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

Fall Semester 2019 - Week 12

Dr Nick Hayward

React Native - component usage

lists - FlatList

- React Native provides suggested view components for lists
  - *two primary examples include `FlatList` and `SectionList`*
- `FlatList` is meant to be used for long lists of data
  - *in particular where data items may change during the lifecycle of an app*
- `FlatList` will only render elements currently shown on screen
  - *not all of the available elements at the same time*

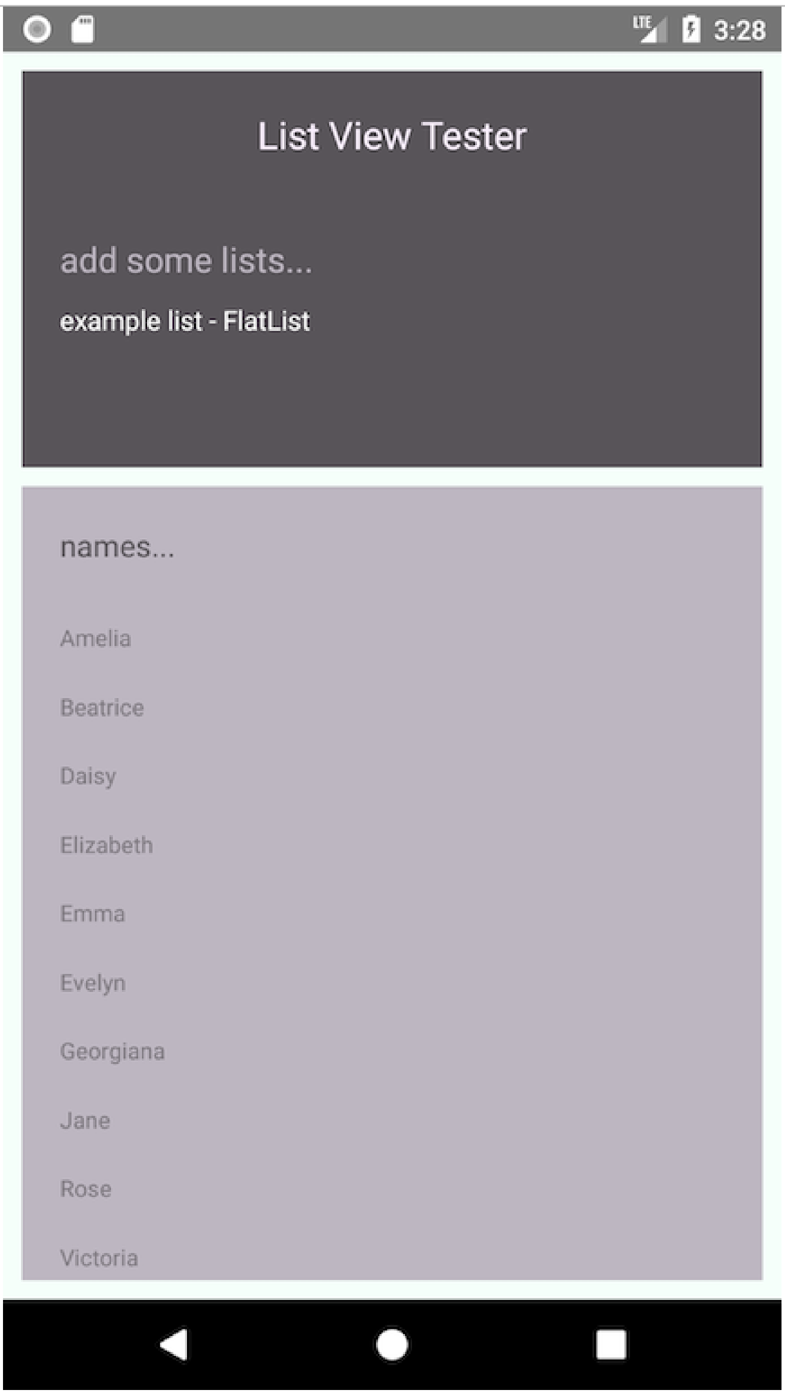
```
<FlatList
    data={[
        {key: 'Amelia'},
        {key: 'Beatrice'},
        {key: 'Daisy'},
    ]
    }
    renderItem={({item}) => <Text style={styles.listItem}>{item.key}</Text>}
/>
```

- component expects two *props*
  - *`data` for the list itself*
  - *`renderItem` to define the output structure for each list item*

```
renderItem={() => <Text></Text>}
```

Image - React Native - Component Usage

lists - FlatList



List View Tester

add some lists...

example list - FlatList

names...

Amelia

Beatrice

Daisy

Elizabeth

Emma

Evelyn

Georgiana

Jane

Rose

Victoria

React Native - FlatList

React Native - component usage

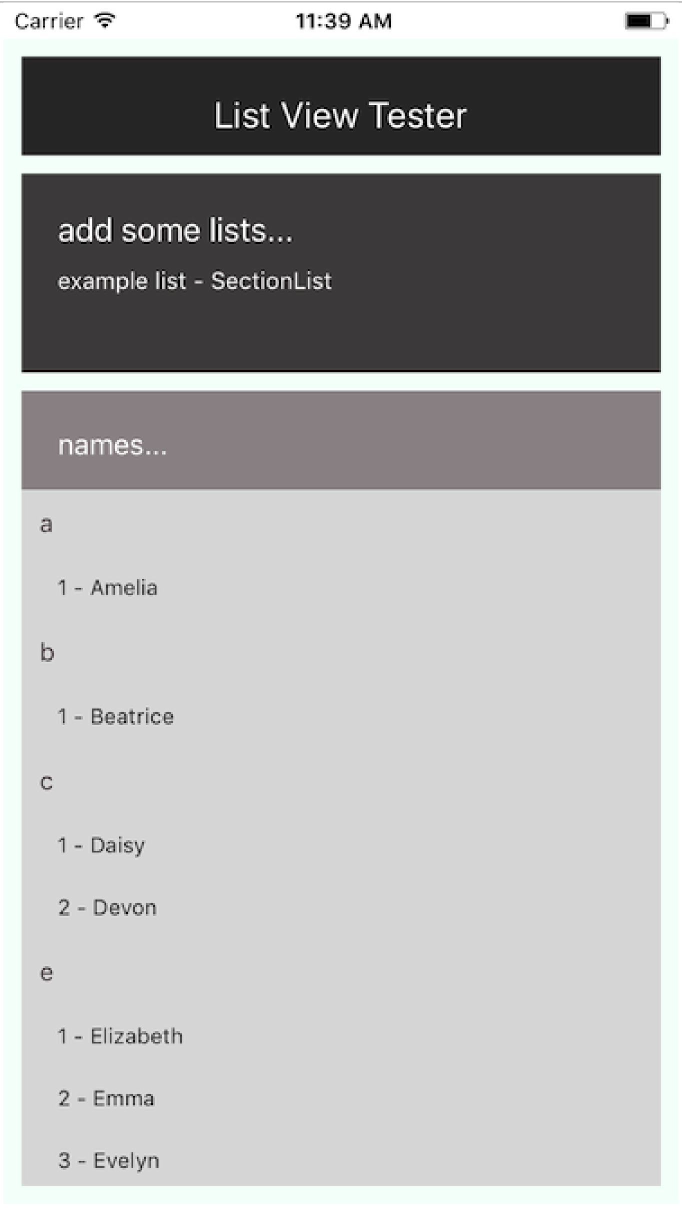
lists - SectionList

- may also create section breaks in a list of data. e.g.

```
<SectionList
  sections={[
        {title: 'a', data:[{key: 1, name: 'Amelia'}]},
        {title: 'b', data:[{key: 1, name: 'Beatrice'}]},
    {title: 'c', data: [{key: 1, name: 'Daisy'}, {key: 2, name: 'Devon'}]},
        {title: 'e', data: [{key: 1, name: 'Elizabeth'}, {key: 2, name: 'Emma'},
        {title: 'g', data:[{key: 1, name: 'Georgiana'}]},
        {title: 'j', data:[{key: 1, name: 'Jane'}]},
        {title: 'r', data:[{key: 1, name: 'Rose'}]},
        {title: 'v', data: [{key: 1, name: 'Victoria'}, {key: 2, name: 'Violet'}]
        {title: 'y', data:[{key: 1, name: 'Yvaine'}]},
    ]}
    //keyExtractor={item => item}
  renderItem={({item}) => <Text style={styles.listItem}>{item.key} - {item.name}<
  renderSectionHeader={({section}) => <Text style={styles.heading4}>{section.titl
/>
```

Image - React Native - Component Usage

lists - SectiontList - top



Carrier 11:39 AM

List View Tester

add some lists...

example list - SectionList

names...

a

1 - Amelia

b

1 - Beatrice

c

1 - Daisy

2 - Devon

e

1 - Elizabeth

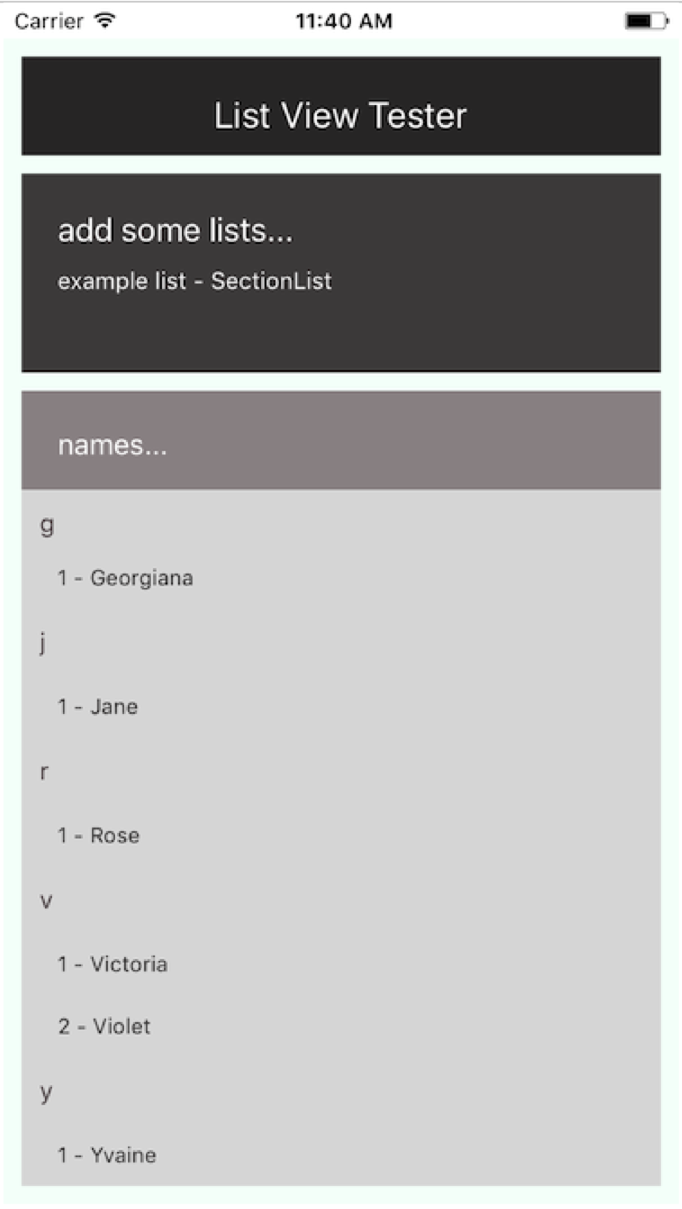
2 - Emma

3 - Evelyn

React Native - SectionList - top

Image - React Native - Component Usage

lists - SectiontList - bottom


Carrier

11:40 AM

List View Tester

add some lists...

example list - SectionList

names...

