app & adding authentication options
  - *e.g. Google Sign-in*
- need to enable specific native SDK support in Firebase
  - *e.g. setup an Android app for the hosted Firebase project*
- in the *Settings* options for the current Firebase project
  - *add any require apps in the Your App section*
- gives us the option to add support for Firebase in various apps
  - *e.g. Android, iOS, and Web app*
- select option to add *Android* support, and complete required fields, e.g.
  - *app nickname -* `basic-fbauth`
  - *package name -* `com.ancientlives.fbauth`
  - ...

Cordova & React Native - Firebase Auth

Cordova app usage - enable Firebase Dynamic Links

- a notable difference between Firebase Authentication with web and Cordova
  - *the use of a redirect instead of the expected popup*
- Firebase requires each project to enable **Dynamic Links**
  - *permits an app to redirect a user's authentication and custom token*
- further details may be found at the following URL,
  - *https://firebase.google.com/docs/dynamic-links/*
- use the following link to select a project to use with Dynamic Links
  - *https://console.firebase.google.com/project/_/durablelinks/links/*
- after adding Dynamic Links
  - *need to record domain created for current Firebase project, e.g.*
  - *`https://myproject.page.link`*

Cordova & React Native - Firebase Auth

Cordova app usage - setup app

- after creating a Cordova app, adding support for the required platforms...
  - *need to install the following plugins for authentication support*

```
# plugin for  build info (app name, ID...)
cordova plugin add cordova-plugin-buildinfo --save
# plugin handles Universal Links (Android app link redirects)
cordova plugin add cordova-universal-links-plugin --save
# plugin handles opening secure browser views on iOS/Android mobile devices
cordova plugin add cordova-plugin-browsertab --save
# plugin handles opening a browser view in older versions of iOS and Android
cordova plugin add cordova-plugin-inappbrowser --save
# plugin handles deep linking through Custom Scheme for iOS
# & adds *com.firebase.cordova* in an iOS bundle ID...
cordova plugin add cordova-plugin-customurlscheme --variable \
    URL_SCHEME=com.firebase.cordova --save
```

Cordova & React Native - Firebase Auth

Cordova app usage - update `config.xml`

- then, we need to update `config.xml` to work with Dynamic Links
  - e.g.

```xml
<universal-links>
  <host name="myproject.page.link" scheme="https" />
  <host name="myproject.firebaseapp.com" scheme="https">
    <path url="/__/auth/callback" />
  </host>
</universal-links>
```

Cordova & React Native - Firebase Auth

Cordova app usage - update for Android

- specific to an Android app
  - *need to update the `manifestWriter.js` file in the following install directory*
  - *`./plugins/cordova-universal-links-plugin/hooks/lib/android/`*
- need to update it as follows

from

```
var pathToManifest = path.join(cordovaContext.opts.projectRoot,
'platforms', 'android', 'cordovaLib', 'AndroidManifest.xml');
```

to

```
var pathToManifest = path.join(
    cordovaContext.opts.projectRoot,
    'platforms',
    'android',
    'app',
    'src',
    'main',
    'AndroidManifest.xml');
```

- we may then add the required JS logic to the Cordova app
  - *test authentication with Firebase and Google sign-in*

Cordova & React Native - Firebase Auth

Cordova app usage - redirecting a user

- as a user logs in and logs out of an application
  - *need to ensure they are redirected correctly*
  - *to appropriate content, page, or screen for their authentication status*
- e.g. a user might be redirected to their account page after logging in
  - *then to the home page upon logout*
- with explicit routing frameworks, we may define such pages or screens
  - *including custom stack navigation...*
- we may also restrict access relative to log in status
  - *e.g. user, editor, admin...*

