Build your first Firebase powered Ionic 2 app.

Using the JavaScript SDK

Jorge Vergara

Contents

Thank you

There are 3 people I want to thank.

My wife, Evelyn, you push me to be a better person and to work harder every single day, I wouldn't be here without you.

My son, Emmanuel, you're the motivation for everything I do.

You, yeah, you, the one reading this, without your support, none of these could be possible.

Chapter 1

Preparation

1.1 Introduction

With the recent move to Ionic 2 + Firebase 3 I've gotten a lot of people asking me both in the comments and emails about problems they are having trying to build apps.

And those problems are always the same, they don't know how to retrieve data from firebase and display it in an Ionic Page, or they are having trouble with one of the auth functions.

So I decided to address all those issues in this guide, building a complete data driven app with Firebase.

I'm going to go through all the problems you've been having while building this app.

1.2 What we'll build

We are going to be building an app for managing events.

I've been thinking a lot about what to make, and I decided for this because I actually built my wife's startup with Ionic 1 and it's all about managing events for a very specific niche.

Since I know I'm going to start updating it soon to be an Ionic 2 app I decided to make a lite version of it for this tutorial.

The main things the app will do is:

- Allow event manager to create an account, login, reset password (you know, all the auth stuff).
- Allow for editing the user profile: changing passwords, emails, etc.
- Create events, people.
- Add people to event's guest lists.
- Calculate revenue for the event.
- Take pictures of guests and add them to Firebase Storage.
- Be secure: Meaning make sure that only the person who owns the account can see the data.

You can find the entire source code in this Github repository.

1.3 Is your development environment up to date?

Before writing any code, we are going to take a few minutes to install everything you need to be able to build this app, that way you won't have to be

switching context between coding and installing.

The first thing you'll do is install node.js make sure you get version 6.x, even tho V4 works for most users it will install the wrong version of typings and a lot of things will break in your app.

The second thing you'll do is make sure you have Ionic, Cordova and Typescript installed, you'll do that by opening your terminal and typing:

```
$ npm install -g ionic cordova typescript
```

Depending on your operating system (mostly if you run on Linux or Mac) you might have to add **sudo** before the **npm install....** command.

One cool thing to note here is that the Ionic CLI is out of beta. If you aren't new to Ionic you'll remember that we used to install **ionic@beta** instead.

Also, since Ionic released V2RC0 you don't need to install typings anymore, they use **npm @types** now.

1.4 Create the App

Now that you made sure everything is installed and up to date, you'll create a new Ionic 2 app.

For this you just need to (*while still in your terminal*) navigate to the folder you'd like to create it in.

For me, it's my Development folder in my ∼/ directory:

```
$ cd Development
$ ionic start eventTutorial blank --v2
$ cd eventTutorial
```

What those lines there do is the following:

- First, you'll navigate to the Development folder.
- Second, you'll create the new Ionic 2 app:
  - `ionic start` creates the app.
  - `eventTutorial` is the name we gave it.
  - `blank` tells the Ionic CLI you want to start with the blank template.
  - `-v2` tells the Ionic CLI you want to create an Ionic 2 project instead of an Ionic 1 project.
- Third, you'll navigate into the new `eventTutorial` folder, that's where all of your app's code is going to be.

From now on, whenever you are going to type something in the terminal it's going to be in this folder, unless I say otherwise.

1.5 The `npm` packages that come with the project

When you use the Ionic CLI to create a new project, it's going to do a lot of things for you, one of those things is making sure your project has the necessary `npm` packages/modules it needs.

That means, the `start` command is going to install **ionic-angular** all of the **angularjs** requirements and more, here's what the `package.json` file would look like:

```
{
  "name": "ionic-hello-world",
  "author": "Ionic Framework",
  "homepage": "http://ionicframework.com/",
  "private": true,
  "scripts": {
    "ionic:build": "ionic-app-scripts build",
    "ionic:serve": "ionic-app-scripts serve"
  },
  "dependencies": {
    "@angular/common": "2.2.1",
    "@angular/compiler": "2.2.1",
    "@angular/compiler-cli": "2.2.1",
    "@angular/core": "2.2.1",
    "@angular/forms": "2.2.1",
    "@angular/http": "2.2.1",
    "@angular/platform-browser": "2.2.1",
    "@angular/platform-browser-dynamic": "2.2.1",
    "@angular/platform-server": "2.2.1",
    "@ionic/storage": "1.1.7",
    "ionic-angular": "2.0.0-rc.4",
    "ionic-native": "2.2.11",
    "ionicons": "3.0.0",
    "rxjs": "5.0.0-beta.12",
    "zone.js": "0.6.26"
  },
  "devDependencies": {
    "@ionic/app-scripts": "0.0.48",
    "typescript": "^2.0.3"
  },
  "description": "event-tutorial: An Ionic project",
  "cordovaPlugins": [
    "cordova-plugin-device",
    "cordova-plugin-console",
    "cordova-plugin-whitelist",
    "cordova-plugin-splashscreen",
    "cordova-plugin-statusbar",
    "ionic-plugin-keyboard"
  ],
  "cordovaPlatforms": []
}
```

Depending on when you read this, these packages might change (*specially version numbers*) so keep that in mind, also you can email me at j@javebratt.com if you have any questions/issues/problems with this.

1.6 Install the packages you'll need

For this project to work you'll need to install a couple of extra packages.

First, we'll need to update `@ionic/app-scripts` to use the latest version, since that's the one with Webpack support, meaning it works with Firebase out of the box, you won't have to create custom builds.

```
$ npm install @ionic/app-scripts@latest --save
```

That will tell Ionic to use the latest version of the module and you'll get webpack support.

And then you'll be ready to go, now you need to install Firebase (*hey, that's what this post is all about :P*)

Open your terminal (*you should already be in the project folder*) and install the packages in this order:

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

1.7 Import

Now you can import everything you'll need inside `src/app/app.module.js` so we don't have to keep going back to that file:

NOTE: Since you now need to declare every component, page, provider, etc in this file, this would be a good moment to create them and add them all, just go ahead and run this commands in your terminal:

```
$ ionic generate page EventDetail
$ ionic generate page EventCreate
$ ionic generate page EventList
$ ionic generate page Login
$ ionic generate page Profile
$ ionic generate page ResetPassword
$ ionic generate page Signup
```

That will generate all the pages we need, and then:

```
$ ionic generate provider EventData
$ ionic generate provider AuthData
$ ionic generate provider ProfileData
```

That will generate all the providers we'll use, don't worry if you don't understand something from those pages, we'll explain every single one of those in detail when we get to them.

