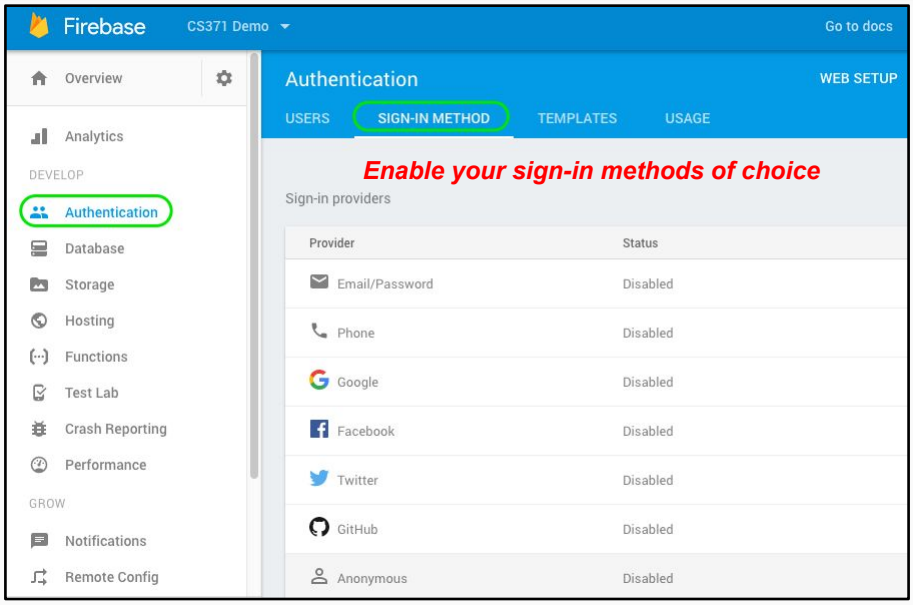
Firebase User Authentication

Authentication Options

- Email/Password
- Facebook accounts
- GitHub accounts
- Google accounts
- Twitter accounts
- Phone numbers
- Online documentation: `firebase.auth`

Authentication Dashboard



Logging in is an *async* process!

After a login call, your function continues running to the next instruction, but the login result arrives later in the future

Asynchronous Function Calls

- Asynchronous function calls do not produce the result when the function returns to its caller
  - The result is delivered at a later time, usually via a callback function
  - Examples: XMLHttpRequest **onload** handler

Async example: Event Listener and AJAX loader

```
<!-- button handling -->
var saveBtn = document.getElementById(“save”);
var loadBtn = document.getElementById(“load”);

saveBtn.addEventListener(‘click’, () => { /* your code #1 here */ } );
loadBtn.addEventListener(‘click’, () => { /* your code #2 here */ } );

console.log(“AAA”);
```

```
<!-- button handling -->
var xhr = new XMLHttpRequest();
xhr.onload = function(e) {
  /* your code #3 here */
};

xhr.open(__________);
xhr.send();
```

Deeply Nested Callbacks?

```
var xhr = new XMLHttpRequest();
xhr.onload = function(e) {
  if (xhr.status === 200) {
     // make a second request based on the content in xhr.responseText?

     var url2 = getUrl (xhr.resposeText);
     var xhr2 = new XMLHttpRequest();
     xhr2.onload = function(e2) {
       // this is a nested callback !!!
     };
     xhr2.open(_______);
     xhr2.send();
     //
  }
};

xhr.open(__________);
xhr.send();
```


Deeply Nested Callbacks

- 2-level nesting already looks bad
- Deeper nesting looks like callback pyramid
- More serious problem
  - How do you recover from error if one of the (inner) callbacks fails
- Solution: Java Promises

JavaScript Promises

- A (human) promise = an *action* that will happen in the *future*
  - Usually involves two parties (the promiser/producer and promisee/consumer)
- A promise = a handle to a *function result* to be delivered in the *future*
- States of a promise
  - **Pending**: the function is still working and result has not been delivered
  - **Fulfilled**: the function has finished its operation and the the result is delivered
  - **Rejected**: the function failed to complete its operation

Producer: Delivering a promise

Producing and Consuming Promises

```
// Promise constructor takes a function with two args
new Promise ( function (resolve, reject) {    } );
```

```
// producer
function myProducerExample(a, b) {
  var prom = new Promise( (resolve, reject) =>
    {
      if (b != 0)
        resolve(a / b);
      else
        reject("Attempt to divide by 0");
    });

  return prom;
}
```

```
// consumer
myProducerExample(6, 2)
  .then (res => {
    console.log("Result is ", res);  // output 3
  })
  .catch (err => {
    console.log ("Got an error", err);
  });
```


Email/Password Authentication

- createUserWithEmailAndPassword()
- signInWithEmailAndPassword()
- signOut()
- onAuthStateChanged()