g

1 - Georgiana

j

1 - Jane

r

1 - Rose

v

1 - Victoria

2 - Violet

y

1 - Yvaine

React Native - SectionList - bottom

React Native - component usage

ScrollView

- scrolling in React Native apps is achieved with a generic scrolling container
  - *`ScrollView`*
- specific view container can itself accept multiple child components and views
- scrollview container option to specify direction
  - *either horizontal or vertical*
- general usage
  - *add a `ScrollView` using the same general pattern as a standard `View` component*
  - *return a `ScrollView` as eiher the primary container for a component*
  - *or a child of a standard `View`*
  - *an app's screen may either scroll top to bottom*
  - *or simply present a component with scroll features*

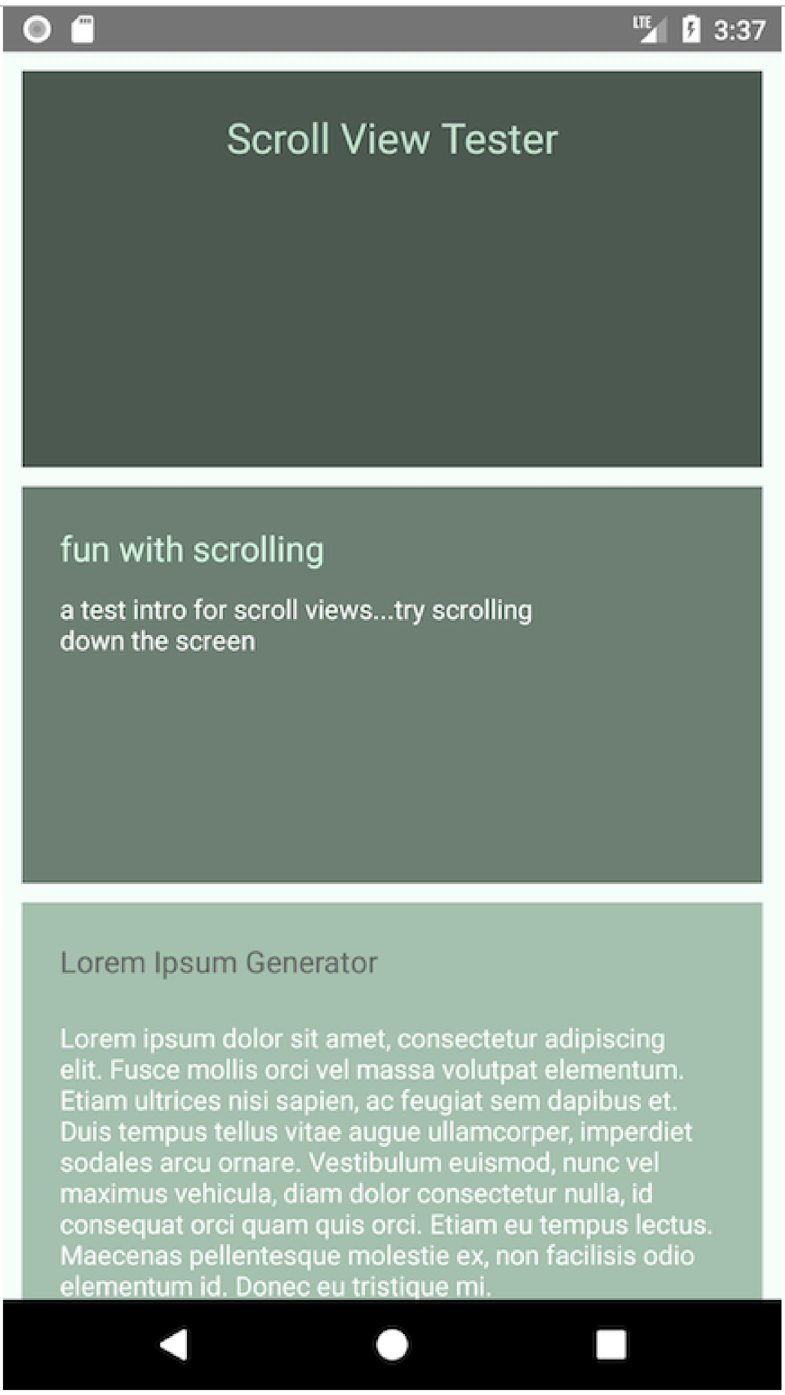
React Native - component usage

ScrollView - example

```
export default class ScrollTester extends Component {
  render() {
    return (
                <View style={styles.container}>
                    <View style={styles.headingBox}>
            <Text style={styles.heading1}>
                Scroll View Tester
            </Text>
            </View>
            <View style={styles.subHeadingBox}>
            <Text style={styles.heading2}>
                {intro.heading}
            </Text>
            <Text style={styles.content}>
                {intro.description}
            </Text>
            </View>
                    <ScrollView>
                    <View style={styles.contentBox}>
            <Text style={styles.heading3}>
                Lorem Ipsum Generator
            </Text>
            <Text style={styles.content}>
                            ...
                        </Text>
            </View>
                    </ScrollView>
        </View>
    );
  }
}
```

Image - React Native - Component Usage

lists - ScrollView



Scroll View Tester

fun with scrolling

a test intro for scroll views...try scrolling down the screen

Lorem Ipsum Generator

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce mollis orci vel massa volutpat elementum. Etiam ultrices nisi sapien, ac feugiat sem dapibus et. Duis tempus tellus vitae augue ullamcorper, imperdiet sodales arcu ornare. Vestibulum euismod, nunc vel maximus vehicula, diam dolor consectetur nulla, id consequat orci quam quis orci. Etiam eu tempus lectus. Maecenas pellentesque molestie ex, non facilisis odio elementum id. Donec eu tristique mi.

React Native - ScrollView

React Native - Component usage

text input

- a default component to handle user text input
- component `TextInput` is similar to a standard input field
  - *allowing a user to simply enter any required text content*
- to use `TextInput` with an app
  - *need to add the default module from React Native*
  - *add as part of the standard `import` statement*
- `TextInput` component includes a useful *prop*, `onChangeText`
  - *accepts callback function for each time text is changed in input field*
- also includes a complementary *prop*, `onSubmitEditing`
  - *handles text as it is submitted*
  - *again using a defined callback function*

React Native - Component usage

text input - props usage

- might accept user text input for a given value
  - *such as a name, place, &c.*
- then dynamically update the *view*
- e.g.

```
<TextInput
  style={styles.textInput}
  placeholder={this.state.quoteInput}
  onChangeText={(quoteText) => this.setState({quoteText})}
/>
```

React Native - Component usage

text input - props and state

- example relies upon calling and setting state for the app
  - *relative to `TextInput` and various `Text` components*
- simple constructor for this app
  - *pass required `props` and define intial values for `state`*

```
export default class TextUpdater extends Component {
  constructor(props) {
    super(props);
    this.state = {
      quoteInput: 'enter a favourite quotation...',
      quoteText: 'the unexamined life is not worth living...'
    };
  }
}
```

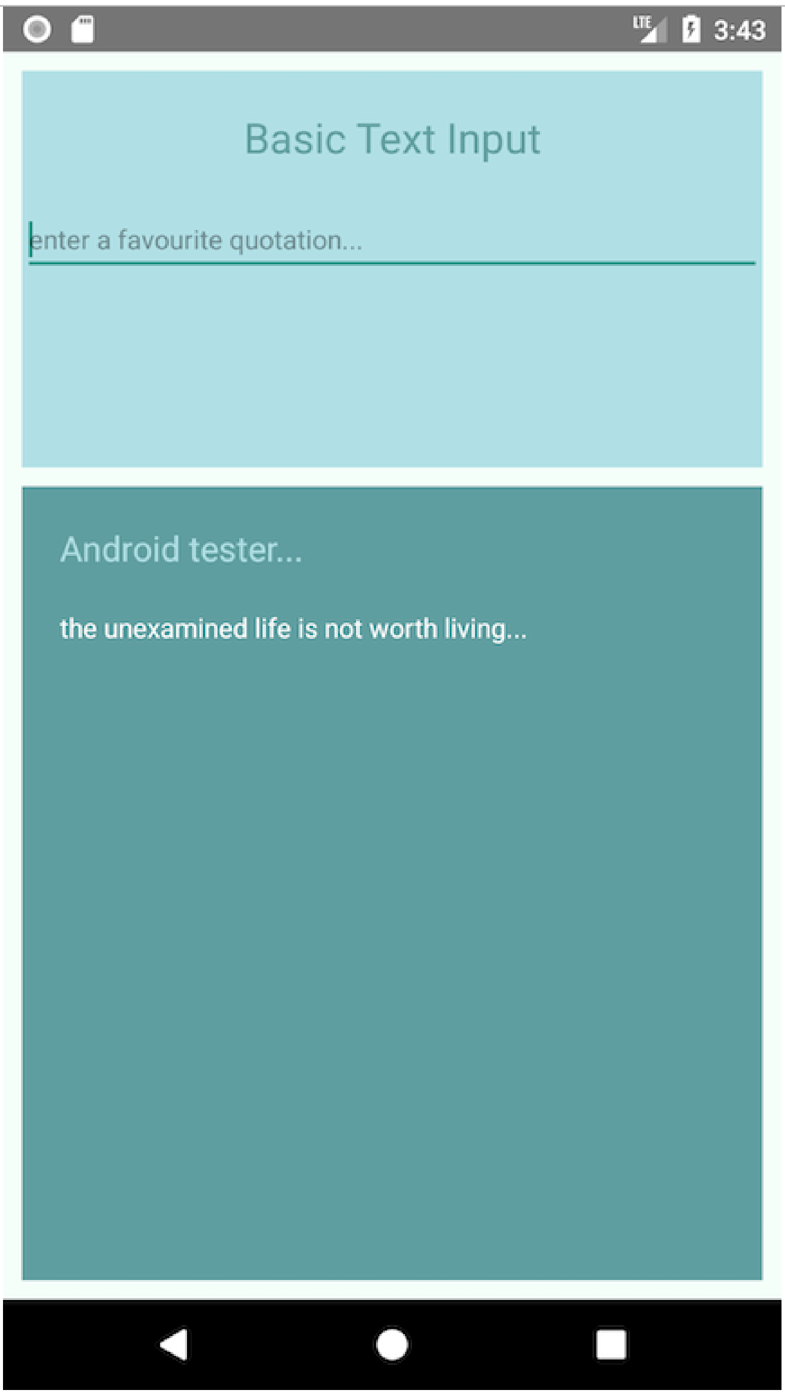
- then use the properties on `state`
  - *to set initial values for the text input field and the text output,*

```
<TextInput
  style={styles.textInput}
  placeholder={this.state.quoteInput}
  onChangeText={(quoteText) => this.setState({quoteText})}
/>
```

```
<Text style={styles.content}>
    {this.state.quoteText}
</Text>
```

Image - React Native - Component Usage

text input



Basic Text Input

enter a favourite quotation...

