Intro to Firebase

based on slides by David Herrera

Resources

- Reading and Writing to Firebase
- Retrieving Data
- API Reference

Firebase Basics

What is Firebase?

- Firebase is a "Mobile Backend as a Service" (MBaaS) platform that offers many features that are common to modern apps.
- Firebase operates on the "freemium" model (free up to a certain amount of usage, you pay per amount above the threshold).
- Bought by Google in 2015.
- Currently well integrated into Google developer services.
- Firebase is essentially in charge of cloud messaging now.

Google

Build better apps

Cloud Firestore
Store and sync app data at global scale

Firebase ML BETA
Machine learning for mobile developers

Cloud Functions
Run mobile backend code without managing servers

Authentication
Authenticate users simply and securely

Hosting
Deliver web app assets with speed and security

Cloud Storage
Store and serve files at Google scale

Realtime Database
Store and sync app data in milliseconds

Improve app quality

Crashlytics
Prioritize and fix issues with powerful, realtime crash reporting

Performance Monitoring
Gain insight into your app's performance

Test Lab
Test your app on devices hosted by Google

App Distribution BETA
Distribute pre-release versions of your app to your trusted testers

Grow your business

In-App Messaging BETA
Engage active app users with contextual messages

Google Analytics
Get free and unlimited app analytics

Predictions
Smart user segmentation based on predicted behavior

A/B Testing BETA
Optimize your app experience through experimentation

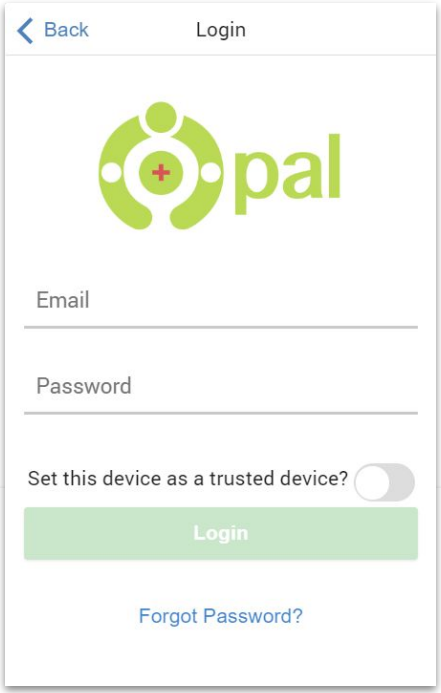
Cloud Messaging
Send targeted messages and notifications

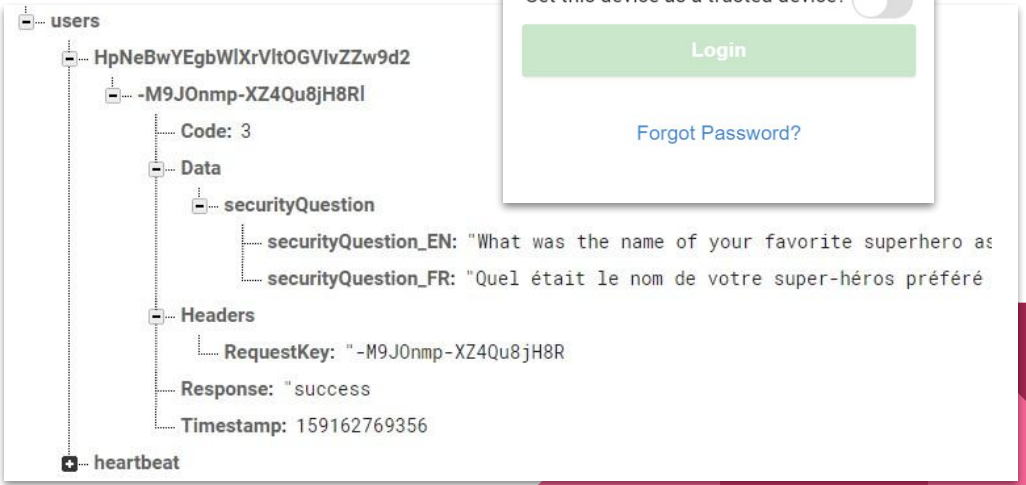
Remote Config
Modify your app without deploying a new version

Dynamic Links
Drive growth by using deep links with attribution

Firebase in Opal

- Authentication
- Realtime Database
- Crashlytics
- App Distribution
- Cloud Messaging





Getting Started with Firebase

- Firebase is accessible via a Google account.
- Create a new project for a "realtime database".
- Add Firebase to your web project:
  - Head to Project Settings / Your Apps.
  - Select the web platform (HTML bracket icon).
  - Follow the setup instructions.
  - Copy-and-paste the configurations into your app.
- More detailed instructions can be found here: Add Firebase to your JavaScript project, and in the Firebase assignment instructions.

The AngularJS / Webpack Way

1. **npm install firebase angularfire --save**
2. Make firebase available everywhere using webpack's ProvidePlugin:

```
plugins: [
    new webpack.ProvidePlugin( definitions: {
        firebase: "firebase",
    })
]
```

3. In **app.js**, import `firebase from "firebase"`, and add your firebase configurations (used to initialize the connection).
4. If there are no errors, Firebase should be available globally in your workspace.