Cordova & React Native - Firebase Auth

Cordova app usage - user access

- for a single page app
  - *we may restrict certain content relative to a user's authentication status*
- a simple test of this status may be executed
  - *test in the state listener for Firebase authentication*
  - e.g.

```
// provides listener for user authenication
// checks if a user is logged in or not...
firebase.auth().onAuthStateChanged((user) => {
    if (user) {
        loginBtn.style.display = 'none';
        logoutBtn.style.display = 'inline';
        console.log('user logged in');
    } else {
        loginBtn.style.display = 'inline';
        logoutBtn.style.display = 'none';
        console.log('user logged out');
    }
});
```

- now modifying value of `display` property for each button
  - *updated relative to a user's authentication status*
- shows appropriate button to user dependent upon their *auth* state
- we might show certain content and options for an authenticated user
  - *or execute an async query for that user to the Firebase data store...*

Cordova & React Native - Firebase Auth

Cordova app usage - app content

- one option we may test is simply showing and hiding content
- relative to a user's *auth* state
- e.g. a user logs into the app
  - *content is queried from the connected Firebase datastore*
  - *app's UI is then updated with this content*

```
// provides listener for user authenication
// checks if a user is logged in or not...
firebase.auth().onAuthStateChanged((user) => {
    const output = document.getElementById('fb-content');
    if (user) {
        loginBtn.style.display = 'none';
        logoutBtn.style.display = 'inline';
        outputData(output);
        console.log('user logged in - data output');
    } else {
        loginBtn.style.display = 'inline';
        logoutBtn.style.display = 'none';
        clearData(output);
        console.log('user logged out - data output removed');
    }
});
```

Cordova & React Native - Firebase Auth

Cordova app usage - async data loading

- as data is loaded asynchronously from Firebase
  - *only loaded in app when a user has logged in successfully*
  - e.g.

```javascript
// get ref in db once
// call forEach() on return snapshot
// push values to local array
// unique id for each DB parent object is `key` property on snapshot
function loadData() {
  // get data from FB
    const data = db.ref('egypt/ancient_sites')
      .once('value')
      .then((snapshot) => {
        const sites = [];
        snapshot.forEach((siteSnapshot) => {
          sites.push({
            id: siteSnapshot.key,
            ...siteSnapshot.val()
          });
        });
            return sites;
      });
        // return data Promise
        return data;
}
```

Cordova & React Native - Firebase Auth

Cordova app usage - output data

- then call the `then()` method in the `outputData()` function to update the UI
  - e.g.

```javascript
// prepare data from loadData() for rendering
function outputData(elem) {
    // use data Promise - append to DOM...
    const output = loadData().then((data) => {
        for (site in data) {
            const p = document.createElement('p');
            const title = document.createTextNode(data[site]['title']);
            p.appendChild(title);
            elem.appendChild(p);
        }
    });
    // return the generated output for rendering...
    return output;
}
```

- we might then abstract this further with separate functions and logic
  - *e.g. render updates, element building, validation &c.*

Cordova & React Native - Firebase Auth

Cordova app usage - clear data

- as a user logs out of the app
  - *need a function to delete the rendered content*
  - e.g.

```
// check child nodes relative to passed element
function clearData(elem) {
    // check passed element for child nodes
    while (elem.firstChild) {
        // remove child...
        elem.removeChild(elem.firstChild);
    }
}
```

- checks passed element for child nodes
  - *while they exist, simply remove them from the UI*
  - *deletes the required app content*

Cordova & React Native - Firebase Auth

React Native - app usage

- various options for adding authentication to a React Native app
- for Firebase authentication, we may consider the following options
  - *Firebase*
  - *React Native Firebase*
- *React Native Firebase* offers a complete solution for working with Firebase services
- e.g. *React Native Firebase* includes a starter boilerplate app
  - *includes support for each service available with minimal configuration*
- we may also consider OAuth 2.0 options, e.g.
  - *React Native OAuth*

React Native - fetching data

HTML5 Fetch API - intro

- React Native also provides support for the developing HTML5 Fetch API
- also use other JS libraries such as *axios* or standard *XMLHttpRequest*
  - *no CORS (cross-origin resource sharing) issues with React Native*
- use for network based queries, API requests, and so on...
- start with a simple query structure with fetch

```
fetch('https://your-server/api/getnotes.json')
```