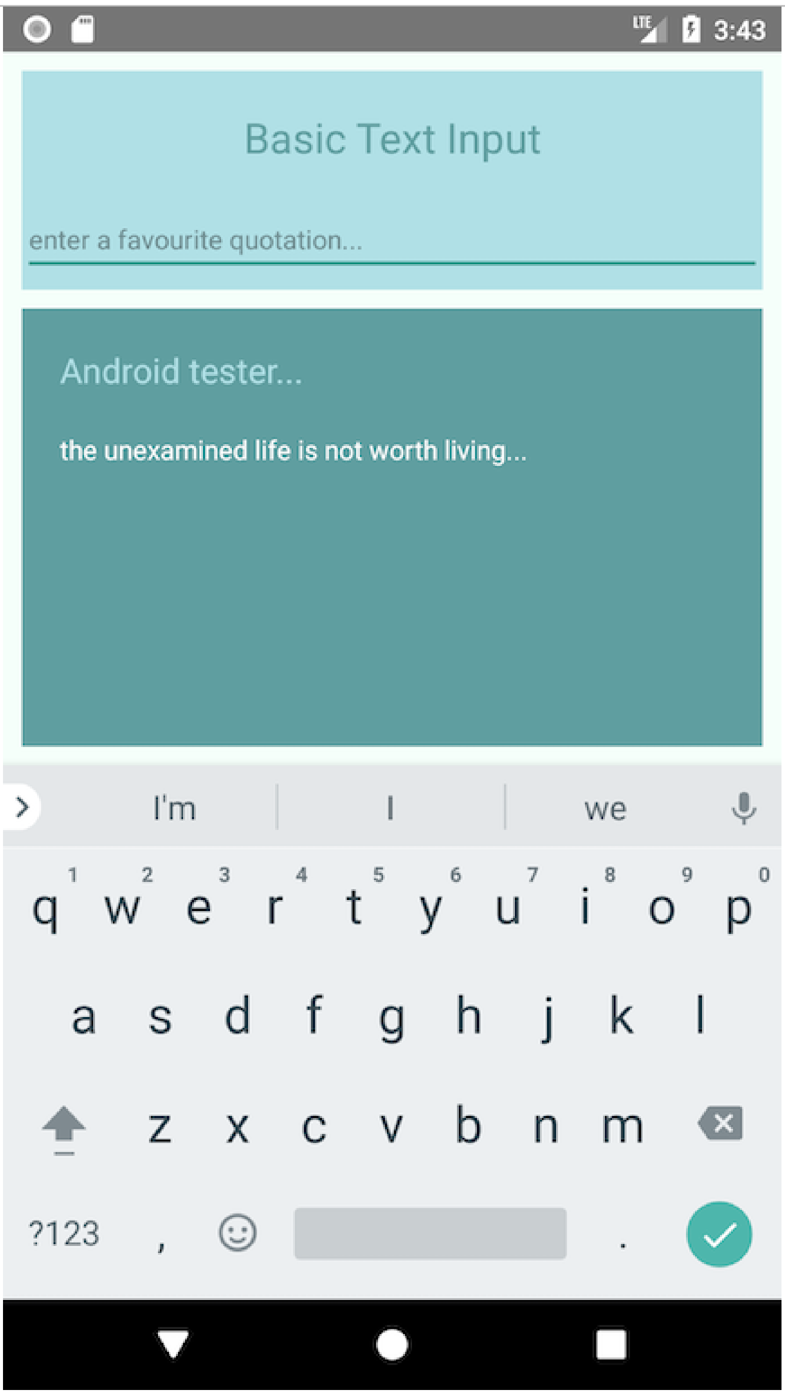
Android tester...

the unexamined life is not worth living...

React Native - text input

Image - React Native - Component Usage

text input


Basic Text Input

enter a favourite quotation...

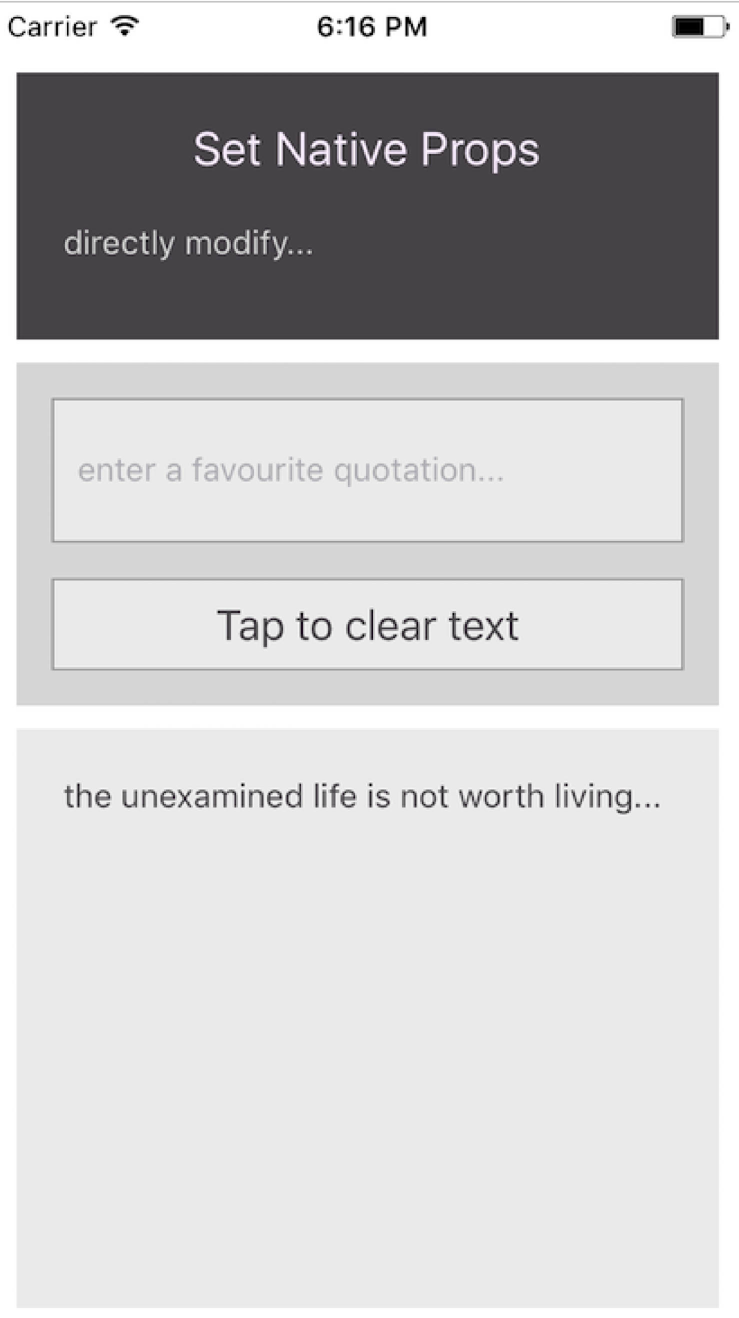
Android tester...

the unexamined life is not worth living...

React Native - text input

Image - React Native - Component Usage

text input - use setNativeProps

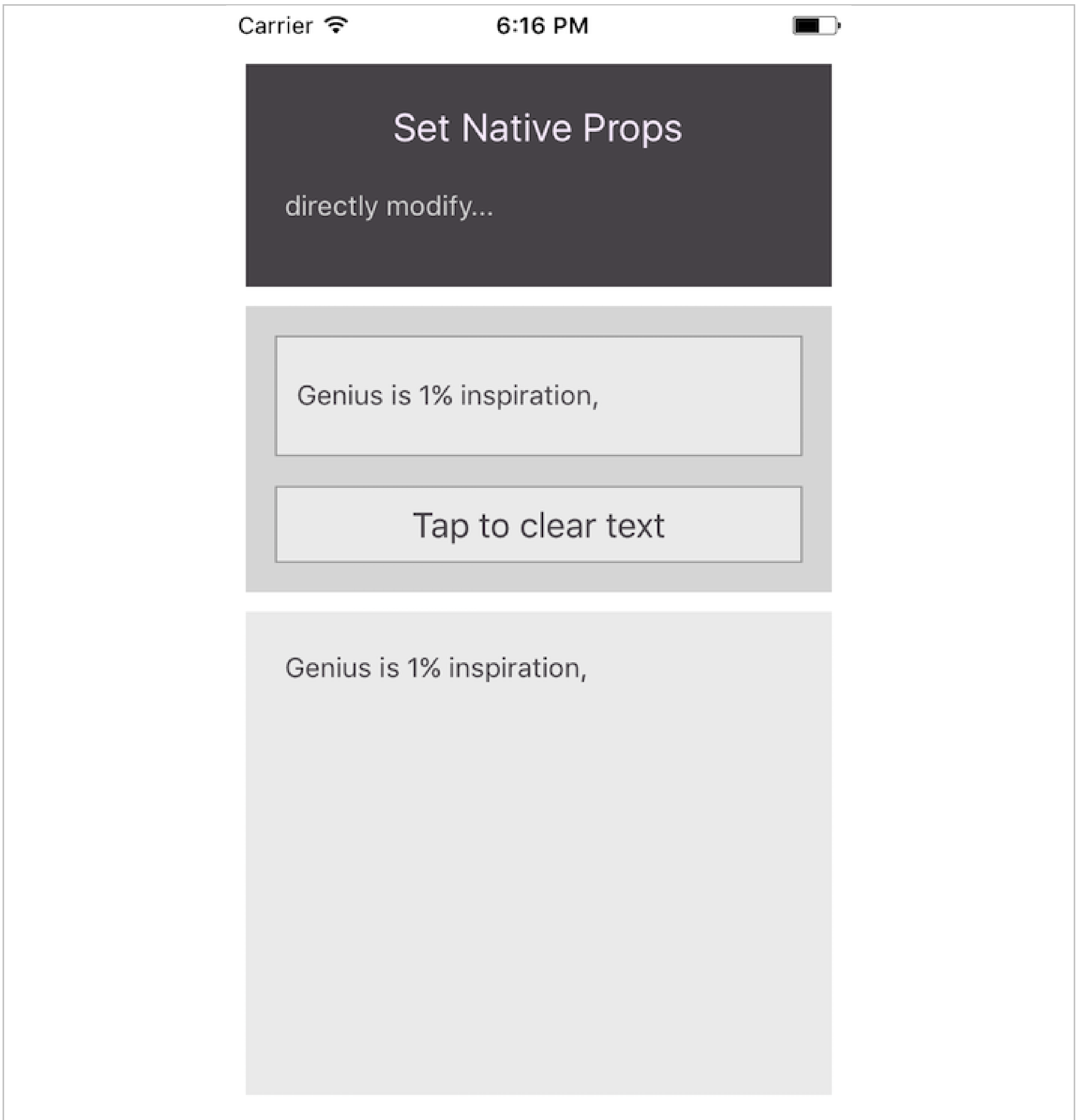


the unexamined life is not worth living...

React Native - text input

Image - React Native - Component Usage

text input - use setNativeProps



React Native - text input

React Native - component usage

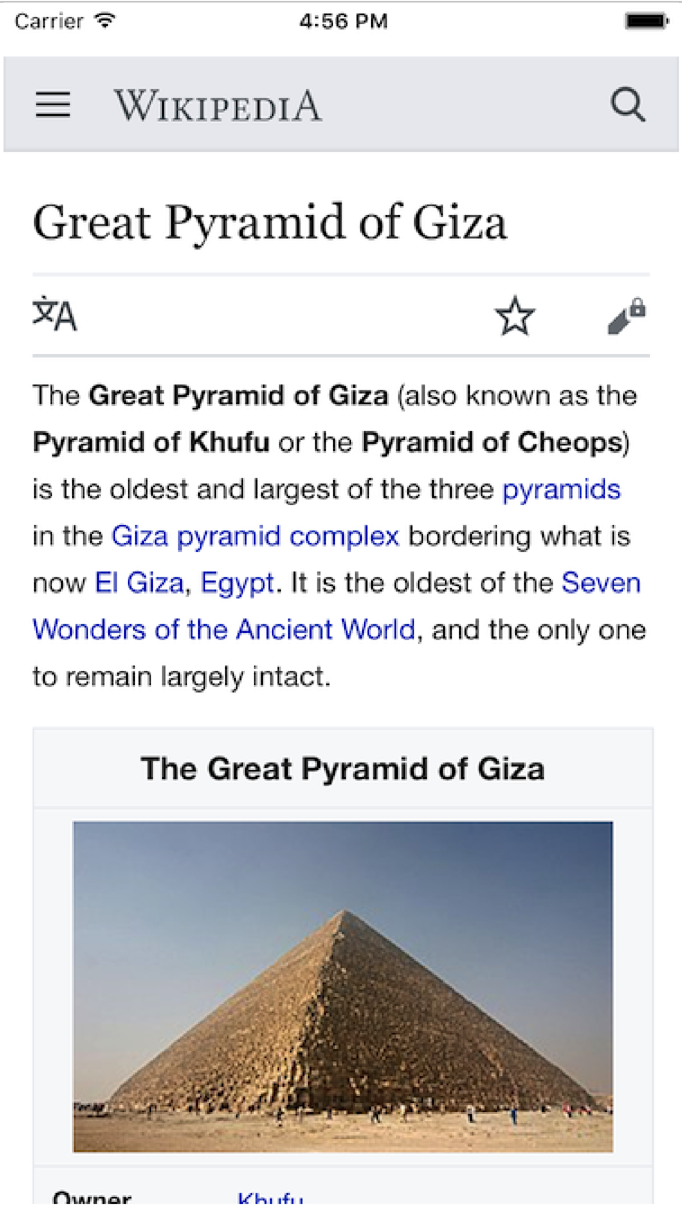
embed web content

- React Native offers a component solution for embedding web content
  - *embedded directly in a WebView*
  - *as a child to an existing view &c.*
- similar functionality to native WebView modules
- `WebView` component provides developers with a variety of props
  - *to help manipulate and structure a rendered web page*
- also use various available callbacks
  - *provide an option to register to specific events*
  - *e.g. error handling, message responses, navigation state change...*

```
<WebView style={styles.web}
  scalesPageToFit
  automaticallyAdjustContentInsets
  source={{
  uri: 'https://en.wikipedia.org/wiki/Great_Pyramid_of_Giza'
  }} />
```

Image - React Native - Component Usage

WebView - load external page &c.