Now, you can open your **app.module.ts** and import everything we'll be using, this is the only time you'll see this file :)

```
import { NgModule, ErrorHandler } from '@angular/core';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';

// Import pages
import { HomePage } from '../pages/home/home';
import { EventCreatePage } from '../pages/event-create/event-create';
import { EventDetailPage } from '../pages/event-detail/event-detail';
import { EventListPage } from '../pages/event-list/event-list';
import { LoginPage } from '../pages/login/login';
import { ProfilePage } from '../pages/profile/profile';
import { ResetPasswordPage } from '../pages/reset-password/reset-password';
import { SignupPage } from '../pages/signup/signup';

// Import providers
import { AuthData } from '../providers/auth-data';
import { EventData } from '../providers/event-data';
import { ProfileData } from '../providers/profile-data';
```

And then add the initialize to **@NgModule**:

```
@NgModule({
  declarations: [
    MyApp,
    HomePage,
    EventCreatePage,
    EventDetailPage,
    EventListPage,
    LoginPage,
    ProfilePage,
    ResetPasswordPage,
    SignupPage
  ],
  imports: [
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    HomePage,
    EventCreatePage,
    EventDetailPage,
    EventListPage,
    LoginPage,
    ProfilePage,
    ResetPasswordPage,
    SignupPage
  ],
  providers: [
    {provide: ErrorHandler, useClass: IonicErrorHandler},
    AuthData,
    EventData,
    ProfileData
  ]
})
export class AppModule {}
```

In the end the file should look like this:

```
import { NgModule, ErrorHandler } from '@angular/core';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';

// Import pages
import { HomePage } from '../pages/home/home';
import { EventCreatePage } from '../pages/event-create/event-create';
```

```
import { EventDetailPage } from '../pages/event-detail/event-detail';
import { EventListPage } from '../pages/event-list/event-list';
import { LoginPage } from '../pages/login/login';
import { ProfilePage } from '../pages/profile/profile';
import { ResetPasswordPage } from '../pages/reset-password/reset-password';
import { SignupPage } from '../pages/signup/signup';

// Import providers
import { AuthData } from '../providers/auth-data';
import { EventData } from '../providers/event-data';
import { ProfileData } from '../providers/profile-data';

@NgModule({
  declarations: [
    MyApp,
    HomePage,
    EventCreatePage,
    EventDetailPage,
    EventListPage,
    LoginPage,
    ProfilePage,
    ResetPasswordPage,
    SignupPage
  ],
  imports: [
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    HomePage,
    EventCreatePage,
    EventDetailPage,
    EventListPage,
    LoginPage,
    ProfilePage,
    ResetPasswordPage,
    SignupPage
  ],
  providers: [
    {provide: ErrorHandler, useClass: IonicErrorHandler},
    AuthData,
    EventData,
    ProfileData
  ]
})
export class AppModule {}
```

Right there your app should be able to run without any errors when you do

`ionic serve`

If you are running intro trouble getting this to work you can message me and I'll help you troubleshoot. If this is an important production project I'd advise you use the JS SDK instead.

1.8 Install the cordova plugins you'll use

For this tutorial you'll need to install cordova plugins to work with your phone's native capabilities.

Actually you'll need just one, we are going to use the phone's camera to take pictures of our guests and securely upload them to Firebase Storage.

For that you'll need to install the camera plugin from `ionic-native`

It's just as easy as opening your terminal and typing:

```
$ ionic plugin add cordova-plugin-camera
```

That's it, you now have installed everything you'll need to build this app, including npm packages, cordova plugins.

Right now you should have the skeleton of your app and it should run. Go ahead and try it, open the terminal and type:

```
$ ionic serve
```

The app should run in the browser without any errors showing you the HomePage with a placeholder that says something like "*the world is your oyster and go to the docs*"

If it doesn't run copy the stack trace or take screen-shots and shoot me an email so I can help you debug what's going wrong.

1.9 Next Steps

When you are ready move to the next chapter, there you'll initialize your app and create the auth module.

Chapter 2

Authentication

2.1 Introduction

If you've ever built your own authentication system you know it can be a pain, setting up secure servers, building the entire back-end, it can take a while, when all you really want is to focus on making your app great.

That right there is the main reason I chose Firebase as the back-end for every single app I make, and with their latest announcement at I/O 16 (Firebase 3) I know I made the right choice.

Today I want to show how to integrate Firebase 3 email auth system with your Ionic 2 app, after all, you are building more than to-do lists now and need to be able to handle on-line users.

That's enough for an intro so let's get down to business.

2.2 Ionic 2 and Firebase 3 Email Auth

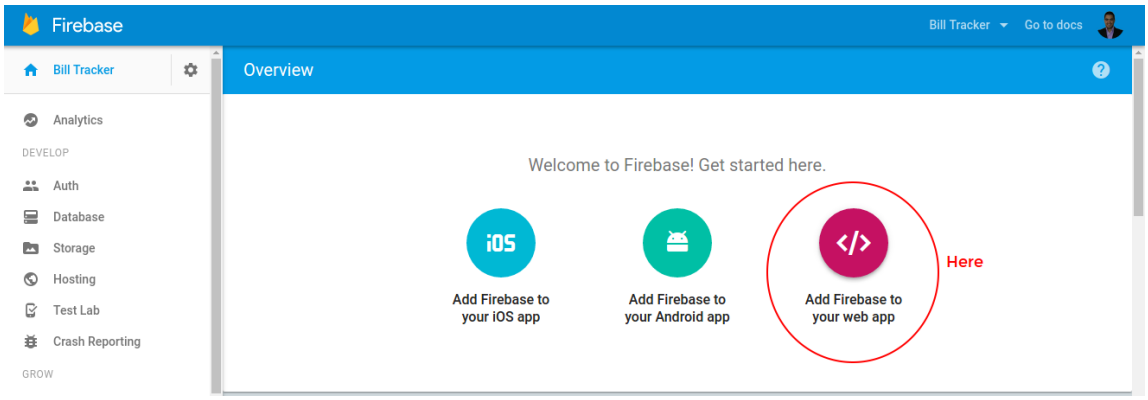
Let's start working on some code, the first thing you'll need to do is go to your `app.component.ts` and initialize Firebase, for that you'll first import Firebase:

```
import firebase from 'firebase';
```

And then you'll initialize it inside the constructor:

```
firebase.initializeApp({
  apiKey: "",
  authDomain: "",
  databaseURL: "",
  storageBucket: "",
  messagingSenderId: ""
});
```

You can get that data inside your Firebase's Console:



In the end, that file should look like this:

```
import { Component } from '@angular/core';
import { Platform } from 'ionic-angular';
import { StatusBar } from 'ionic-native';

import { HomePage } from '../pages/home/home';

import firebase from 'firebase';


@Component({
  template: `<ion-nav [root]="rootPage"></ion-nav>`
})
export class MyApp {
  rootPage: any;

  constructor(platform: Platform) {
    firebase.initializeApp({
      apiKey: "",
      authDomain: "",
      databaseURL: "",
      storageBucket: "",
      messagingSenderId: ""
    });

    platform.ready().then(() => {
      // Okay, so the platform is ready and our plugins are available.
      // Here you can do any higher level native things you might need.
      StatusBar.styleDefault();
    });
  }
}
```

Now that Firebase is initialized, were going to create an auth observer, so we can listen to any authentication changes, and take a user to a different page when logging out.

For example, if someone was using the app and had to close it, when the user opens it again youll want to check if its an existing user logged in or if its a new person, depending on that result send him either to the Login page or to the Home page.

For that to happen we need to first import both HomePage and LoginPage:

```
import { LoginPage } from '../pages/login/login';
import { HomePage } from '../pages/home/home';
```

Now well do a couple more things to set up our observer, well use it to tell it where to go depending on the users state.