- Fetch API with return a promise
  - *we can then chain to* `then()`
  - *or perhaps use with async or await using ES6 JavaScript*
- might also add a second paremeter to this fetch query

```
fetch('https://your-server/api/getnotes.json', {
  method: 'POST',
  headers: {
    ...
  },
  body: JSON.stringify({
    ...
  })
})
```

React Native - fetching data

HTML5 Fetch API - working with the data

- response from a Fetch request will return a *Blob*
- response contains metadata
- access return data using a promise chain &c.

```
fetch('https://your-server/api/getnotes.json')
  .then(result => result.json())
  .then(yourData => this.setState({
    yourData
    })
  )
  .catch(error => {
    console.error(error);
  });
```

Mobile Design & Development - Authentication

Fun Exercise

Four apps with variant login and logout designs,

- Login designs - http://linode4.cs.luc.edu/teaching/cs/demos/422/gifs/login/
  - *Animation*
  - *Colour*
  - *Slide*
  - *Transition*

For each design, consider the following

- ease of use
  - *e.g. recognition of usage, options, variant logins...*
- did aesthetics help with login options?
- from a developer perspective
  - *what is required as the user logs into the app or service?*
  - *what is the relationship between the login option and app's data?*
- which login option do you find intuitive?
  - *which do you prefer?*

~ 10 minutes

React Native - navigation

intro to navigator

- React Native was initially released in 2015
  - *it came with a default navigator component to help structure internal navigation*
  - *structured stack control and management*
- community development and usage has moved towards various open project
- a popular option is the package **react-navigation**
  - *available from NPM*
- basic navigator components are stack-based
  - *similar to OnsenUI, jQuery Mobile navigation &c.*
- such components use a standard screen stack for navigating through an application
- as a user navigates to a new screen
  - *the navigator will push it onto the stack*
- as they navigate back
  - *a view &c. will simply be popped from the stack*

React Native - navigation

basic usage - part 1

- create a new app with React Native,

```
react-native init BasicAppNavigation
```

- then install `react-navigation` community package

```
yarn add react-navigation
```

or

```
npm install react-navigation --save
```

React Native - navigation

basic usage - part 2

- React Navigation designed to meet many different navigation requirements
  - *it uses a concept of different Navigators to setup apps*
- start by importing package into `App.js`

```
import { createStackNavigator, createAppContainer } from 'react-navigation';
```

- then set the required file for our configuration of the routing

```
import RootStack from './config/routes';
```

React Native - navigation

basic usage - part 3

- in the `config` folder of our `src` directory
  - *add a `routes.js` file to store details of screens and routes*

```
import HomeScreen from '../screens/homescreen';
import DataScreen from '../screens/datascreen';
import { createStackNavigator } from 'react-navigation';

const RootStack = createStackNavigator(
  {
    Home: HomeScreen,
        Data: DataScreen
    // Login: LoginScreen,
        // Logout: LogoutScreen
  },
  {
    initialRouteName: 'Home',
  }
);

export default RootStack;
```

- import required screens and their content and structure
- use screens as part of the routes for the app's navigation
- export the routes for use within our app

React Native - navigation

basic usage - part 4

- output a dynamic title for each screen navigation
  - *define a static property, `navigationOptions`*
  - *add to class for each screen component*

```
// define header title for screen
static navigationOptions = {
  title: "Ancient Sites"
}
```

- might also set this as dynamic to accept a props for each navigation request

```
// define header title for screen - add params
static navigationOptions = ({ navigation }) => ({
  title: `Sites - ${navigation.state.params.cards}`
})
```

React Native - navigation

basic usage - part 5

- add a component, such as a button, to allow us to call the `navigate` function
  - *add to `render()` method in `homescreen`*