Great Pyramid of Giza

The **Great Pyramid of Giza** (also known as the **Pyramid of Khufu** or the **Pyramid of Cheops**) is the oldest and largest of the three pyramids in the Giza pyramid complex bordering what is now El Giza, Egypt. It is the oldest of the Seven Wonders of the Ancient World, and the only one to remain largely intact.

The Great Pyramid of Giza

Owner Khufu

React Native - WebView

React Native - component usage

iOS - SegmentedControlIOS

- some components in React Native may be specific to a given mobile OS
  - *e.g. Segmented Control component is specific to iOS*
- offers a simple split option to switch between two groupings of content
- e.g. we might use this component as follows

```
<SegmentedControlIOS
  values={['Giza', 'Luxor']}
  selectedIndex={this.state.selectedIndex}
  onChange={(event) => {
    this.setState({selectedIndex: event.nativeEvent.selectedSegmentIndex});
  }}
/>
```

- instead of passing expected `onValueChange` props
  - *we can pass a callback prop for `onChange`*
- prop will receive an event argument
  - *e.g. from `nativeEvent` as shown in this example*
- also abstract this usage to pass in required values for each segment

Mobile Design & Development - More UI Components & Usage

Fun Exercise

Four groups, two apps

- Music - http://linode4.cs.luc.edu/teaching/cs/demos/422/gifs/music/
- Travel Booking - http://linode4.cs.luc.edu/teaching/cs/demos/422/gifs/travelbooking/

For each app, consider the following

- define UI components for the app?
- how is the app using lists for various views?
- how is the app combining multiple components to create the required UI layout?
  - *e.g. various list views, scrolling, text input &c.*
- how are the UI components defining UX for the app?

~ 10 minutes

Cordova & React Native - Data

intro

- already seen data examples for Cordova
  - *including IndexedDB, Native Storage, various APIs...*
- React Native equally capable of accessing data stores
  - *a popular option for object based data storage is Firebase*
- useful to understand how React Native works
  - *with remote queries, fetching data, and authentication...*
- setup and add our own login and authentication for an app
- leverage an existing social provider
  - *e.g. Facebook, GitHub, Google, Microsoft, Twitter...*
- similar patterns and usage to web apps

Cordova & React Native - Data - Firebase

NoSQL options

- other data store and management options now available to us as developers
- depending upon app requirements consider
  - *Firebase*
  - *RethinkDB*
  - *AWS - including Amplify*
  - *MongoDB, Redis...*
- as a data store, Firebase offers a hosted NoSQL database
  - *data store is JSON-based*
  - *offering quick, easy development from webview to data store*
- syncs an app's data across multiple connected devices in milliseconds
  - *available for offline usage as well*
- provides an API for accessing these JSON data stores
  - *real-time for all connected users*
- Firebase as a hosted option more than just data stores and real-time API access
- Firebase has grown a lot over the last year
  - *many new features announced at Google I/O conference in May 2016*
  - *analytics, cloud-based messaging, app authentication*
  - *file storage, test options for Android*
  - *notifications, adverts...*

Cordova & React Native - Data - Firebase

Firebase - intro

- Cordova & React Native do not limit data stores or queries to just Firebase
- Firebase is hosted platform, acquired by Google
  - *provides options for data starage, authentication, real-time database querying...*
- it provides and API for data access
  - *access and query JavaScript object data stores*
  - *query in real-time*
  - *listeners available for all connected apps and users*
  - *synchronisation in milliseconds for most updates...*
  - *notifications*

Cordova & React Native - Data - Firebase

Firebase - Authentication

- **authentication** with Firebase provides various backend services and SDKs
  - *help developers manage authentication for an app*
  - *service supports many different providers, including Facebook, Google, Twitter &c.*
  - *using industry standard **OAuth 2.0** and **OpenID Connect** protocols*
- custom solutions also available per app
  - *email*
  - *telephone*
  - *messaging*
  - ...

Cordova & React Native - Data - Firebase

Firebase - Cloud Storage

- **Cloud Storage** used for uploading, storing, downloading files
  - *accessed by apps for file storage and usage...*
  - *features a useful safety check if and when a user's connection is broken or lost*
  - *files are usually stored in a Google Cloud Storage bucket*
  - *files accessible using either Firebase or Google Cloud*
  - *consider using Google Cloud platform for image filtering, processing, video editing...*
  - *modified files may then become available to Firebase again, and connected apps*
  - e.g. *Google's Cloud Platform*

Cordova & React Native - Data - Firebase

Firebase - Real-time database

- **Real-time Database** offers a hosted NoSQL data store
  - *ability to quickly and easily sync data*
  - *data synchronisation is active across multiple devices, in real-time*
  - *available as and when the data is updated in the cloud database*
- other services and tools available with Firebase
  - *analytics*
  - *advertising services such as adwords*
  - *crash reporting*
  - *notifications*
  - *various testing options...*

Cordova & React Native - Data - Firebase

Firebase - basic setup

- start using Firebase by creating an account with the service
  - *using a standard Google account*
  - *Firebase*
- login to Firebase
  - *choose either Get Started material or navigate to Firebase console*
- at *Console* page, get started by creating a new project
  - *click on the option to* `Add project`
  - *enter the name of this new project*
  - *and select a region*
- then redirected to the *console dashboard* page for the new project
  - *access project settings, config, maintenance...*
- reference documentation for the Firebase Real-Time database,
  - *https://firebase.google.com/docs/reference/js/firebase.database*

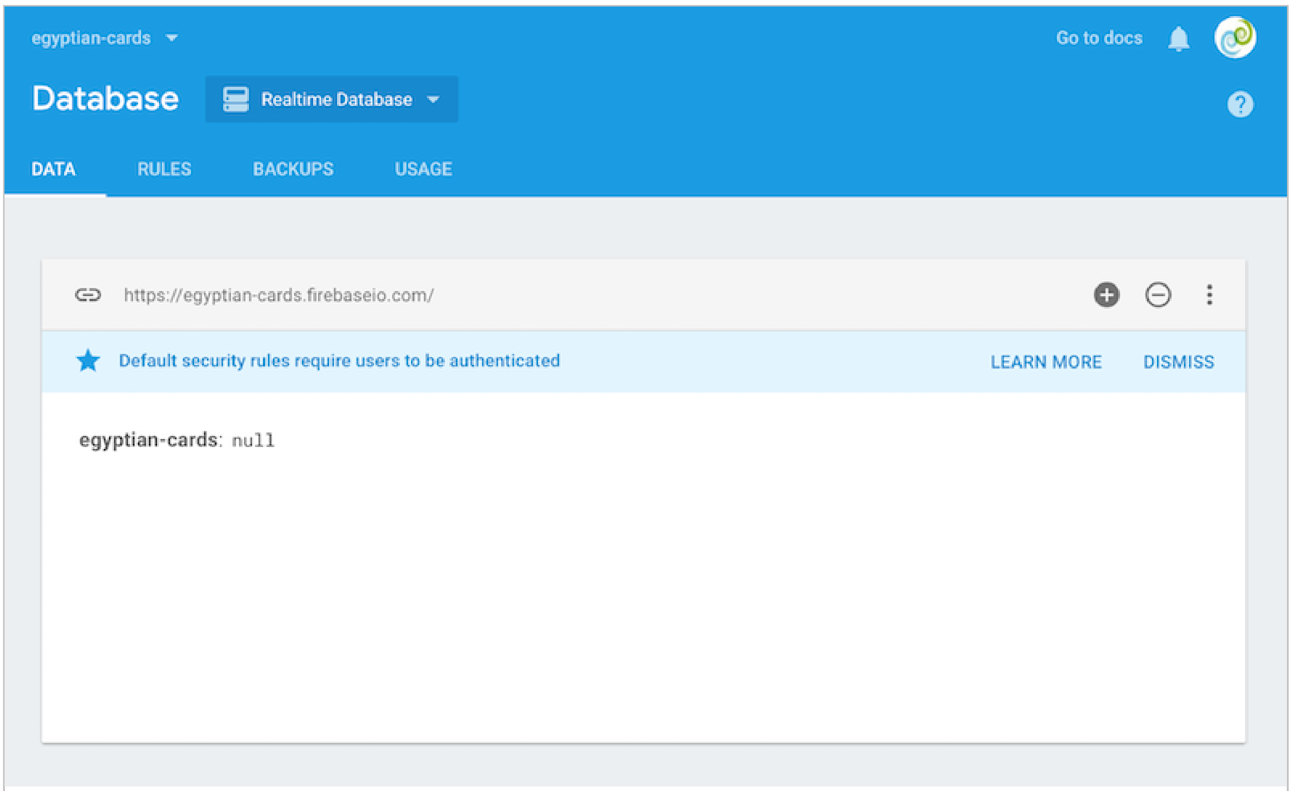
Cordova & React Native - Data - Firebase

Firebase - create real-time database

- now setup a database with Firebase for a test React Native app
- start by selecting *Database* option from left sidebar on the Console Dashboard
  - *available under the DEVELOP option*
- then select *Get Started* for the real-time database
- presents an empty database with an appropriate name to match current project
- data will be stored in a JSON format in the real-time database
- working with Firebase is usually simple and straightforward for most apps
- get started quickly direct from the Firebase console
  - *or import some existing JSON...*