NOTE: Theres a bug where the **`onAuthStateChanged()`** function isnt wrapped correctly in the right zone, Im reading more about it but you can fix it using NgZone.

First well need to import **`NgZone`**:

```
import { Component, NgZone } from '@angular/core';
```

Then create a zone variable right before the constructor:

```
zone: NgZone;
```

And then create the auth observer making sure to run **`NgZone`** inside the function:

```
firebase.auth().onAuthStateChanged((user) => {
  this.zone.run( () => {
    if (!user) {
      this.rootPage = LoginPage;
    } else {
      this.rootPage = HomePage;
    }
  });
});
```

You are creating an observer for the user object. By using an observer, you ensure that the Auth object isn't in an intermediate state (such as initialization) when you get the current user.

You can read more on that on Firebase docs.

2.3 The Auth Provider

Now it's time to create our auth service, remember we created a provider called AuthData that's going to handle Firebase «» Ionic interactions. Let's open that file.

Delete the methods there and leave it like this:

```
import {Injectable} from '@angular/core';

@Injectable()
export class AuthData {
  constructor() {}
}
```

Now let's start adding to it, first we'll need to import everything we are going to be using inside the file:

```
import { Injectable } from '@angular/core';

import firebase from 'firebase';

@Injectable()
export class AuthData {
  // Here we declare the variables we'll be using.
  public fireAuth: any;
  public userProfile: any;

  constructor() {

  }

}
```

And now we need to get firebase references:

```
constructor() {
  this.fireAuth = firebase.auth();
  this.userProfile = firebase.database().ref('/userProfile');
}
```

`this.fireAuth = firebase.auth();` creates a firebase auth reference, now you can get access to all the auth methods with `this.fireAuth`.

`this.userProfile = firebase.database().ref('/userProfile');` is creating a database reference to the `userProfile` node on my firebase database.

The first function we'll create is the login function:

```
loginUser(email: string, password: string): any {
  return this.fireAuth.signInWithEmailAndPassword(email, password);
}
```

Let's break down what's going on there:

- We are creating a function that's called `loginUser`, it takes an email and a password, both strings.
- That function is returning Firebase login.
- We are calling `this.fireAuth.signInWithEmailAndPassword()` passing the email & password.

That was an easy one, see that Firebase's `signInWithEmailAndPassword()` is taking care of all the login logic and all we are doing is handling our end on the app.

What if the user doesn't have an account to login to? Easy, let's create a signup function now:

```
signupUser(email: string, password: string): any {
  return this.fireAuth.createUserWithEmailAndPassword(email, password)
    .then((newUser) => {
      this.userProfile.child(newUser.uid).set({email: email});
    });
}
```

There are a few things going on here:

- We are creating a **`signupUser`** function that takes email & password both strings.
- We are passing those to **`.createUserWithEmailAndPassword()`** which handles the user creating logic for us.
- After the user is created Firebase also logs him in, and then we need to create a **`userProfile`** node for that user:

```
this.userProfile.child(newUser.uid).set({email: email});
```

Yeah Jorge, but what if my user does have an account but can't remember his password? Well dear reader then you create a reset password function:

```
resetPassword(email: string): any {
  return this.fireAuth.sendPasswordResetEmail(email);
}
```

- We are creating a **`resetPassword`** function that takes an email as a string.
- We are passing that email to:

`this.fireAuth.sendPasswordResetEmail(email)`

And Firebase will take care of the reset login, they send an email to your user with a password reset link, the user follows it and changes his password without you breaking a sweat.

And lastly we'll need to create a logout function:

```
logoutUser(): any {
  return this.fireAuth.signOut();
}
```

That one doesn't take any arguments it just checks for the current user and logs him out.

2.4 THE ACTUAL PAGES

So we created our service to handle all the Firebase «» Ionic stuff, now we need to create the actual pages our user is going to see, let's start with the login page.

2.4.1 Login

Open `pages/login/login.html` and create a simple login for to capture email and password:

```
<ion-header>
  <ion-navbar color="primary">
    <ion-title>
      Login
    </ion-title>
  </ion-navbar>
```

```
</ion-header>

<ion-content padding>
  <img src="http://placehold.it/300x100">

  <form [formGroup]="loginForm" (submit)="loginUser()" novalidate>

    <ion-item>
      <ion-label stacked>Email</ion-label>
      <ion-input #email formControlName="email" type="email"
      (change)="elementChanged(email)" placeholder="Your email address"
      [class.invalid]="!loginForm.controls.email.valid &&
      (emailChanged || submitAttempt)"></ion-input>
    </ion-item>

    <ion-item class="error-message" *ngIf="!loginForm.controls.email.valid &&
    (emailChanged || submitAttempt)">
      <p>Please enter a valid email.</p>
    </ion-item>

    <ion-item>
      <ion-label stacked>Password</ion-label>
      <ion-input #password formControlName="password"
      type="password" (change)="elementChanged(password)"
      placeholder="Your password"
      [class.invalid]="!loginForm.controls.password.valid &&
      (passwordChanged || submitAttempt)"></ion-input>
    </ion-item>

    <ion-item class="error-message"
    *ngIf="!loginForm.controls.password.valid  && (passwordChanged ||
      submitAttempt)">
      <p>Your password needs more than 6 characters.</p>
    </ion-item>

    <button ion-button block type="submit">
      Login
    </button>

  </form>

    <button ion-button block clear (click)="goToSignup()">
      Create a new account
    </button>

    <button ion-button block clear (click)="goToResetPassword()">
      I forgot my password
    </button>
</ion-content>
```

If you are new to Ionic 2 you might be wondering what **formGroup** is.

Well it's handling our value, it's part of Angular2 FormBuilder class, that I'm going to use to validate my form later.

> Quick note: I love **formGroup** on empty values it sends null instead of undefined so less work for me sending data to firebase

That code should be easy to follow, we are creating a login form, a couple of buttons to take us to the signup page or the reset password page, and we are adding an image just to give it something nice to see.

Now open your **login.scss** and give it some style:

```
page-login {
  form {
    margin-bottom: 32px;
    button {
      margin-top: 20px;
    }
  }

  p {
    font-size: 0.8em;
    color: #d2d2d2;
  }

  ion-label {
    margin-left: 5px;
  }

  ion-input {
    padding: 5px;
  }

  .invalid {
    border-bottom: 1px solid #FF6153;
  }

  .error-message .item-inner {
    border-bottom: 0 !important;
  }

}
```

Nothing to fancy, just a few margins to make everything look a bit better.

If you open your login.ts file now, you'll have something similar to this:

```
import { NavController } from 'ionic-angular';
import { Component } from '@angular/core';

@Component({
  selector: 'page-login',
  templateUrl: 'login.html',
})
export class LoginPage {

  constructor(public nav: NavController) {

  }

}
```

First thing we'll add are the imports, so everything is available when you need it:

```
import {
  NavController,
  LoadingController,
  AlertController } from 'ionic-angular';
import { Component } from '@angular/core';
import { FormBuilder, Validators } from '@angular/forms';
import { AuthData } from '../../providers/auth-data';
import { SignupPage } from '../signup/signup';
import { HomePage } from '../home/home';
import { ResetPasswordPage } from '../reset-password/reset-password';
```

See we are importing **FormBuilder** from angular, **AuthData** to handle our functions and the pages so we can push our user there.