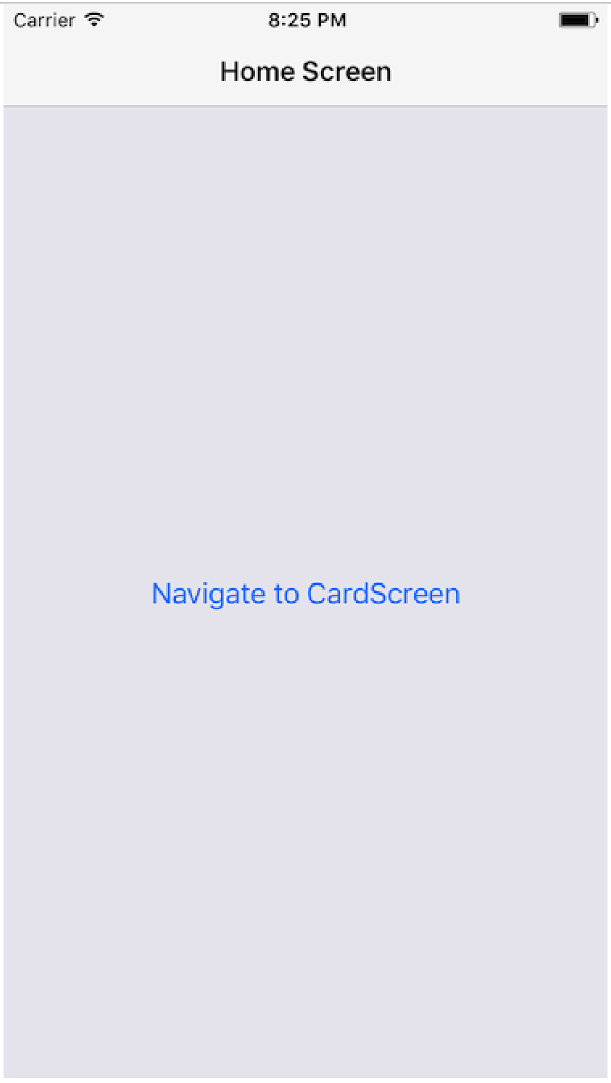
```
<Button
  title="View Data"
  onPress={() => this.props.navigation.navigate('Data', { cards: 'Egypt' })}
/>
```

- pass an argument for the required screen name
  - *defined in the config for the routes*
- we might pass a parameter for name of screen &c. to next screen
- e.g. accessed and used for title of screen

```
// define header title for screen - add params
static navigationOptions = ({ navigation }) => ({
  title: `Sites - ${navigation.state.params.cards}`
})
```