Image - Firebase

create a database



https://egyptian-cards.firebaseio.com/

Default security rules require users to be authenticated

egyptian-cards: `null`

Firebase - create a database

Cordova & React Native - Data - Firebase

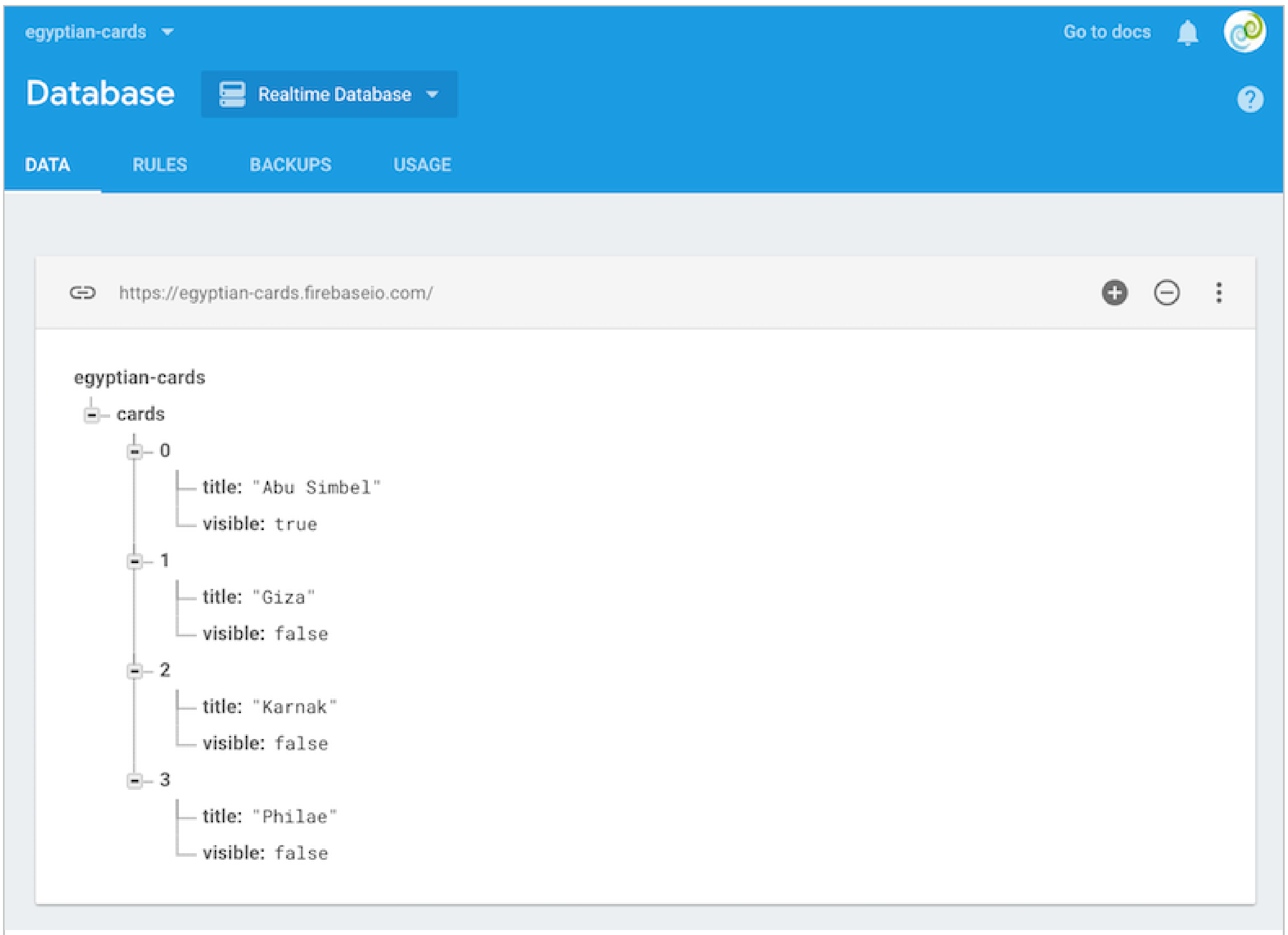
Firebase - import JSON data

- we might start with some simple data to help test Firebase
- import JSON into our test database
  - *then query the data &c. from the app*

```json
{
  "cards": [
    {
      "visible": true,
      "title": "Abu Simbel",
      "card": "temple complex built by Ramesses II"
    },
    {
      "visible": false,
      "title": "Amarna",
      "card": "capital city built by Akhenaten"
    },
    {
      "visible": false,
      "title": "Giza",
      "card": "Khufu's pyramid on the Giza plateau outside Cairo"
    },
    {
      "visible": false,
      "title": "Philae",
      "card": "temple complex built during the Ptolemaic period"
    }
  ]
}
```

Image - Firebase

JSON import



```
egyptian-cards
  cards
    0
      title: "Abu Simbel"
      visible: true
    1
      title: "Giza"
      visible: false
    2
      title: "Karnak"
      visible: false
    3
      title: "Philae"
      visible: false
```

Firebase - import JSON file

Cordova & React Native - Data - Firebase

Firebase - permissions

- initial notification in Firebase console after creating a new database
  - *Default security rules require users to be authenticated*
- permissions with Firebase database
  - *select RULES tab for current database*
- lots of options for database rules
  - *Firebase - database rules*
- e.g. for testing initial React Native we might remove authentication rules
- change rules as follows

from

```
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

to

```
{
  "rules": {
    ".read": "true",
    ".write": "true"
  }
}
```

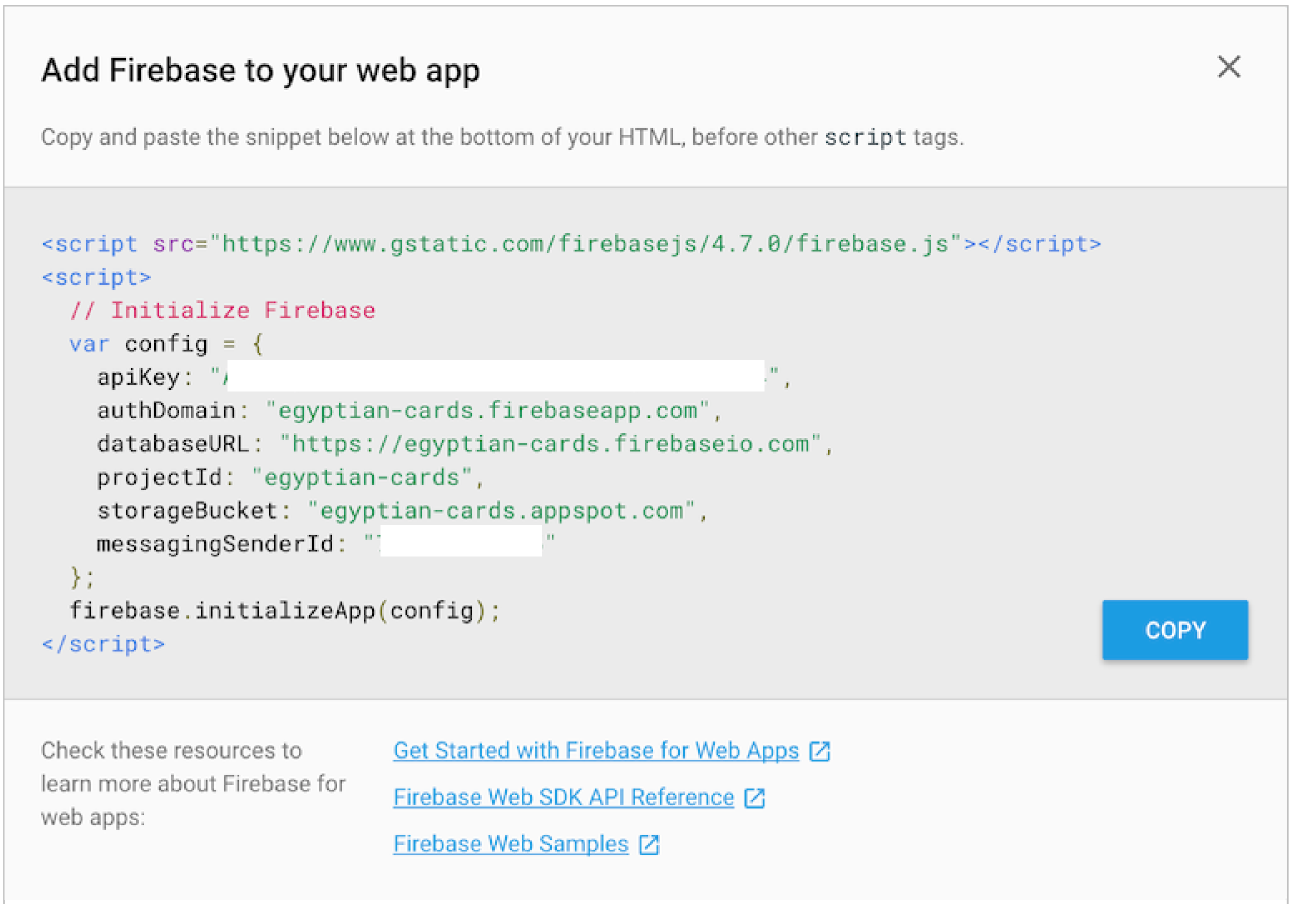
React Native - Data - Firebase

add Firebase to React Native - part 1

- we can now test our new Firebase database with an app
- need to start by getting some useful information from Firebase
  - *select the Project Overview link in the left sidebar*
  - *then click on the icon for Add app*
  - *options for Android and iOS native, plus **JavaScript***
- we can take advantage of the provided JavaScript SDK with React Native
- Firebase console will show us a modal with initialisation settings
  - *config settings for adding Firebase usage to our app*

Image - Firebase

initialisation config settings



Add Firebase to your web app

Copy and paste the snippet below at the bottom of your HTML, before other `script` tags.

```
<script src="https://www.gstatic.com/firebasejs/4.7.0/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: "[redacted]",
    authDomain: "egyptian-cards.firebaseapp.com",
    databaseURL: "https://egyptian-cards.firebaseio.com",
    projectId: "egyptian-cards",
    storageBucket: "egyptian-cards.appspot.com",
    messagingSenderId: "[redacted]"
  };
  firebase.initializeApp(config);
</script>
```

Check these resources to learn more about Firebase for web apps:

Get Started with Firebase for Web Apps

Firebase Web SDK API Reference

Firebase Web Samples

Firebase - config settings

React Native - Data - Firebase

add Firebase to React Native - part 2

- start by copying these config values for use with our React Native app
- Firebase runs on a JavaScript thread
  - *certain complex applications, e.g. detailed animations &c.*
  - *may be adversely affected by this structure…*
- might consider using a community package called `react-native-Firebase`
  - *package acts as a wrapper around the Firebase SDK for Android and iOS*
  - *React Native Firebase*
- for most React Native apps we simply integrate Firebase JavaScript SDK
  - *install using NPM or Yarn*

```
npm install firebase --save
```

or

```
yarn add firebase
```

React Native - Data - Firebase

add Firebase to React Native - part 3

- after installing Firebase support for our app
  - *add a new file, `firebase.js`, to a `services` folder in the `src` directory*
- `firebase.js` - specify an initialisation function for working with Firebase services
- working with the initialisation config data provided by Firebase
  - *for the JavaScript SDK for our app*
- need to import the firebase module
  - *then setup a function to handle the initialisation config*

```
import * as firebase from "firebase";

export const initialize = () => firebase.initializeApp({
    apiKey: "__your-api-key__",
    authDomain: "egyptian-cards.firebaseapp.com",
    databaseURL: "https://egyptian-cards.firebaseio.com",
    projectId: "egyptian-cards",
    storageBucket: "egyptian-cards.appspot.com",
    messagingSenderId: "__your-sender-id__"
})
```

React Native - Data - Firebase

add Firebase to React Native - part 4

- need to export the `initialize` function from `firebase.js`
  - *use in a central config file for API usage*
- create a new file for API config management in the `src/services` directory
- config file helps manage multiple services and APIs within a project's structure
- import the `initialize` function for Firebase

```
import { initialize } from './firebase';
```

- then export the functionality for Firebase

```
export const initApi = () => initialize();
```

React Native - Data - Firebase

add Firebase to React Native - part 5

- need to setup Firebase usage in our application root, `App.js`
- use the `componentDidMount` lifecycle hook to call the `initApi()` function
- ensure Firebase is ready and available for our app

```
export default class extends Component {
    componentDidMount() {
        initApi();
    }

    render() {
        return (
          ...
        )
    }
}
```

React Native - Data - Firebase

add Firebase to React Native - part 6

- after setup and initialisation, we can start to consider working with our Firebase database
- benefits of Firebase is that the SDK allows our apps and database to be in sync
  - *as and when updates are registered*
- we need to setup database listeners to ensure the state of our app is updated
  - *whenever a database is modified on Firebase...*
- add such listeners to our `firebase.js` file

```
// setup listener for firebase updates
export const setListener = (endpoint, updaterFn) => {
    firebase.database().ref(endpoint).on('value', updaterFn);
    return () => firebase.database().ref(endpoint).off();
}
```

- using this function to perform two key tasks
- after passing arguments for `endpoint` and `updateFn`
  - *get reference to endpoint for our Firebase database*