Now you need to tell your **LoginPage** that it's going to use **AuthData** as a provider, as easy as:

Create the **loginForm** and inject both **FormBuilder** and **AuthData** in the constructor:

```
export class LoginPage {
  public loginForm;

  constructor(public nav: NavController, public authData: AuthData,
    public formBuilder: FormBuilder,public alertCtrl: AlertController,
    public loadingCtrl: LoadingController) {

      this.loginForm = formBuilder.group({
        email: ['', Validators.compose([Validators.required])],
        password: ['', Validators.compose([Validators.minLength(6),
        Validators.required])]
      });

    }

}
```

See we are using **`formBuilder`** inside the constructor to initialize the fields and give them a required validator.

If you want to know more about **`FormBuilder`** check Angular's docs.

Now let's create our functions:

```
loginUser(){
  this.submitAttempt = true;

  if (!this.loginForm.valid){
    console.log(this.loginForm.value);
  } else {
    this.authData.loginUser(this.loginForm.value.email,
      this.loginForm.value.password).then( authData => {
    this.nav.setRoot(HomePage);
  }, error => {
    this.loading.dismiss().then( () => {
      let alert = this.alertCtrl.create({
        message: error.message,
        buttons: [
          {
            text: "Ok",
            role: 'cancel'
          }
        ]
      });
      alert.present();
    });
```

```
  });

  this.loading = this.loadingCtrl.create({
    dismissOnPageChange: true,
  });
  this.loading.present();
  }
}
```

Our login function just takes the value of the form fields and pass them to our **loginUser** function inside our **AuthData** service.

It's also calling Ionic2's loading component, since the app needs to communicate with the server to log the user in there might be a small delay to send the user to the HomePage so we are using a loading component to give a visual so the user understand that's loading :P

```
goToSignup(){
  this.nav.push(SignupPage);
}
```

```
goToResetPassword(){
  this.nav.push(ResetPasswordPage);
}
```

Those 2 should be really easy to understand, we are sending the user to the **SignupPage** or the **ResetPasswordPage**

Your file should look like this:

```
import {
  NavController,
  LoadingController,
  AlertController } from 'ionic-angular';
import { Component } from '@angular/core';
import { FormBuilder, Validators } from '@angular/forms';
import { AuthData } from '../../providers/auth-data';
import { SignupPage } from '../signup/signup';
```

```
import { HomePage } from '../home/home';
import { ResetPasswordPage } from '../reset-password/reset-password';

@Component({
  selector: 'page-login',
  templateUrl: 'login.html',
})
export class LoginPage {
  public loginForm;
  emailChanged: boolean = false;
  passwordChanged: boolean = false;
  submitAttempt: boolean = false;
  loading: any;

  constructor(public nav: NavController, public authData: AuthData,
    public formBuilder: FormBuilder, public alertCtrl: AlertController,
    public loadingCtrl: LoadingController) {

    /**
     * Creates a ControlGroup that declares the fields available,
     * their values and the validators that they are going
     * to be using.
     *
     * I set the password's min length to 6 characters because
     * that's Firebase's default, feel free to change that.
     */
    this.loginForm = formBuilder.group({
      email: ['', Validators.compose([Validators.required])],
      password: ['', Validators.compose([Validators.minLength(6),
      Validators.required])]
    });
  }

  /**
   * Receives an input field and sets the corresponding fieldChanged
   * property to 'true' to help with the styles.
   */
  elementChanged(input){
    let field = input.inputControl.name;
    this[field + "Changed"] = true;
  }

  /**
   * If the form is valid it will call the AuthData service to log the user
   * in displaying a loading component while the user waits.
   *
   * If the form is invalid it will just log the form value,
   * feel free to handle that as you like.
   */
  loginUser(){
```

```
    this.submitAttempt = true;

    if (!this.loginForm.valid){
      console.log(this.loginForm.value);
    } else {
      this.authData.loginUser(this.loginForm.value.email,
        this.loginForm.value.password).then( authData => {
        this.nav.setRoot(HomePage);
      }, error => {
        this.loading.dismiss().then( () => {
          let alert = this.alertCtrl.create({
            message: error.message,
            buttons: [
              {
                text: "Ok",
                role: 'cancel'
              }
            ]
          });
          alert.present();
        });
      });

      this.loading = this.loadingCtrl.create({
        dismissOnPageChange: true,
      });
      this.loading.present();
    }
  }

  goToSignup(){
    this.nav.push(SignupPage);
  }

  goToResetPassword(){
    this.nav.push(ResetPasswordPage);
  }

}
```

2.4.2 Reset Password Page

Now let's help our user reset his password, go to:

```
pages/reset-password/reset-password.html
```

Once you are there create a simple form that takes an email address:

```
<ion-header>
  <ion-navbar color="primary">
    <ion-title>
      Reset your Password
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <img src="http://placehold.it/300x100">
  <form [formGroup]="resetPasswordForm" (submit)="resetPassword()" novalidate>

    <ion-item>
      <ion-label stacked>Email</ion-label>
      <ion-input #email formControlName="email" type="email"
      (change)="elementChanged(email)" placeholder="Your email address"
      [class.invalid]="!resetPasswordForm.controls.email.valid &&
      (emailChanged || submitAttempt)"></ion-input>
    </ion-item>

    <ion-item class="error-message"
    *ngIf="!resetPasswordForm.controls.email.valid  &&
    (emailChanged || submitAttempt)">
      <p>Please enter a valid email.</p>
    </ion-item>

    <button ion-button block type="submit">
      Reset your Password
    </button>
  </form>
</ion-content>
```

Nothing crazy here, just an email input here, and open the ‘ reset-password.scss‘ to add a few margins:

```
page-reset-password {
  form {
    margin-bottom: 32px;
```

```
    button {
      margin-top: 20px;
    }
  }

  p {
    font-size: 0.8em;
    color: #d2d2d2;
  }

  ion-label {
    margin-left: 5px;
  }

  ion-input {
    padding: 5px;
  }

  .invalid {
    border-bottom: 1px solid #FF6153;
  }

  .error-message .item-inner {
    border-bottom: 0 !important;
  }

}
```

Now go to **reset-password.ts** and open the file:

```
import { NavController } from 'ionic-angular';
import { Component } from '@angular/core';

@Component({
  selector: 'page-reset-password',
  templateUrl: 'reset-password.html',
})
export class ResetPasswordPage {

  constructor(public nav: NavController) {

  }

}
```

Just like before, we are going to be adding the imports and injecting our ser-

vices into the constructor:

```
import {
  NavController,
  LoadingController,
  AlertController } from 'ionic-angular';
import { Component } from '@angular/core';
import { FormBuilder, Validators } from '@angular/forms';
import { AuthData } from '../../providers/auth-data';

@Component({
  selector: 'page-reset-password',
  templateUrl: 'reset-password.html',
})
export class ResetPasswordPage {
  public resetPasswordForm;


  constructor(public authData: AuthData, public formBuilder: FormBuilder,
    public nav: NavController, public loadingCtrl: LoadingController,
    public alertCtrl: AlertController) {

    this.resetPasswordForm = formBuilder.group({
      email: ['', Validators.compose([Validators.required])],
    })
  }

}
```

Then you'll just need to create the actual reset password function:

```
resetPassword(){
  this.submitAttempt = true;

  if (!this.resetPasswordForm.valid){
    console.log(this.resetPasswordForm.value);
  } else {
    this.authData.resetPassword(this.resetPasswordForm.value.email)
    .then((user) => {
      let alert = this.alertCtrl.create({
        message: "We just sent you a reset link to your email",
        buttons: [
          {
            text: "Ok",
            role: 'cancel',
            handler: () => {
              this.nav.pop();
```

```
            }
          }
        ]
      });
      alert.present();

    }, (error) => {
      var errorMessage: string = error.message;
      let errorAlert = this.alertCtrl.create({
        message: errorMessage,
        buttons: [
          {
            text: "Ok",
            role: 'cancel'
          }
        ]
      });

      errorAlert.present();
    });
  }
}
```

Same as login, it takes the value of the form field, sends it to the **AuthData** service and displays a loading component while we get Firebase's response.