Firebase Configurations

```
// Initialize Firebase
// TODO: Replace with your project's customized code snippet
var config = {
  apiKey: "<API_KEY>",
  authDomain: "<PROJECT_ID>.firebaseapp.com",
  databaseURL: "https://<DATABASE_NAME>.firebaseio.com",
  projectId: "<PROJECT_ID>",
  storageBucket: "<BUCKET>.appspot.com",
  messagingSenderId: "<SENDER_ID>",
};
firebase.initializeApp(config);
```

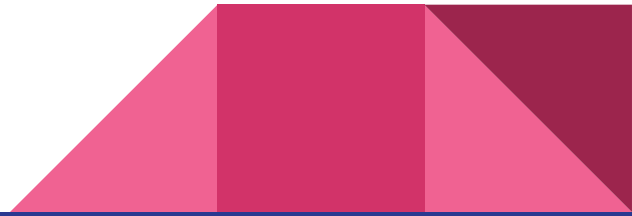
Firebase Data Structure

- Firebase is a **no-SQL** database based on a JSON format.
- There are **key/value** pairs at each level.
- We can add data to Firebase given a value based on a **key.**
- We can listen to changes in an specific **key's sub-tree.**
- The structure of Firebase DB is **not normalized.**

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

Firebase References

- In Firebase, we use **references**, which are paths in the database relative to the root.
- We attach **listeners** to references.
- We use references to write to the database under a given path.
- We can provide a few basic filters or orderings when querying those references.



Firebase References

```
var ref = firebase.database().ref();
var refMessages = ref.ref("messages");
var ref2Messages = firebase.database().ref("messages");
var refUsers = firebase.database().ref("users").orderByKey();
var refMessages = firebase.database().ref("messages")
            .orderByChild("lastMessageDate");
// Querying type example
ref.child("users").orderByChild("userId")
            .equalTo("54ca2c11d1afc1612871624a").limitToFirst(1);
```

Reading & Writing

Firebase Reading

- To read data, we instantiate a **listener** at a given path.
- There are two **types of listener**:
  - once
  - on
- There are several **events:**
  - value
  - child_added
  - child_changed
  - child_removed
  - child_moved

General syntax

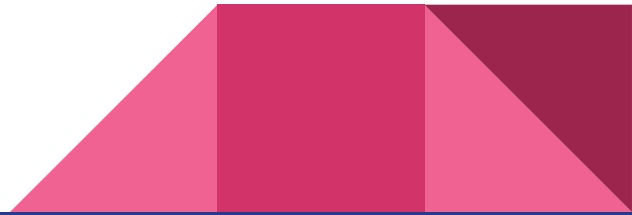
```
var ref = firebase.database().ref("messages");
ref.<type-of-listener>(<type-of-event>), function(snapshot){
        if(snapshot.exists())
        {
                var val = snapshot.value();
                var key = snapshot.key();
                console.log(val, key);
        }
});
```

Example

```
ref.on("child_changed",function(snapshot){
        if(snapshot.exists())
        {
                var val = snapshot.value();
                var key = snapshot.key();
                console.log(val, key);
        }
}
```

Firebase Reading—Types of Listeners

- **ref.on(“<event-type>”,function(){})**
  - Gives an update any time there is a change to the path corresponding to the specified event type.
  - To disconnect all listeners from a path (ref), we use ref.off().
  - A single listener can be disconnected from a path by passing it as input to ref.off().
- **ref.once(“<event-type>”,function(){})**
  - Only listens to the reference until the first change.
  - After it fetches the first time, the reference is disconnected.



Firebase Reading—Event Types

Event	Typical usage
`value`	Used to read a static snapshot of the contents at a given database path. It is triggered once with the initial data and again every time the data changes. The event callback is passed a snapshot containing all data at that location, including child data.
`child_added`	Typically used when retrieving a list of items from the database. Unlike value which returns the entire contents of the location, child_added is triggered once for each existing child and then again every time a new child is added to the specified path. The event callback is passed a snapshot containing the new child's data. For ordering purposes, it is also passed a second argument containing the key of the previous child.
`child_changed`	Triggered any time a child node is modified. This includes any modifications to descendants of the child node. It is typically used in conjunction with child_added and child_removed to respond to changes to a list of items. The snapshot passed to the event callback contains the updated data for the child.

Firebase Reading—Event Types

Event	Typical usage
`child_removed`	Triggered when an immediate child is removed. It is typically used in conjunction with child_added and child_changed. The snapshot passed to the event callback contains the data for the removed child.
`child_moved`	Used when working with numerically indexed ordered data. This event will be triggered when a child's sort order changes such that its position relative to its siblings changes. The DataSnapshot passed to the callback will be for the data of the child that has moved. It is also passed a second argument which is a string containing the key of the previous sibling child by sort order, or null if it is the first child.

Reference: Retrieving Data

Firebase Writing

- Adds or replaces data at the given path.
- Three types:
  - update
  - push
  - set

```
// Creates a random string as key and pushes request, then listens to that
// key for updates
var newPostRef = ref.child("request").push({"requesType":"GetConversatios"});
var key = newPostRef().key;
newPostRef.child("users"+'/'+key).once("value",function(){});
// Updates the sub-tree
refConversation.child(idConversation).update({"lastMessage":...});
// Overwrites the sub-tree
refConversation.child(idConversation).set({"lastMessage":...});
// Deletes the sub-tree
refConversation.child(idConversation).set(null);
```

Firebase Writing

```
// Creates a random string as key and pushes request, then listens to that
// key for updates
var newPostRef = ref.child("request").push({"requesType":"GetConversatios"});
var key = newPostRef().key;
newPostRef.child("users"+'/'+key).once("value",function(){});
// Updates the sub-tree
refConversation.child(idConversation).update({"lastMessage":...});
// Overwrites the sub-tree
refConversation.child(idConversation).set({"lastMessage":...});
// Deletes the sub-tree
refConversation.child(idConversation).set(null);
```

- **Push:** Uses a randomly generated string as key, and writes to it the value provided.
- **Update:** Does not overwrite the entire sub-tree, simply adds or overwrites the keys provided.
- **Set:** Overwrites sub-tree pointed to by reference completely (all keys and values previously set are deleted).

Demo