```
firebase.database().ref(endpoint)
```

- we can send other required endpoints for our app and Firebase database
  - *such as `cards` in our current example*
- then call the `on()` function allowing us to pass `udpaterFn`
  - *passed as we call the `setListener` function in our app*
- then return a function to allow us to remove the attached listener later in our app

React Native - Data - Firebase

add Firebase to React Native - part 7

- start to use such listeners and functionality in our app
- create a `getCards()` function in `api.js` file
  - *use the `setListener` we created in `firebase.js`*

```
// get cards from current firebase database
export const getCards = (updaterFn) => setListener('cards', updaterFn);
```

- then import this function for a given screen in our app, such as the Card screen,

```
import { getCards } from '../services/api';
```

- then set our state to use this function, and the cards from the database

```
componentDidMount() {
  this.unsubscribeGetCards = getCards((snapshot) => {
    this.setState({
      messages: Object.values(snapshot.val())
    })
  })
}
```

React Native - Data - Firebase

add Firebase to React Native - part 8

- in `componentDidMount()` lifecycle hook
  - *use `Object.values` on Firebase `snapshot.val()`*
  - *`FlatList` component we're using for rendering expects an array*
  - *Firebase returns an object for the values*
- `getCards` is calling `setListener`
  - *returns a function for a remove listener*

```
firebase.database().ref(endpoint).off();
```

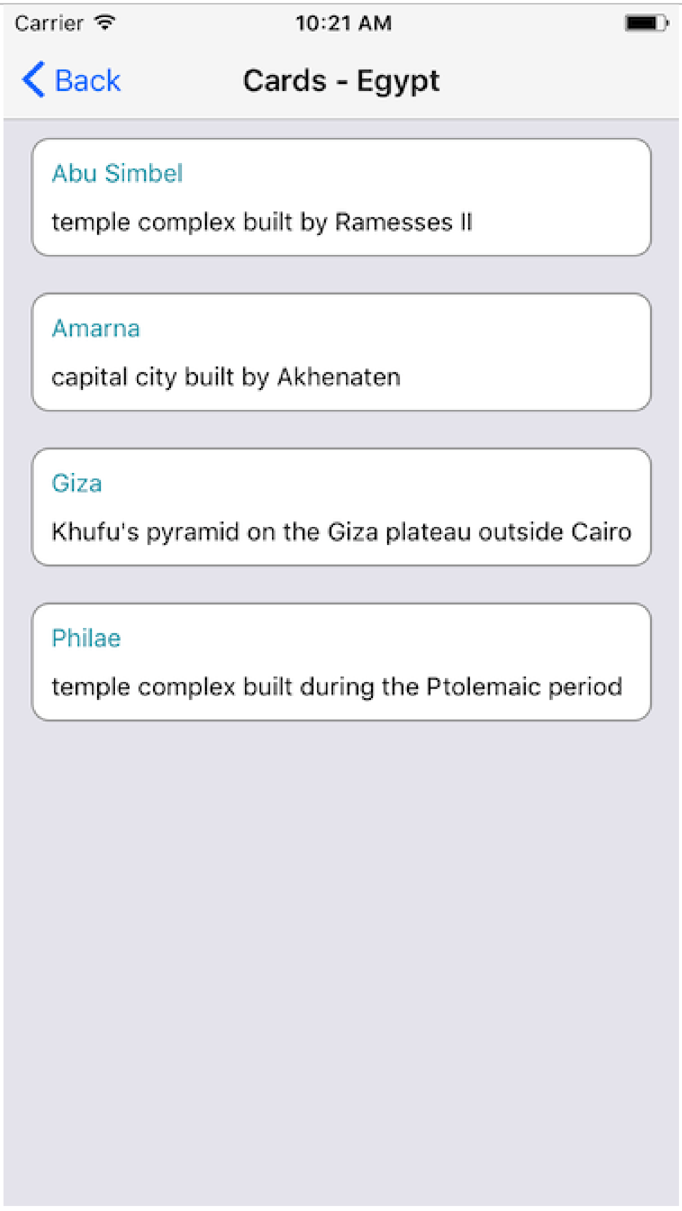
- set the result for `getCards` to `this.unsubscribeGetCards`
- then later call it as necessary in the lifecycle hook for `componentWillUnmount`
- might also add a single call, instead of constantly checking for updates

```
firebase.database().ref(endpoint).once('value')
```

- returns a *promise*
  - *we can use in a standard manner, or chain with `then()`...*

Image - Firebase

render data from database



Carrier 10:21 AM

Back Cards - Egypt

Abu Simbel

temple complex built by Ramesses II

Amarna

capital city built by Akhenaten

Giza

Khufu's pyramid on the Giza plateau outside Cairo

Philae

temple complex built during the Ptolemaic period

Firebase - render data

Cordova & React Native - Data - Firebase

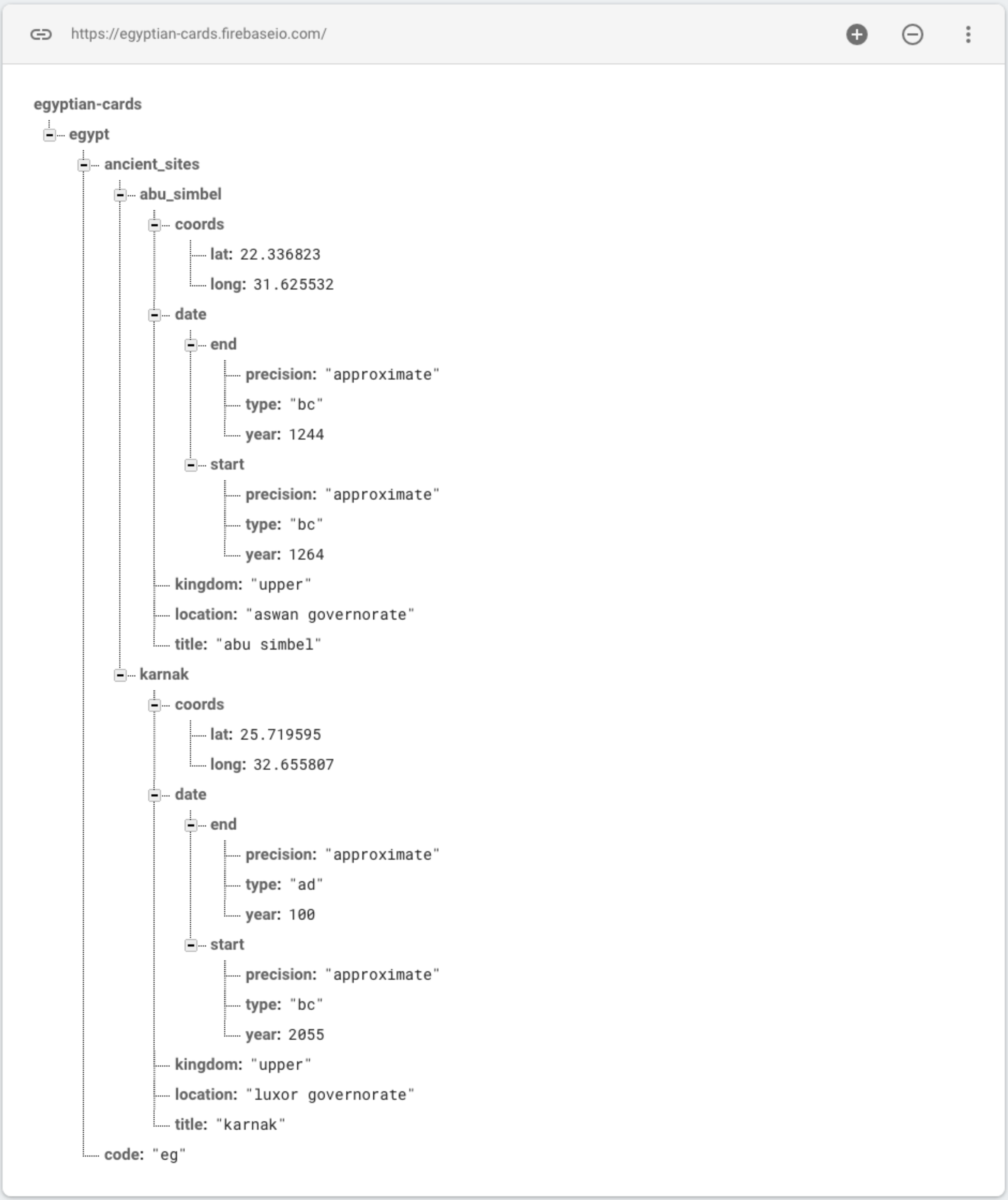
add data with plain JS objects

- plain objects as standard Firebase storage
  - *helps with data updating*
  - *helps with auto-increment pushes of data...*

```
{
  "egypt": {
    "code": "eg",
    "ancient_sites": {
      "abu_simbel": {
        "title": "abu simbel",
        "kingdom": "upper",
        "location": "aswan governorate",
        "coords": {
          "lat": 22.336823,
          "long": 31.625532
        },
        "date": {
          "start": {
            "type": "bc",
            "precision": "approximate",
            "year": 1264
          },
          "end": {
            "type": "bc",
            "precision": "approximate",
            "year": 1244
          }
        }
      },
      "karnak": {
        "title": "karnak",
        "kingdom": "upper",
        "location": "luxor governorate",
        "coords": {
          "lat": 25.719595,
          "long": 32.655807
        },
        "date": {
          "start": {
            "type": "bc",
            "precision": "approximate",
          "year": 2055
        },
        "end": {
          "type": "ad",
          "precision": "approximate",
          "year": 100
        }
      }
    }
  }
 }
}
```

Image - Firebase

JSON import



```
https://egyptian-cards.firebaseio.com/

egyptian-cards
  egypt
    ancient_sites
      abu_simbel
        coords
          lat: 22.336823
          long: 31.625532
        date
          end
            precision: "approximate"
            type: "bc"
            year: 1244
          start
            precision: "approximate"
            type: "bc"
            year: 1264
        kingdom: "upper"
        location: "aswan governorate"
        title: "abu simbel"
      karnak
        coords
          lat: 25.719595
          long: 32.655807
        date
          end
            precision: "approximate"
            type: "ad"
            year: 100
          start
            precision: "approximate"
            type: "bc"
            year: 2055
        kingdom: "upper"
        location: "luxor governorate"
        title: "karnak"
    code: "eg"
```

Firebase - import JSON file

Cordova - Data - Firebase

add to app's `index.html`

- start testing Cordova setup with default config in app's `index.html` file
  - e.g.