It should look like this:

```
import {
  NavController,
  LoadingController,
  AlertController } from 'ionic-angular';
import { Component } from '@angular/core';
import { FormBuilder, Validators } from '@angular/forms';
import { AuthData } from '../../providers/auth-data';

@Component({
  selector: 'page-reset-password',
  templateUrl: 'reset-password.html',
})
export class ResetPasswordPage {
  public resetPasswordForm;
  emailChanged: boolean = false;
  passwordChanged: boolean = false;
  submitAttempt: boolean = false;
```

```
constructor(public authData: AuthData, public formBuilder: FormBuilder,
  public nav: NavController, public loadingCtrl: LoadingController,
  public alertCtrl: AlertController) {

  this.resetPasswordForm = formBuilder.group({
    email: ['', Validators.compose([Validators.required])],
  })
}

/**
 * Receives an input field and sets the corresponding fieldChanged
 * property to 'true' to help with the styles.
 */
elementChanged(input){
  let field = input.inputControl.name;
  this[field + "Changed"] = true;
}

/**
 * If the form is valid it will call the AuthData service to reset
 * the user's password displaying a loading
 * component while the user waits.
 *
 * If the form is invalid it will just log the form value,
 * feel free to handle that as you like.
 */
resetPassword(){

  this.submitAttempt = true;

  if (!this.resetPasswordForm.valid){
    console.log(this.resetPasswordForm.value);
  } else {
    this.authData.resetPassword(this.resetPasswordForm.value.email)
    .then((user) => {
      let alert = this.alertCtrl.create({
        message: "We just sent you a reset link to your email",
        buttons: [
          {
            text: "Ok",
            role: 'cancel',
            handler: () => {
              this.nav.pop();
            }
          }
        ]
      });
      alert.present();

    }, (error) => {
      var errorMessage: string = error.message;
```

```
        let errorAlert = this.alertCtrl.create({
          message: errorMessage,
          buttons: [
            {
              text: "Ok",
              role: 'cancel'
            }
          ]
        });

        errorAlert.present();
      });
    }

  }


}
```

2.4.3 Signup

It's time now to let out users create accounts on our app, we'll be working on **`pages/signup/signup.html`** create a basic form asking for email and password (You can ask for more stuff here, like the user's name for example)

```
<ion-header>
  <ion-navbar color="primary">
    <ion-title>
      Create an Account
    </ion-title>
  </ion-navbar>
</ion-header>


<ion-content padding>
  <img src="http://placehold.it/300x100">
  <form [formGroup]="signupForm" (submit)="signupUser()" novalidate>

    <ion-item>
      <ion-label stacked>Email</ion-label>
      <ion-input #email formControlName="email" type="email"
      (change)="elementChanged(email)" placeholder="Your email address"
      [class.invalid]="!signupForm.controls.email.valid &&
      (emailChanged || submitAttempt)"></ion-input>
```

```
    </ion-item>

    <ion-item class="error-message"
    *ngIf="!signupForm.controls.email.valid
    && (emailChanged || submitAttempt)">
      <p>Please enter a valid email.</p>
    </ion-item>

    <ion-item>
      <ion-label stacked>Password</ion-label>
      <ion-input #password formControlName="password" type="password"
      (change)="elementChanged(password)" placeholder="Your password"
      [class.invalid]="!signupForm.controls.password.valid &&
      (passwordChanged || submitAttempt)"></ion-input>
    </ion-item>
    <ion-item class="error-message"
    *ngIf="!signupForm.controls.password.valid
    && (passwordChanged || submitAttempt)">
      <p>Your password needs more than 6 characters.</p>
    </ion-item>

    <button ion-button block type="submit">
      Create an Account
    </button>

  </form>

</ion-content>
```

Nothing new here, same things we used in the login form.

Now add some margins on the **signup.scss** file:

```
page-signup {
  form {
    margin-bottom: 32px;
    button {
      margin-top: 20px;
    }
  }

  p {
    font-size: 0.8em;
    color: #d2d2d2;
  }

  ion-label {
```

```
    margin-left: 5px;
  }

  ion-input {
    padding: 5px;
  }

  .invalid {
    border-bottom: 1px solid #FF6153;
  }

  .error-message .item-inner {
    border-bottom: 0 !important;
  }

}
```

And finally open your **signup.ts** file, import the services you'll need and inject them to your constructor:

```
import {
  NavController,
  LoadingController,
  AlertController } from 'ionic-angular';
import { Component } from '@angular/core';
import { FormBuilder, Validators } from '@angular/forms';
import { AuthData } from '../../providers/auth-data';
import { HomePage } from '../home/home';

@Component({
  selector: 'page-signup',
  templateUrl: 'signup.html'
})
export class SignupPage {
  public signupForm;
  emailChanged: boolean = false;
  passwordChanged: boolean = false;
  submitAttempt: boolean = false;
  loading: any;


  constructor(public nav: NavController, public authData: AuthData,
    public formBuilder: FormBuilder, public loadingCtrl: LoadingController,
    public alertCtrl: AlertController) {

    this.signupForm = formBuilder.group({
      email: ['', Validators.compose([Validators.required])],
```

```
      password: ['', Validators.compose([Validators.minLength(6),
      Validators.required])]
    })
  }

}
```

You've done this twice now so there shouldn't be anything new here, now create the signup function:

```
signupUser(){
  this.submitAttempt = true;

  if (!this.signupForm.valid){
    console.log(this.signupForm.value);
  } else {
    this.authData.signupUser(this.signupForm.value.email,
      this.signupForm.value.password).then(() => {
      this.nav.setRoot(HomePage);
    }, (error) => {
      this.loading.dismiss();
      let alert = this.alertCtrl.create({
        message: error.message,
        buttons: [
          {
            text: "Ok",
            role: 'cancel'
          }
        ]
      });
      alert.present();
    });

    this.loading = this.loadingCtrl.create({
      dismissOnPageChange: true,
    });
    this.loading.present();
  }
}
```