Email/Password Authentication (1)

```
<script>
  // AFTER your app is initialized
  firebase.auth().createUserWithEmailAndPassword(email, password)
    .catch( error => {
      console.log(error.message);
    });
</script>
```

```
<script>
  // AFTER your app is initialized
  // this will CREATE and LOGIN
  firebase.auth().signInWithEmailAndPassword(email, password)
    .catch( error => {
      console.log(error.message);
    });
</script>
```

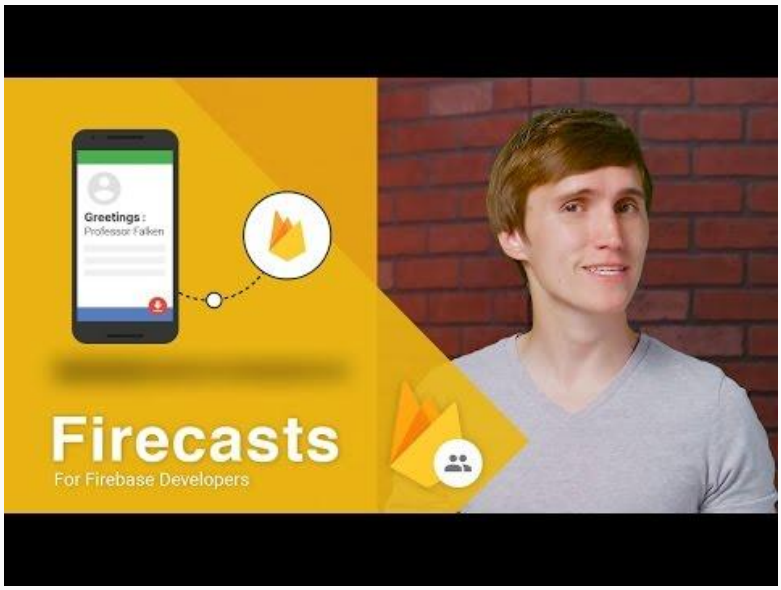

Email/Password Authentication (2)

More detailed info on user attributes: firebase.auth.User

```
<script>
  // AFTER your app is initialized
  firebase.auth().onAuthStateChanged( user => {
    if (user != null) {            // user is NOT logged in
      // do necessary work to clean up your app
    } else {                       // user is logged in
      console.log("User email is ", user.email);
    }
  });
</script>
```

```
<script>
  firebase.auth().signOut();
</script>
```

Email/Password Authentication



3rd party Account Providers

- Account Providers: Facebook, Google, Twitter, GitHub, ….
- Use FirebaseUI for Web

Using FirebaseUI (1)

2-letter language code

```
<!-- add this to the header of your HTML page -->
<script src="https://www.gstatic.com/firebasejs/ui/2.4.0/firebaseui-auth__en.js">
</script>
<link type="text/css" rel="stylesheet"
   href="https://www.gstatic.com/firebasejs/ui/2.4.0/firebase-ui-auth.css" />
```


Using FirebaseUI (2a)

```
<!-- add this JavaScript in the body -->
var uiConfig = {
  signInSuccessUrl: "./logged-in.html",
  signInOptions: [
    firebase.auth.GoogleAuthProvider.PROVIDER_ID,
    firebase.auth.EmailAuthProvider.PROVIDER_ID       /* as many as you need */
  ]
};
var ui = new firebaseui.auth.AuthUI (firebase.auth());
ui.start ('#auth-container', uiConfig);
```

```
<!-- in welcome.html -->
<body>
  <div id="auth-container"></div>
</body>
```

Using FirebaseUI (2b): Single Page Apps (SPAs)

```
<!-- in main.html -->
<body>
  <div id="non-auth-users">
    <!-- UI for non-authenticated user goes here -->
    <div id="auth-container"></div>
  </div>

  <div id="auth-users">
    <!-- UI for authenticated user goes here →
    You are a logged in user
  </div>
  <script src="main.js"></script>
</body>
```

Defined two s

- *For non-authenticated users*
- *For authenticated users*

but hide one of them depending on the user login state.

```
var fbConfig = { apiKey: "__________",  /* and more */ };
var uiConfig = {
  signInSuccessUrl: "./main.html", /* and more */ };

firebase.initializeApp(fbConfig);

firebase.auth().onAuthStateChanged ( user => {
  var nonau = document.getElementById("non-auth-users");
  var au = document.getElementById("auth-users");

  if (user == null) {
    var ui = new firebaseui.auth.AuthUI (firebase.auth());
    ui.start ('#auth-container', uiConfig);
    au.hidden = true;
    nonau.hidden = false;
  } else {
    au.hidden = false;
    nonau.hidden = true;
  }
});
```