```html
<!-- JS - Firebase app -->
<script src="https://www.gstatic.com/firebasejs/5.5.8/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: "YOUR_API_KEY",
    authDomain: "422cards.firebaseapp.com",
    databaseURL: "https://422cards.firebaseio.com",
    projectId: "422cards",
    storageBucket: "422cards.appspot.com",
    messagingSenderId: "282356174766"
  };
  firebase.initializeApp(config);
</script>
```

- example includes initialisation information so the SDK has access to
  - *Authentication*
  - *Cloud storage*
- Realtime Database
- Cloud Firestore

n.b. don't forget to modify the above values to match your own account and database...

Cordova - Data - Firebase

customise API usage

- possible to customise required components per app
- allows us to include only features required for each app
  - *e.g. the only **required** component is*
- `firebase-app` - core Firebase client (required component)

```
<!-- Firebase App is always required and must be first -->
<script src="https://www.gstatic.com/firebasejs/5.5.8/firebase-app.js"></script>
```

- we may add a mix of the following optional components,
- `firebase-auth` - various authentication options
- `firebase-database` - realtime database
- `firebase-firestore` - cloud Firestore
- `firebase-functions` - cloud based function for Firebase
- `firebase-storage` - cloud storage
- `firebase-messaging` - Firebase cloud messaging

Cordova - Data - Firebase

modify JS in app's `index.html`

```html
<!-- Add additional services that you want to use -->
<script src="https://www.gstatic.com/firebasejs/5.5.3/firebase-auth.js"></script>
<script src="https://www.gstatic.com/firebasejs/5.5.3/firebase-database.js"></scr
<script src="https://www.gstatic.com/firebasejs/5.5.3/firebase-firestore.js"></sc
<script src="https://www.gstatic.com/firebasejs/5.5.3/firebase-messaging.js"></sc
<script src="https://www.gstatic.com/firebasejs/5.5.3/firebase-storage.js"></scri

<script src="https://www.gstatic.com/firebasejs/5.5.3/firebase-functions.js"></sc
```

- then define an object for the `config` of the required services and options,

```javascript
var config = {
  // add API key, services &c.
};
firebase.initializeApp(config);
```

Cordova - Data - Firebase

initial app usage - DB connection

- after defining required config and initialisation
  - *start to add required listeners and calls to app's JS*

define DB connection

- we can establish a connection to our Firebase DB as follows,

```
const db = firebase.database();
```

- then use this reference to connect and query our database

Cordova - Data - Firebase

initial app usage - `ref()` method

- with the connection to the database
  - *we may then call the `ref()`, or reference, method*
  - *use this method to read, write &c. data in the database*
- by default, if we call `ref()` with no arguments
  - *our query will be relative to the root of the database*
  - *e.g. reading, writing &c. relative to the whole database*
- we may also request a specific reference in the database
  - *pass a location path, e.g.*

```
db.ref('egypt/ancient_sites/abu_simbel/title').set('Abydos');
```

- allows us to create multiple parts of the Firebase database
- such parts might include,
  - *multiple objects, properties, and values &c.*
- a quick and easy option for organising and distributing data

Cordova & React - Data - Firebase

write data - intro

- also write data to the connected database
  - *again from a JavaScript based application*
- Firebase supports many different JavaScript datatypes, including
  - *strings*
  - *numbers*
  - *booleans*
  - *objects*
  - *arrays*
  - ...
- i.e. any values and data types we add to JSON
  - *n.b. Firebase may not maintain the native structure upon import*
  - *e.g. arrays will be converted to plain JavaScript objects in Firebase*

Cordova & React - Data - Firebase

write data - set all data

- `set` data for the whole database by calling the `ref()` method at the *root*
  - e.g.

```
db.ref().set({
  site: 'abu-simbel',
  title: 'Abu Simbel',
  date: 'c.1264 B.C.',
  visible: true,
  location: {
    country: 'Egypt',
    code: 'EG',
    address: 'aswan'
  }
  coords: {
    lat: '22.336823',
    long: '31.625532'
  }
});
```

Cordova & React - Data - Firebase

write data - set data for a specific data location

- also write data to a specific location in the database
- add an argument to the `ref()` method
  - *specifying required location in the database*
  - e.g.

```
db.ref('egypt/ancient_sites/abu_simbel/location').set('near aswan');
```

- `ref()` may be called relative to any depth in the database from the *root*
- allows us to update anything from whole DB to single property value

Cordova & React - Data - Firebase

Promises with Firebase

- Firebase includes native support for Promises and associated chains.
  - *we do not need to create our own custom Promises*
- we may work with a return Promise object from Firebase
  - *using a standard chain, methods...*
- e.g. when we call the `set()` method
  - *Firebase will return a Promise object for the method execution*
- `set()` method will not explicitly return anything except for success or error
  - *we can simply check the return promise as follows,*

```
db.ref('egypt/ancient_sites/abu_simbel/title')
  .set('Abu Simbel')
  .then(() => {
    // log data set success to console
    console.log('data set...');
  })
  .catch((e) => {
    // catch error from Firebase - error logged to console
    console.log('error returned', e);
  });
```

Cordova & React - Data - Firebase

remove data - intro

- we may lso delete and remove data from the connected database
- various options for removing such data, including
  - *specific location*
  - *all data*
  - `set()` *with* `null`
  - *by updating data*
  - ...

Cordova & React - Data - Firebase

remove data - specify location

- we may also delete data at a specific location in the connected database
  - e.g.

```
db.ref('egypt/ancient_sites/abu_simbel/kingdom')
  .remove()
  .then(() => {
    // log data removed success to console
    console.log('data removed...');
  })
  .catch((e) => {
    // catch error from Firebase - error logged to console
    console.log('error returned', e);
  });
```

Cordova & React - Data - Firebase

remove data - all data

- also remove all of the data in the connected database
  - e.g.

```
db.ref()
  .remove()
  .then(() => {
    // log data removed success to console
    console.log('data removed...');
  })
  .catch((e) => {
    // catch error from Firebase - error logged to console
    console.log('error returned', e);
  });
```

Cordova & React - Data - Firebase

remove data - `set()` with null

- another option specified in the Firebase docs for deleting data
  - *by using `set()` method with a `null` value*
  - e.g.

```
db.ref('egypt/ancient_sites/abu_simbel/kingdom')
  .set(null)
  .then(() => {
    // log data removed success to console
    console.log('data set to null...');
  })
  .catch((e) => {
    // catch error from Firebase - error logged to console
    console.log('error returned', e);
  });
```

Cordova & React - Data - Firebase

update data - intro

- also combine setting and removing data in a single pattern
  - *using the `update()` method call to the defined database reference*
- meant to be used to update multiple items in database in a single call
- we must pass an object as the argument to the `update()` method

Cordova & React - Data - Firebase

update data - existing properties

- to update multiple existing properties
  - e.g.

```
db.ref('egypt/ancient_sites/abu_simbel/').update({
  title: 'The temple of Abu Simbel',
  visible: false
});
```

Cordova & React - Data - Firebase

update data - add new properties

- also add a new property to a specific location in the database

```
db.ref('egypt/ancient_sites/abu_simbel/').update({
  title: 'The temple of Abu Simbel',
  visible: false,
  date: 'c.1264 B.C.'
});
```

- still set new values for the two existing properties
  - *`title` and `visible`*
- add a new property and value for `data`
- `update()` method will only update the specific properties
  - *does not override everything at the reference location*
  - *compare with the `set()` method...*

Cordova & React - Data - Firebase

update data - remove properties

- also combine these updates with option to remove an existing property
  - e.g.

```
db.ref('egypt/ancient_sites/abu_simbel/').update({
  card: null,
  title: 'The temple of Abu Simbel',
  visible: false,
  date: 'c.1264 B.C.',
});
```

- `null` used to delete specific property from reference location in DB
- at the reference loaction in the DB, we're able to combine
  - *creating new property*
  - *updating a property*
  - *deleting existing properties*

Cordova & React - Data - Firebase

update data - multiple properties at different locations

- also combine updating data in multiple objects at different locations
  - *locations relative to initial passed reference location*
  - e.g.

```
db.ref().update({
  'egypt/ancient_sites/abu_simbel/visible': true,
  'egypt/ancient_sites/karnak/visible': false
});
```

- relative to the root of the dabatase
  - *now updated multiple `title` properties in different objects*
- *n.b.* update is only for child objects relative to specified ref location
  - *due to character restrictions on the property name*
  - *e.g. the name may not begin with `.`, / &c.*

Cordova & React - Data - Firebase

update data - Promise chain

- `update()` method will also return a Promise object
  - *allows us to chain the standard methods*
  - e.g.

```
db.ref().update({
  'egypt/ancient_sites/abu_simbel/visible': true,
  'egypt/ancient_sites/karnak/visible': false
}).then(() => {
  console.log('update success...');
}).catch((e) => {
  console.log('error = ', e);
});
```

- as with `set()` and `remove()`
  - *Promise object itself will return success or error for method call*

Cordova & React - Data - Firebase

read data - intro

- fetch data from the connected database in many different ways, e.g.
  - *all of the data*
  - *or a single specific part of the data*
- also connect and retrieve data once
- another option is to setup a listener
  - *used for polling the database for live updates...*

Cordova & React - Data - Firebase

read data - all data, once

- retrieve all data from the database a single time

```
// ALL DATA ONCE - request all data ONCE
// - returns Promise value
db.ref().once('value')
  .then((snapshot) => {
    // snapshot of the data - request the return value for the data at the time o
    const data = snapshot.val();
    console.log('data = ', data);
  })
  .catch((e) => {
    console.log('error returned - ', e);
  });
```

Cordova & React - Data - Firebase

read data - single data, once

- we may query the database once for a single specific value
  - e.g.

```
// SINGLE DATA - ONCE
db.ref('egypt/ancient_sites/abu_simbel/').once('value')
  .then((snapshot) => {
    // snapshot of the data - request the return value for the data at the time o
    const data = snapshot.val();
    console.log('single data = ', data);
  })
  .catch((e) => {
    console.log('error returned - ', e);
  });
```

- returns value for object at the specified location
  - `egypt/ancient_sites/abu_simbel/`

Cordova & React - Data - Firebase

read data - listener for changes - subscribe

- also setup listeners for changes to the connected database
  - *then continue to poll the DB for any subsequent changes*
  - e.g.

```
// LISTENER - poll DB for data changes
// - any changes in the data
db.ref().on('value', (snapshot) => {
  console.log('listener update = ', snapshot.val());
});
```

- `on()` method polls the DB for any changes in `value`
- then get the current snapshot value for the data stored
- any change in data in the online database
  - *listener will automatically execute defined success callback function*

Cordova & React - Data - Firebase

read data - listener for changes - subscribe - error handling

- also add some initial error handling for subscription callback
  - e.g.

```
// LISTENER - SUBSCRIBE
// - poll DB for data changes
// - any changes in the data
db.ref().on('value', (snapshot) => {
  console.log('listener update = ', snapshot.val());
}, (e) => {
  console.log('error reading db', e);
});
```

Cordova & React - Data - Firebase

read data - listener - why not use a Promise?

- as listener is notified of updates to the online database
  - *we need the callback function to be executed*
- callback may need to be executed multiple times
  - *e.g. for many updates to the stored data*
- a Promise may only be resolved a single time
  - *with either `resolve` or `reject`*
- to use a Promise in this context
  - *we would need to instantiate a new Promise for each update*
  - *would not work as expected*
  - *therefore, we use a standard callback function*
- a callback may be executed as needed
  - *each and every time there is an update to the DB*

Cordova & React - Data - Firebase

read data - listener for changes - unsubscribe

- need to *unsubscribe* from all or specific changes in online database
  - e.g.

```
db.ref().off();
```

- removes *all* current subscriptions to defined DB connection

Cordova & React - Data - Firebase

read data - listener for changes - unsubscribe

- also *unsubscribe* a specific subscription by passing callback
  - *callback as used for the original subscription*
- abstract the callback function
  - *pass it to both `on()` and `off()` methods for database `ref()` method*
  - e.g.

```
// abstract callback
const valChange = (snapshot) => {
  console.log('listener update = ', snapshot.val());
};
```

Cordova & React - Data - Firebase

read data - listener for changes - unsubscribe

- then pass this variable as callback argument
  - *for both subscribe and unsubscribe events*
  - e.g.