Image - React Native

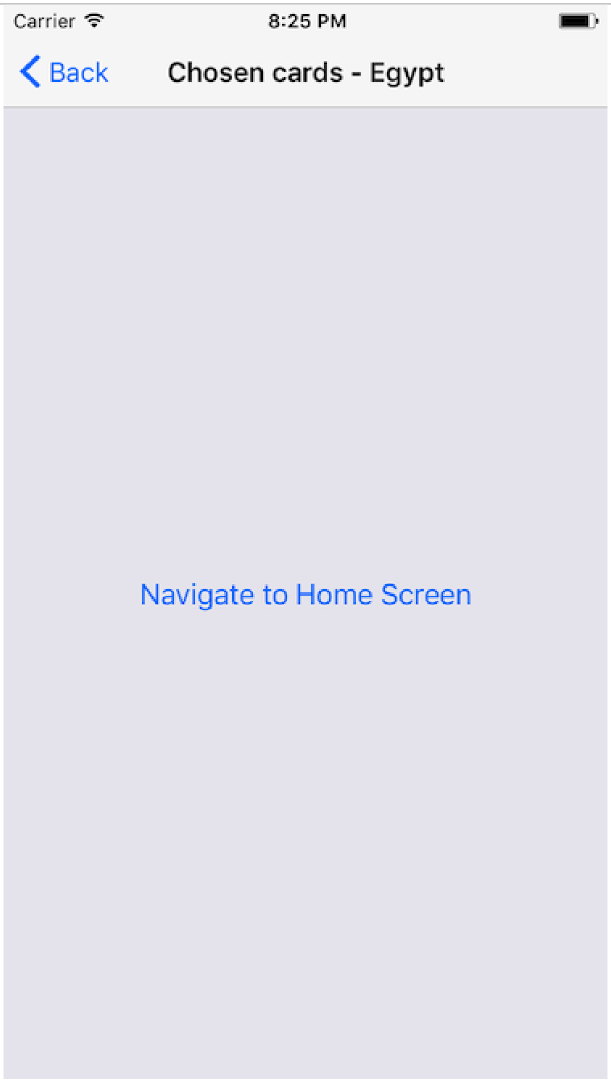
navigation - part 1



React Native - navigation

Image - React Native

navigation - part 2



React Native - navigation

Navigation & Usage

basic flows and concepts

- we may use navigation with various flows and app types
- e.g.
  - *stack navigation*
  - *tab bars*
  - *sliders*
  - *modals*
  - *splashscreens...*