Your **`signup.ts`** should look like this:

```
import {
  NavController,
  LoadingController,
  AlertController } from 'ionic-angular';
import { Component } from '@angular/core';
import { FormBuilder, Validators } from '@angular/forms';
import { AuthData } from '../../providers/auth-data';
import { HomePage } from '../home/home';

@Component({
  selector: 'page-signup',
  templateUrl: 'signup.html'
})
export class SignupPage {
  public signupForm;
  emailChanged: boolean = false;
  passwordChanged: boolean = false;
  submitAttempt: boolean = false;
  loading: any;


  constructor(public nav: NavController, public authData: AuthData,
    public formBuilder: FormBuilder, public loadingCtrl: LoadingController,
    public alertCtrl: AlertController) {

    this.signupForm = formBuilder.group({
      email: ['', Validators.compose([Validators.required])],
      password: ['', Validators.compose([Validators.minLength(6),
      Validators.required])]
    })
  }

  /**
   * Receives an input field and sets the corresponding fieldChanged
   * property to 'true' to help with the styles.
   */
  elementChanged(input){
    let field = input.inputControl.name;
    this[field + "Changed"] = true;
  }

  /**
   * If the form is valid it will call the AuthData service to sign
   * the user up password displaying a loading
   *  component while the user waits.
   *
   * If the form is invalid it will just log the form value,
   * feel free to handle that as you like.
   */
  signupUser(){
    this.submitAttempt = true;
```

```
    if (!this.signupForm.valid){
      console.log(this.signupForm.value);
    } else {
      this.authData.signupUser(this.signupForm.value.email,
        this.signupForm.value.password).then(() => {
        this.nav.setRoot(HomePage);
      }, (error) => {
        this.loading.dismiss();
        let alert = this.alertCtrl.create({
          message: error.message,
          buttons: [
            {
              text: "Ok",
              role: 'cancel'
            }
          ]
        });
        alert.present();
      });

      this.loading = this.loadingCtrl.create({
        dismissOnPageChange: true,
      });
      this.loading.present();
    }
  }
}
```

And that's it, now users can create accounts on your app :-)

2.4.4 Test the LogOut.

If you want to test the logOut function just go to your HomePage and add a log out button, I like to have it on my nav-bar:

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Home Page
    </ion-title>
    <ion-buttons end>
      <button ion-button icon-only (click)="logOut()">
```

```
        <ion-icon name="log-out"></ion-icon>
        </button>
    </ion-buttons>
  </ion-navbar>
</ion-header>
```

And in **home.ts** add the logout function:

```
logOut(){
  this.authData.logoutUser().then(() => {
    this.nav.setRoot(LoginPage);
  });
}
```

The logout function logs the user out and sends him to the LoginPage

Remember you'll need to import the **LoginPage** and the **AuthData** service and inject them in your class.

And there you have it, you have a fully functional auth system working on your app now.

If you run into trouble don't hesitate to ask for help, just send me an email, I'm happy to help other developers get started using my favorite combo Ionic + Firebase :-)

Chapter 3

CRUD Firebase Objects

3.1 Introduction

We just learned about authentication for our new app, a simple app for event managers to keep track of their guests.

Now we need to build a user profile, mostly picked the user profile because I want to show you how to create, update and read objects from Firebase since that seems to be a major pain for several people.

I think that's enough for an intro, so let's jump into business!

3.2 Setup & Cleanup

The first thing we are going to do is to setup everything we'll need for this part of the tutorial, we'll be creating a profile page and a profile data provider.

Remember that the actual files are already created, now we just need to start building on top of them.

The second thing we'll need to do is removing the logOut function from the HomePage, it just feels weird having it there, we are going to move it to the ProfilePage later.

Just go into **src/pages/home/home.html** and find the button inside the **<ion-navbar>** go ahead and replace it with this:

```
<button (click)="goToProfile()">
  <ion-icon name="person"></ion-icon>
</button>
```

Now you'll need to go to **src/pages/home/home.ts** and replace the logOut function with a simple function that sends you to the profile page (remember to change your imports) by the end it should look something like this:

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';
import { ProfilePage } from '../profile/profile';

@Component({
  selector: 'page-home',
  templateUrl: 'home.html',
})
export class HomePage {
  constructor(public nav: NavController) {
    this.nav = nav;
  }

  goToProfile(){
    this.nav.push(ProfilePage);
  }
}
```

See that we removed the AuthData import and changed the logOut function for a simple **goToProfile()** function.

That wasn't so hard, was it?

3.3 The User Profile Provider

Now into a bit more complicated bits, we are going to create our **`ProfileData`** provider, the idea for this provider is that it lets us store our profile information in Firebase's real time database and also change our email and password from our profile data.

This provider should have a function for:

- Getting the user profile
- Updating the user's name
- Updating the user's data of birth (I always save this stuff so I can surprise my users later :P)
- Updating the user's email both in the real time database and the auth data
- Changing the user's password.

The file isn't that big, so I'm going to paste the entire provider here and explain the important parts inside the code with comments.

```
/**
* This should come as no surprise, we need to import Injectable
* so we can use this provider as an injectable.
* We also need to import firebase so we can talk to our DB.
*/
import { Injectable } from '@angular/core';
import firebase from 'firebase';


@Injectable()
export class ProfileData {
  // We'll use this to create a database reference to the userProfile node.
  userProfile: any;
```

```
// We'll use this to create an auth reference to the logged in user.
currentUser: any;


constructor() {
  /**
  * Here we create the references I told you about 2 seconds ago :P
  */
  this.currentUser = firebase.auth().currentUser;
  this.userProfile = firebase.database().ref('/userProfile');

}

/**
* This one should be really easy to follow, we are calling a
* function getUserProfile() that takes no parameters.
* This function returns a DATABASE reference to the userProfile/uid
* of the current user and we'll use it to get the user profile
* info in our page.
*/
getUserProfile(): any {
  return this.userProfile.child(this.currentUser.uid);
}

/**
* This one takes 2 string parameters, firstName & lastName,
* it just saves those 2 to the userProfile/uid node
* for the current user as the firstName & lastName properties.
*/
updateName(firstName: string, lastName: string): any {
  return this.userProfile.child(this.currentUser.uid).update({
    firstName: firstName,
    lastName: lastName,
  });
}

/**
* Pretty much the same as before, just that instead of saving the
* name it's saving the date of birth
*/
updateDOB(birthDate: string): any {
  return this.userProfile.child(this.currentUser.uid).update({
    birthDate: birthDate,
  });
}

/**
* This is were things get trickier, this one is taking the user's
* email and first it's calling the this.currentUser auth reference
* to call it's updateEmail() function, it's very important that you
* understand that this is changing your email in the AUTH portion
```

```
  * of firebase, the one stored in the userProfile/uid node hasn't changed.
  * After it successfully changes your email in the AUTH portion of
  * firebase it updates your email in the real time database in
  * the userProfile/uid node.
  */
  updateEmail(newEmail: string): any {
    this.currentUser.updateEmail(newEmail).then(() => {
      this.userProfile.child(this.currentUser.uid).update({
        email: newEmail
      });
    }, (error) => {
      console.log(error);
    });
  }

  /**
  * Just like before this is changing the user's password, but remember,
  * this has nothing to do with the database this is the AUTH portion of
  * Firebase.
  */
  updatePassword(newPassword: string): any {
    this.currentUser.updatePassword(newPassword).then(() => {
      console.log("Password Changed");
    }, (error) => {
      console.log(error);
    });
  }
}
```

You should probably go get yourself a cookie, that was a lot of code and your sugar levels need a refill :P

There you have a fully functional **`ProfileData`** provider, now we just need to build our **`ProfilePage`** and connect it to our provider.