```
// subscribe
db.ref().on('value', valChange);
// unsubscribe
db.ref().off(valChange);
```

- allows our app to maintain the DB connection
  - *and unsubscribe a specific subscription*

Cordova & React - Data - Firebase

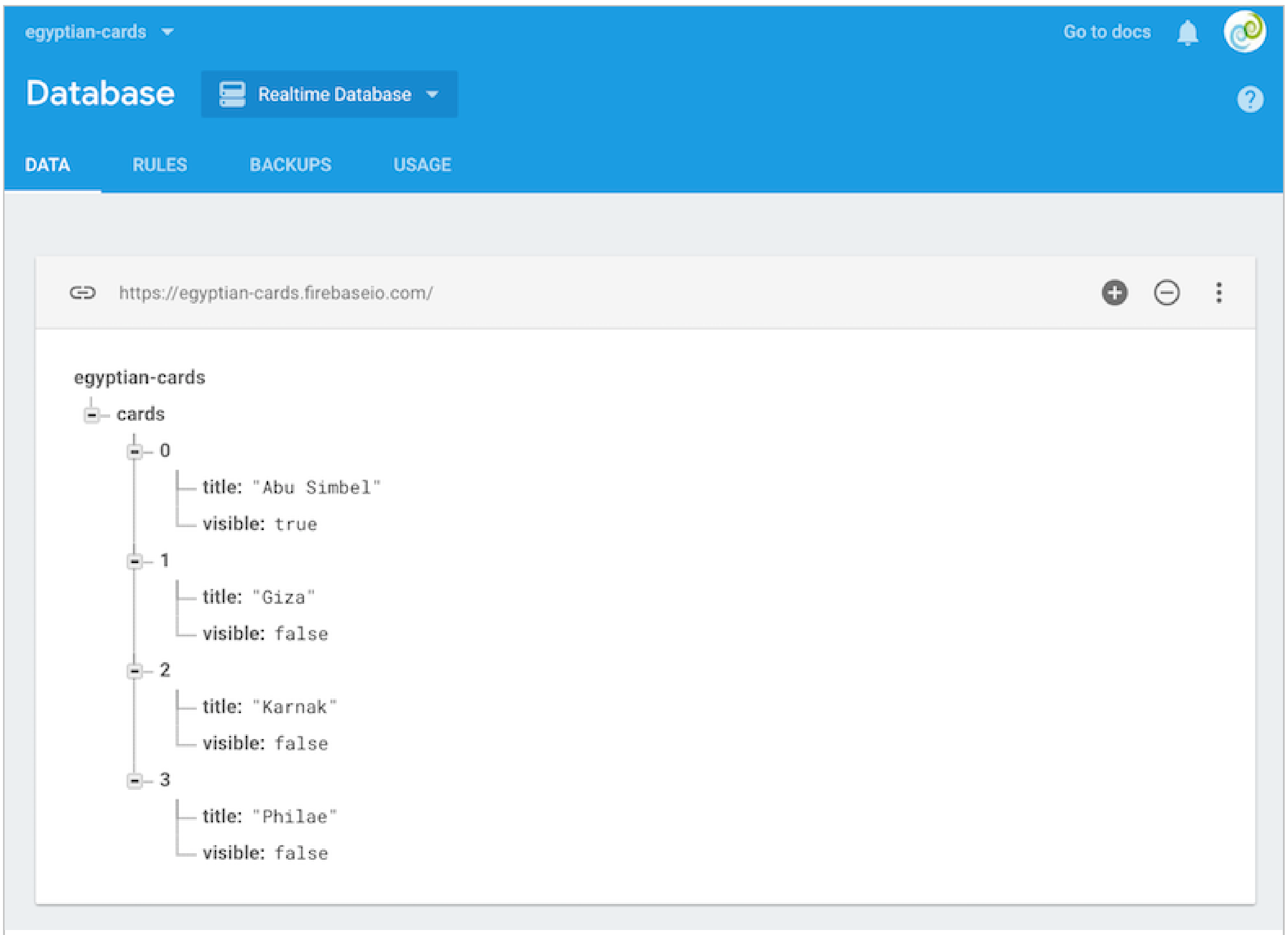
working with arrays

- Firebase does not explicitly support array data structures
  - *converts array objects to plain JavaScript objects*
- e.g. import the following JSON with an array

```
{
  "cards": [
    {
      "visible": true,
      "title": "Abu Simbel",
      "card": "temple complex built by Ramesses II"
    },
    {
      "visible": false,
      "title": "Amarna",
      "card": "capital city built by Akhenaten"
    },
    {
      "visible": false,
      "title": "Giza",
      "card": "Khufu's pyramid on the Giza plateau outside Cairo"
    },
    {
      "visible": false,
      "title": "Philae",
      "card": "temple complex built during the Ptolemaic period"
    }
  ]
}
```

Image - Firebase

JSON import with array



```
egyptian-cards
  cards
    0
      title: "Abu Simbel"
      visible: true
    1
      title: "Giza"
      visible: false
    2
      title: "Karnak"
      visible: false
    3
      title: "Philae"
      visible: false
```

Firebase - import JSON file

Cordova & React - Data - Firebase

working with arrays - index values

- each index value will now be stored as a plain object
  - *with an auto-increment value for the property*
  - e.g.

```
cards: {
  0: {
    card: "temple complex built by Ramesses II",
    title: "Abu Simbel",
    visible: "true"
  }
}
```

Cordova & React - Data - Firebase

working with arrays - access index values

- we may still access each index value from the original array object
  - *without easy access to pre-defined, known unique references*
- e.g. to access the title value of a given card
  - *need to know its auto-generated property value in Firebase*

```
db.ref('cards/0')
```

- reference will be the path to the required object
  - *then access a given property on the object*
- even if we add a unique reference property to each card
  - *still need to know assigned property value in Firebase*

Cordova & React - Data - Firebase

working with arrays - `push()` method

- add new content to an existing Firebase datastore
- we may use the `push()` method to add this data
- a unique property value will be auto-generated for pushed data
  - e.g.

```
// push new data to specific reference in db
db.ref('egypt/ancient_sites/').push({
  "philae": {
    "kingdom": "upper",
    "visible": false
  }
});
```

- new data created with auto-generated ID for parent object
  - e.g.

```
LPcdS31H_u9NOdIn27_
```

- may be useful for dynamic content pushed to a datastore
- e.g. notes, tasks, calendar dates &c.

Cordova & React - Data - Firebase

working with arrays - Firebase `snapshot` methods

- various data snapshot methods in the Firebase documentation
- commonly used method with `snapshot` is the `val()` method
- many additional methods specified in API documentation for *DataSnapshot*
  - *e.g.* `forEach()` *- iterator for plain objects from Firebase*
  - *Firebase Docs - DataSnapshot*

Cordova & React - Data - Firebase

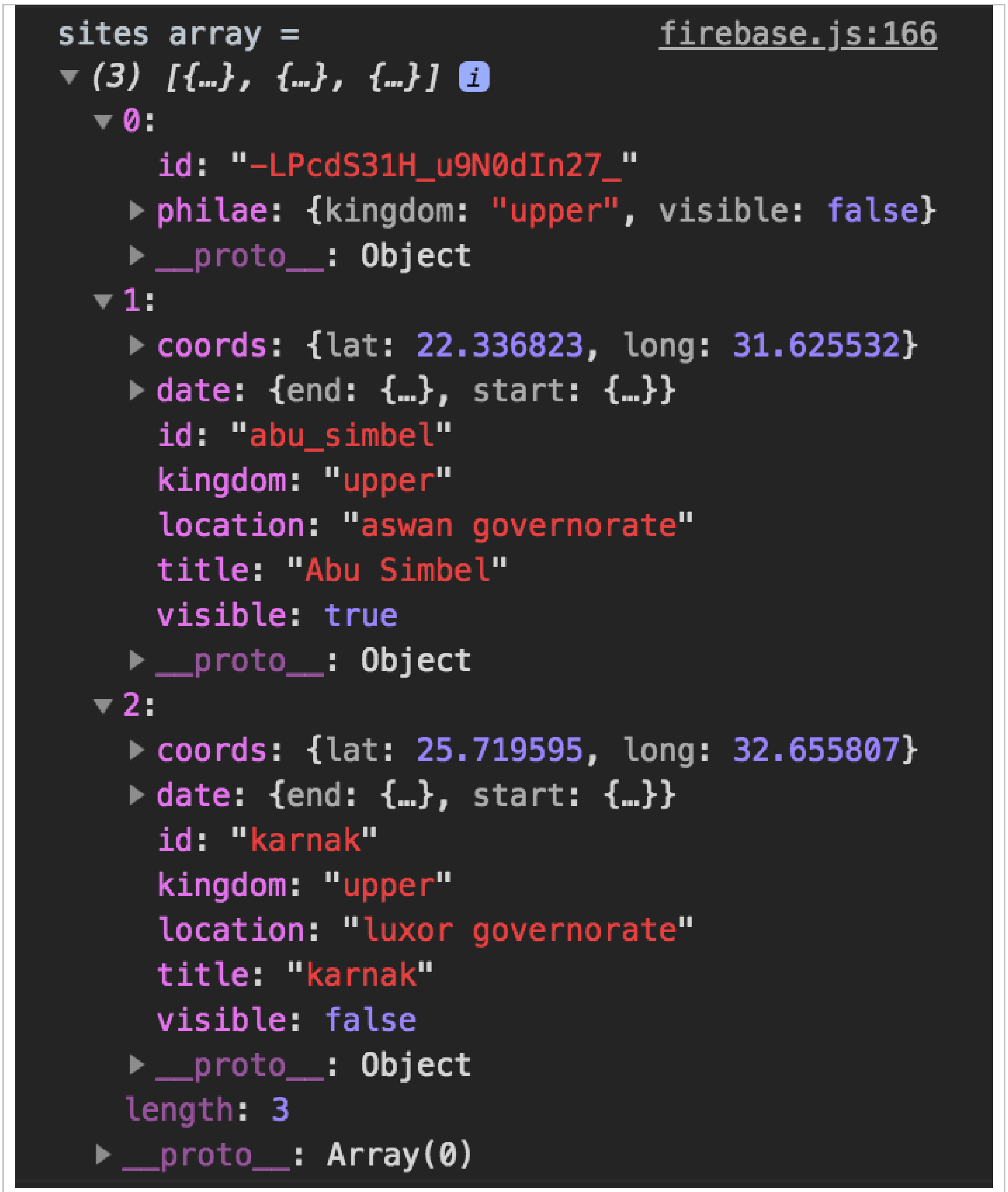
working with arrays - create array from Firebase data

- as we store data as plain objects in Firebase
  - *need to consider how we may work with array-like structures*
  - *i.e. for technologies and patterns that require array data structures*
  - *e.g. Redux*
- need to get data from Firebase, then prepare it for use as an array
- to help us work with Firebase object data and arrays
  - *we may call `forEach()` method on the return `snapshot`*
  - *provides required iterator for plain objects stored in Firebase*
  - 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
db.ref('egypt/ancient_sites')
  .once('value')
  .then((snapshot) => {
    const sites = [];
    snapshot.forEach((siteSnapshot) => {
      sites.push({
        id: siteSnapshot.key,
        ...siteSnapshot.val()
      });
    });
    console.log('sites array = ', sites);
  });
```

Image - Firebase

snapshot `forEach()` - creating a local array



```
sites array =                        firebase.js:166
▼ (3) [{…}, {…}, {…}] i
  ▼ 0:
      id: "-LPcdS31H_u9N0dIn27_"
    ▶ philae: {kingdom: "upper", visible: false}
    ▶ __proto__: Object
  ▼ 1:
    ▶ coords: {lat: 22.336823, long: 31.625532}
    ▶ date: {end: {…}, start: {…}}
      id: "abu_simbel"
      kingdom: "upper"
      location: "aswan governorate"
      title: "Abu Simbel"
      visible: true
    ▶ __proto__: Object
  ▼ 2:
    ▶ coords: {lat: 25.719595, long: 32.655807}
    ▶ date: {end: {…}, start: {…}}
      id: "karnak"
      kingdom: "upper"
      location: "luxor governorate"
      title: "karnak"
      visible: false
    ▶ __proto__: Object
    length: 3
  ▶ __proto__: Array(0)
```

Firebase - local array

- we now have a local array from the Firebase object data
  - *use with options such as Redux...*

Cordova & React - Data - Firebase

add listeners for value changes

- as we modify objects, properties, values &c. in Firebase
  - *set listeners to return notifications for such updates*
  - *e.g. add a single listener for any update relative to full datastore*

```
// LISTENER - SUBSCRIBE - v.2
// - get all data & then push return data to local array...
db.ref('egypt').on('value', (snapshot) => {
  const sites = [];
  snapshot.forEach((siteSnapshot) => {
    sites.push({
      id: siteSnapshot.key,
      ...siteSnapshot.val()
    });
  });
  console.log('sites array after update = ', sites);
});
```

- the `on()` method does not return a Promise object
  - *we need to define a callback for the return data*

References

- React Native
  - *React DevTools*
  - *React Native - Layout Props*
  - *React Native - StatusBar*
- 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*