React Native - navigation & data

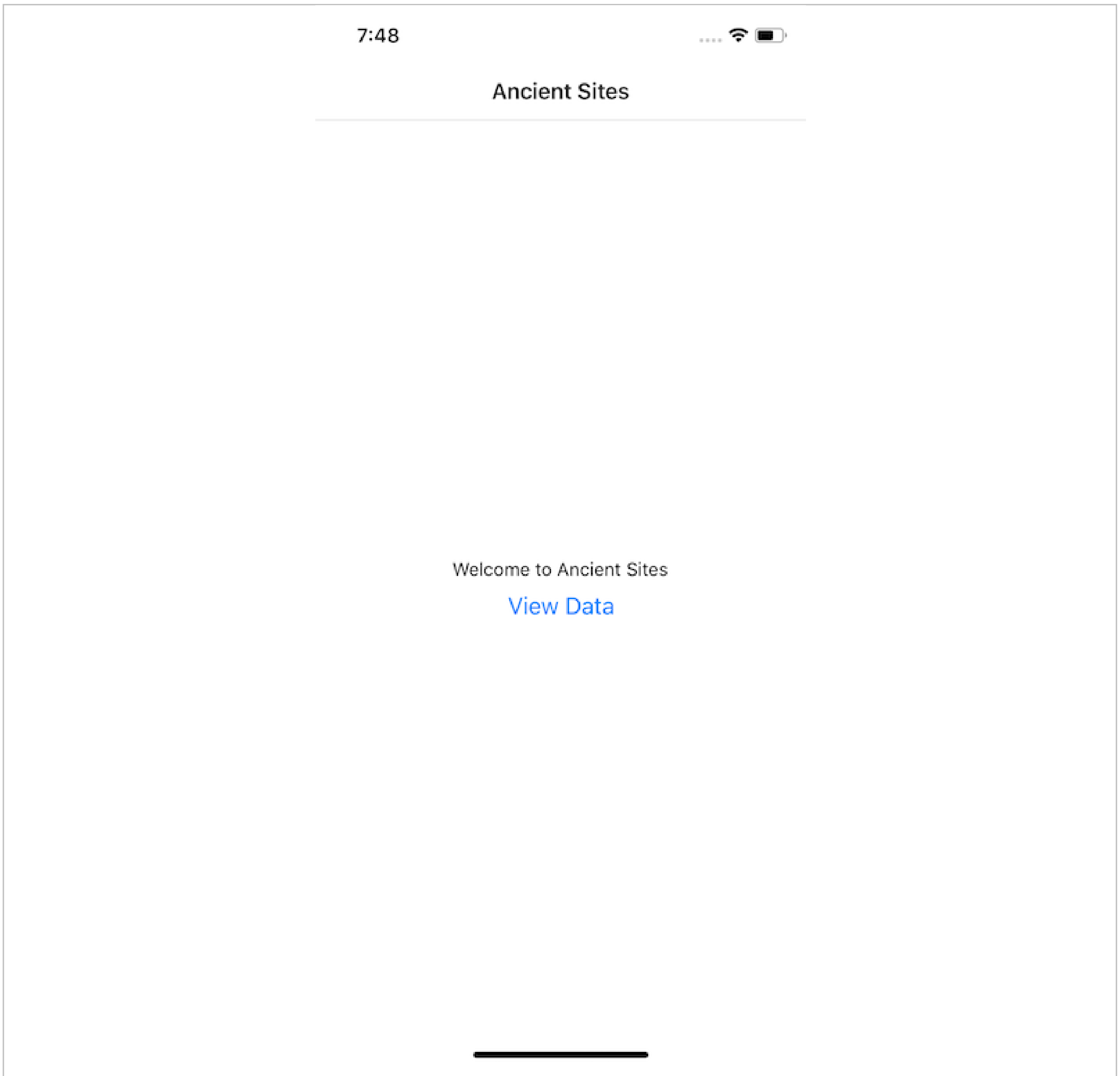
initial app structure

- combine navigation and data usage
  - *react native navigation with Firebase data loading*
- in addition to standard directories
  - `android`, `ios`, `node_modules`
- app structure with components, routes, screens...

```
.
|-- src
|    |-- components
|    |-- config
|    |-- screens
|    |-- services
|    |__ App.js
|__ index.js
```

Image - React Native

navigation & data - home

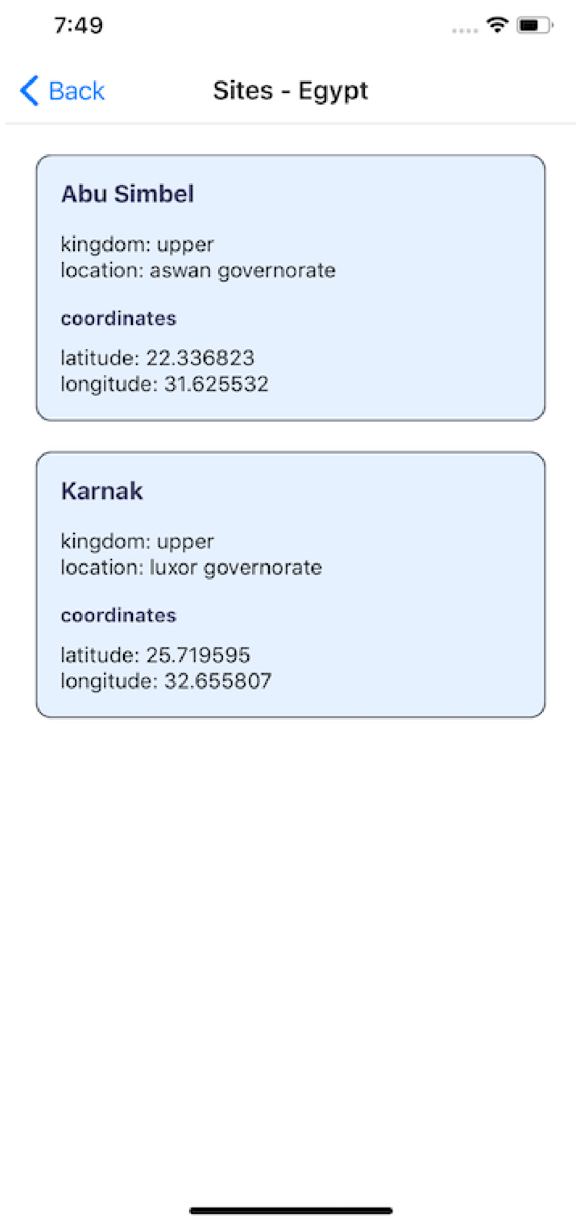


Welcome to Ancient Sites

React Native - navigation & data - home

Image - React Native

navigation & data - cards



7:49

Back

Sites - Egypt

Abu Simbel

kingdom: upper
location: aswan governorate

coordinates

latitude: 22.336823
longitude: 31.625532

Karnak

kingdom: upper
location: luxor governorate

coordinates

latitude: 25.719595
longitude: 32.655807

React Native - navigation& data - cards

React Native - navigation & data

initial app structure - navigation

- navigation structure is defined using `routes`

```
.
|-- src
|   |-- config
|   |   |__ routes.js
| ...
```