3.4 The Profile Page

The first thing I'm going to build for the profile page is the HTML & CSS I want something I can look at while I connect my functions.

The logic behind my html is simple, I want a single ProfilePage, I want users

to be able to update their profile there without going to a different page with a form, so I'm using Ionic's excellent grid system and I'm using a lot of alerts to capture the data in page.

Check it out:

```
<ion-header>
  <ion-navbar color="primary">
    <ion-title>Profile</ion-title>
    <ion-buttons end>
      <button ion-button icon-only (click)="logOut()">
        <ion-icon name="log-out"></ion-icon>
      </button>
    </ion-buttons>
  </ion-navbar>
</ion-header>

<ion-content class="profile">
  <ion-list>
    <ion-list-header>
      Personal Information
    </ion-list-header>

    <ion-item (click)="updateName()">
      <ion-grid>
        <ion-row>
          <ion-col width-50>
            Name
          </ion-col>
          <ion-col *ngIf="userProfile?.firstName || userProfile?.lastName">
            {{userProfile?.firstName}} {{userProfile?.lastName}}
          </ion-col>
          <ion-col class="placeholder-profile" *ngIf="!userProfile?.firstName">
            <span>
              Tap here to edit.
            </span>
          </ion-col>
        </ion-row>
      </ion-grid>
    </ion-item>

    <ion-item>
      <ion-label class="dob-label">Date of Birth</ion-label>
      <ion-datetime displayFormat="MMM D, YYYY" pickerFormat="D MMM YYYY"
        [(ngModel)]="birthDate" (ionChange)="updateDOB(birthDate)">
      </ion-datetime>
    </ion-item>

    <ion-item (click)="updateEmail()">
```

```
      <ion-grid>
        <ion-row>
          <ion-col width-50>
            Email
          </ion-col>
          <ion-col width-50 *ngIf="userProfile?.email">
            {{userProfile?.email}}
          </ion-col>
          <ion-col class="placeholder-profile" *ngIf="!userProfile?.email">
            <span>
              Tap here to edit.
            </span>
          </ion-col>
        </ion-row>
      </ion-grid>
    </ion-item>

    <ion-item (click)="updatePassword()">
      <ion-grid>
        <ion-row>
          <ion-col width-50>
            Password
          </ion-col>
          <ion-col class="placeholder-profile">
            <span>
              Tap here to edit.
            </span>
          </ion-col>
        </ion-row>
      </ion-grid>
    </ion-item>
  </ion-list>
</ion-content>
```

It should be very easy to understand, the labels take the left half of the grid and the values take the right half.

If there's no value we'll show a placeholder that says: "Tap here to edit" this will let our user know that they need to tap there to be able to edit the profile items.

The one that's different is the birth date part:

```
<ion-item>
  <ion-label class="dob-label">Date of Birth</ion-label>
  <ion-datetime displayFormat="MMM D, YYYY" pickerFormat="D MMM YYYY"
    [(ngModel)]="birthDate" (ionChange)="updateDOB(birthDate)">
  </ion-datetime>
</ion-item>
```

I could have opened a modal to update the DOB but I actually went with the **(ionChange)** function, this handy function let's me run my **updateDOB()** function every time the date changes, and I don't have to use a form and submit it :)

Now that we have the HTML I'm going to show you some really basic CSS I used to make it look decent (remember, I come from a back-end background, CSS isn't 'my thing')

```
page-profile {
  ion-item ion-label {
    margin: 0 !important;
  }

  ion-list {
    margin: 0;
    padding: 0;
  }

  ion-list-header {
    background-color: #ECECEC;
    margin: 0;
    padding: 10px 16px;
  }

  ion-item {
    padding: 0;
  }

  ion-datetime {
    padding-left: 3px !important;
  }

  .item-inner {
    border: none !important;
    padding: 0;
  }
```

```
  .placeholder-profile {
    color: #CCCCCC;
  }

  .dob-label {
    color: #000000 !important;
    padding: 10px !important;
    max-width: 50% !important;
  }
}
```

Nothing to weird, just some margins and colors.

That's ready so let's move to our profile.ts file, let's import what we'll need and set up the constructor first:

```
import { NavController, Alert } from 'ionic-angular';
import { Component } from '@angular/core';
import { ProfileData } from '../../providers/profile-data/profile-data';
import { AuthData } from '../../providers/auth-data/auth-data';
import { LoginPage } from '../login/login';

@Component({
  selector: 'page-profile',
  templateUrl: 'profile.html',
})
export class ProfilePage {
  public userProfile: any;
  public birthDate: string;

  constructor(public nav: NavController, public profileData: ProfileData,
    public authData: AuthData) {
    this.nav = nav;
    this.profileData = profileData;

    this.profileData.getUserProfile().on('value', (data) => {
      this.userProfile = data.val();
      this.birthDate = this.userProfile.birthDate;
    });

  }

}
```

Let's look in detail at what we did there:

- We are importing everything we are going to need.
- We are declaring a userProfile variable to hold our profile and a **`birthDate`** variable to hold our user's birth date.
- We are calling **`getUserProfile()`** from the **`ProfileData`** and assigning it's **`val()`** to our userProfile variable, that way we can user it in our page.

So it's time to add the functions now.

First we'll get rid of the easy one, the one we removed from the HomePage, the logout function:

```
logOut(){
  this.authData.logoutUser().then(() => {
    this.nav.rootNav.setRoot(LoginPage);
  });
}
```

This is exactly what you expected it to be (since you saw it in the previous chapter) it calls the **`logoutUser`** function and then it sets the **`LoginPage`** as our rootPage so the user is taken to login without the ability to have a back button.

Now let's move to updating our user's name:

```
updateName(){
  let alert = this.alertCtrl.create({
    message: "Your first name & last name",
    inputs: [
      {
        name: 'firstName',
        placeholder: 'Your first name',
        value: this.userProfile.firstName
      },
      {
        name: 'lastName',
        placeholder: 'Your last name',
```

```
        value: this.userProfile.lastName
      },
    ],
    buttons: [
      {
        text: 'Cancel',
      },
      {
        text: 'Save',
        handler: data => {
          this.profileData.updateName(data.firstName, data.lastName);
        }
      }
    ]
  });
  alert.present();
}
```

We are creating a prompt here to ask users for their first and last names. Once we get them our "Save" button is going to call a handler, that's just going to take those first and last names and sends them to the **`updateName`** function of our **`ProfileData`** provider.