- app screens are defined in JavaScript files in `screens` directory
  - *one file per screen*

```
.
|-- src
|   |-- screens
|   |   |__ datascreen.js
|   |   |__ homescreen.js
| ...
```

React Native - navigation & data

initial app structure - card component

- we may then add specific structure for data output
  - *card output for this app...*
  - *add to `components/card.js`*

```
.
|-- src
|   |-- components
|   |   |__ card.js
| ...
```

React Native - navigation & data

initial app structure - data & services

- data logic for working with Firebase
  - *add to `services` directory*
- `api.js`
  - *initialise APIs*
  - *define listeners...*
- `firebase.js`
  - *add specific logic, config &c. for Firebase service*

```
.
|-- src
|   |-- services
|   |   |__ api.js
|   |   |__ firebase.js
| ...
```

Mobile Design & Development - Navigation

Fun Exercise

Four apps with variant navigation designs,

- Navigation designs - http://linode4.cs.luc.edu/teaching/cs/demos/422/gifs/navigation/
  - *reservations*
  - *shopping*
  - *smart home*
  - *travel passes*

For each design, consider the following

- ease of use
  - *e.g. recognition of usage, options...*
- navigation options presented to the user
  - *implicit, explicit...*
- from a developer perspective
  - *how would you manage the navigation routes?*
  - *are there any reset options for navigation?*

~ 10 minutes

Cross-platform - navigation & data

app structure - intro

- define required structure for sample app, e.g.
  - *components*
  - *config*
  - *screens*
  - *services*
- carefully note available paths and routes through app
- how are routes modified for different users
  - *authenticated*
  - *public*
  - ...
- parameters & props within the app
  - *values passed from one component to another*
  - *values passed from one screen to another*
- reset options for an app's navigation
- specifics for each OS, e.g.
  - *iOS tab bar*
  - *Android FABs, back button...*

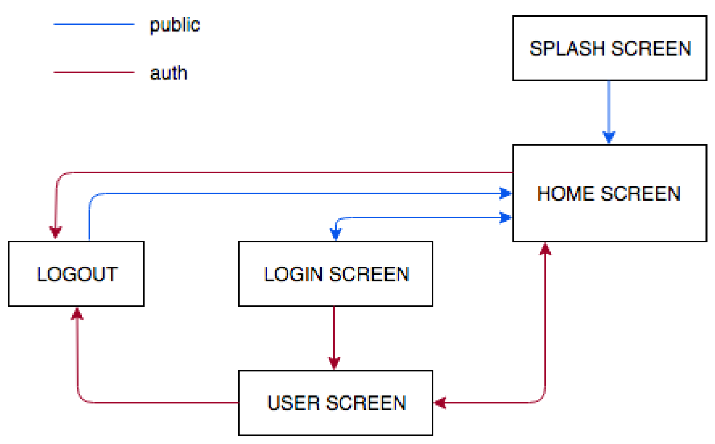
Cross-platform - navigation & data

app structure - public and auth routes

- a more detailed example might include multiple navigation paths
  - *paths relative to user authentication, data, options...*
  - *e.g. app loads with Splashscreen, then redirects to Home Screen.*
- from the *Home Screen*
  - *a user has option to follow public or authenticated routes*
  - *each route will require navigation support*
- authenticated route may contain a minimum set of screens, e.g.
  - *logout*
  - *user*
- public route will often comprise bulk of app's screens, e.g.
  - *login*
  - *data such as a rendering of data store records &c.*
  - *search*
  - *timeline*
  - *maps*
  - *...*
- some crossover between public and authenticated routes
- authenticated user may gain extra features, e.g.
  - *access to specific data for their personal account*
  - *options such as messaging and customisation.*

Image - Navigation

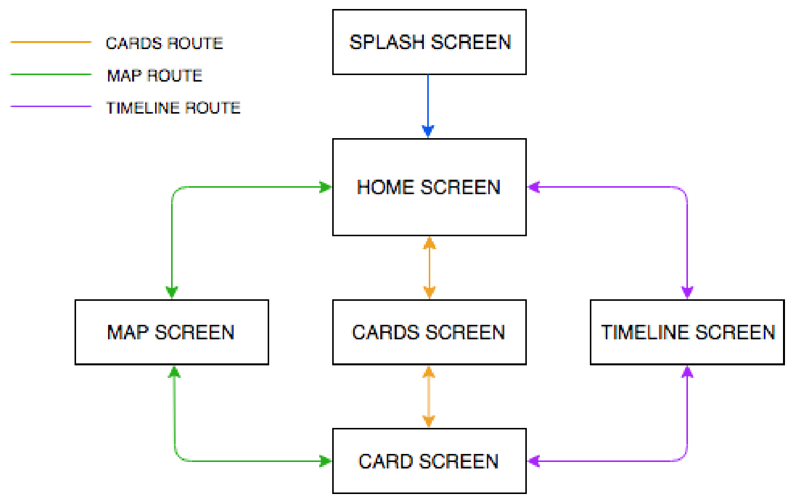
user auth



Navigation - user auth

Image - Navigation

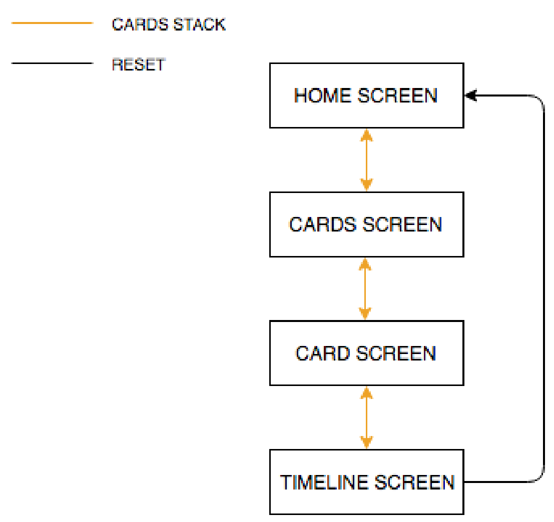
app routes



Navigation - app routes

Image - Navigation

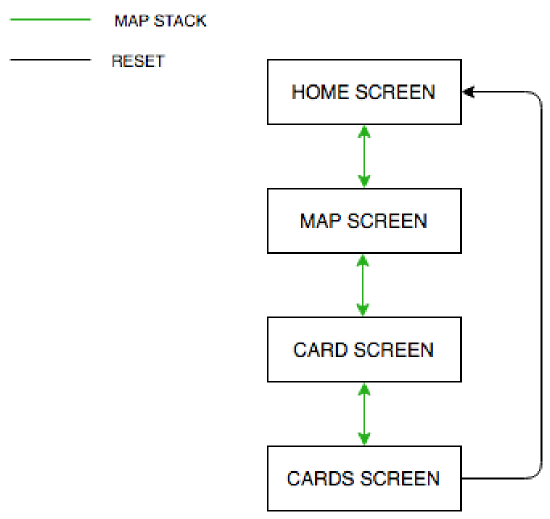
paths and stacks



Navigation - stack example - cards

Image - Navigation

paths and stacks



Navigation - stack example - map

References

- React Native
  - *Firebase NPM package*
  - *React Native Firebase*
  - *React Navigation*
  - *React Native OAuth*
- Various
  - *Axios JS library*
  - *Firebase*
  - *Firebase - database rules*
  - *Firebase Docs - DataSnapshot*
  - *Firebase docs - `on()` events*
  - *Google's Cloud Platform*
  - *MDN - Fetch API*
  - *XMLHttpRequest*
  - *Yarn - Firebase*