The Birth Date is even easier, since we created an **`(ionChange)`** inside the **`<ion-datetime>`** we just need to call that function and use it to pass the birth date to our **`ProfileData`** provider:

```
updateDOB(birthDate){
  this.profileData.updateDOB(birthDate);
}
```

Now email and password are going to be the same as the updateName function, just keep in mind that we are changing the input types to email & password to get the browser validation for them:

```
updateEmail(){
  let alert = this.alertCtrl.create({
    inputs: [
      {
```

```
        name: 'newEmail',
        placeholder: 'Your new email',
      },
    ],
    buttons: [
      {
        text: 'Cancel',
      },
      {
        text: 'Save',
        handler: data => {
          this.profileData.updateEmail(data.newEmail);
        }
      }
    ]
  });
  alert.present();
}

updatePassword(){
  let alert = this.alertCtrl.create({
    inputs: [
      {
        name: 'newPassword',
        placeholder: 'Your new password',
        type: 'password'
      },
    ],
    buttons: [
      {
        text: 'Cancel',
      },
      {
        text: 'Save',
        handler: data => {
          this.profileData.updatePassword(data.newPassword);
        }
      }
    ]
  });
  alert.present();
}
```

That will create both functions and send the respective email & passwords to the ProfileData provider.

And that's it, now you have a better understanding on how to work with objects in Firebase, if you run into any trouble just email me and I'll do my best to help

guide you in the right direction :)

Chapter 4

Working with Lists

4.1 The Event Provider

Now that we are ready to go, let's go into **event-data.ts** and create the functions that are going to communicate with Firebase.

First thing we'll do is import and declare everything we need to use.

```
import { Injectable } from '@angular/core';
import firebase from 'firebase';

@Injectable()
export class EventData {
  public currentUser: any;
  public eventList: any;

  constructor() {
    this.currentUser = firebase.auth().currentUser.uid;
    this.eventList = firebase.database().ref('userProfile/' +
    this.currentUser + '/eventList');

  }

}
```

We can see a couple of things here:

- We are declaring a **currentUser** variable where we get the uid of the logged in user, this is because we're going to store the eventList node inside the **userProfile** node in Firebase, to make it easier on the security rules.
- We are declaring an **eventList** variable, this to get a database reference for the **userProfile/uid/eventList** node and be able to write and read directly from it easier.

Now we need a function to create our events:

```
createEvent(eventName: string, eventDate: string,
  eventPrice: number, eventCost: number): any {
  return this.eventList.push({
    name: eventName,
    date: eventDate,
    price: eventPrice,
    cost: eventCost
  }).then( newEvent => {
    this.eventList.child(newEvent.key).child('id').set(newEvent.key);
  });
}
```

A couple of things to note:

- We are using **.push()** on the **eventList** node because we want firebase to append every new object to this list, and to auto generate a random ID so we know there aren't going to be 2 objects with the same ID.
- We are adding the name, date, ticket price and cost of the event (Mostly because in the next chapter I'm going to use them for transactions + real time updates on revenue per event.)

- I'm adding the new event ID to itself as a new property called id.

NOTE: That might seem weird or a *"bad practice"* for some people, but I actually go with what makes my work easier, sometimes when working with nested data like list inside objects inside lists it's easier to have that id there for reading and filtering my data without needing to call Firebase. This isn't something you NEED to do, it's just something I like doing.

After we have the functions that's going to create our events we'll jut need one more to list them:

```
getEventList(): any {
  return this.eventList;
}
```

I think that one is pretty easy :P it's just returning the firebase reference we created earlier.

And one for receiving an event's ID and returning that event:

```
getEventDetail(eventId): any {
  return this.eventList.child(eventId);
}
```

4.2 The Home Page

Now since I haven't given much thought to the app's UI I'm going to create 2 buttons on the **HomePage** to take me to the **EventCreatePage** or the **EventListPage**.

Go to home.html and add the buttons inside the **<ion-content>** tag:

```
<button ion-button block color="primary" (click)="goToCreate()">
  Create a new Event
</button>

<button ion-button block color="primary" (click)="goToList()">
  See your events
</button>
```

Now go into **home.ts** and create those functions, first you'll need to import both pages:

```
import { EventCreatePage } from '../event-create/event-create';
import { EventListPage } from '../event-list/event-list';
```

And then you create the functions:

```
goToCreate(){
  this.nav.push(EventCreatePage);
}

goToList(){
  this.nav.push(EventListPage);
}
```

Easy right?

Now that we have that it's time to create the event part of the app, we are going to start with adding a new event, for that go to **event-create/event-create.html** and create a few inputs to save the event's name, date, ticket price and costs:

```
<ion-header>
  <ion-navbar>
    <ion-title>New Event</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-item>
    <ion-label stacked>Event Name</ion-label>
    <ion-input [(ngModel)]="eventName" type="text"
    placeholder="What's your event's name?"></ion-input>
  </ion-item>

  <ion-item>
    <ion-label stacked>Price</ion-label>
    <ion-input [(ngModel)]="eventPrice" type="number"
    placeholder="How much will guests pay?"></ion-input>
  </ion-item>

  <ion-item>
    <ion-label stacked>Cost</ion-label>
    <ion-input [(ngModel)]="eventCost" type="number"
    placeholder="How much are you spending?"></ion-input>
  </ion-item>

  <ion-item>
    <ion-label>Event Date</ion-label>
    <ion-datetime displayFormat="D MMM, YY" pickerFormat="DD MMM YYYY"
    [(ngModel)]="eventDate"></ion-datetime>
  </ion-item>

  <button ion-button block
  (click)="createEvent(eventName, eventDate, eventPrice, eventCost)">
    Create Event
  </button>
</ion-content>
```

Nothing we haven't seen in previous examples, we are just using a few inputs to get the data we need, and then creating a **`createEvent()`** function and passing it those values so we can use them later.

After you finish doing this, go to **`event-create/event-create.ts`** first we'll need to import our **`EventData`**

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';
import { EventData } from '../../providers/event-data';

@Component({
  selector: 'page-event-create',
  templateUrl: 'event-create.html',
})
export class EventCreatePage {

  constructor(public nav: NavController, public eventData: EventData) {
    this.nav = nav;
    this.eventData = eventData;
  }

}
```

After doing that we'll create our **`createEvent`** function, it will just send the data to the **`createEvent()`** function we already declared in our EventData provider.

```
createEvent(eventName: string, eventDate: string,
  eventPrice: number, eventCost: number) {
  this.eventData.createEvent(eventName, eventDate, eventPrice, eventCost).then( () => {
    this.nav.pop();
  });
}
```

Nothing to crazy, we are just sending the data to our **`EventData`** provider, and as soon as the event is successfully created we are using **`this.nav.pop()`** to go back a page to the **`HomePage`**.

Now that we can actually create events, we need a way to see our events, so let's go into the **`event-list`** folder and inside the **`event-list.html`** file and create a list of your events:

```
<ion-header>
  <ion-navbar>
    <ion-title>EventList</ion-title>
  </ion-navbar>
```

```
</ion-header>


<ion-content padding>
  <ion-list>
    <ion-item *ngFor="let event of eventList"
    (click)="goToEventDetail(event.id)">
      <h2>{{event?.name}}</h2>
      <p>Ticket: <strong>${{event?.price}}</strong></p>
      <p>Date: <strong>{{event?.date}}</strong></p>
    </ion-item>
  </ion-list>
</ion-content>
```

- We are just creating an item that will repeat itself for every event we have in our database.
- We are showing basic event data like the name, the ticket price for guests and the event date.
- When users tap on the event, they are going to be taken to the event's detail page.
- We are sending the entire event in the **`goToEventDetail()`** function so we don't have to pull it from Firebase again.

Now we need the logic to implement all of that so go into **`event-detail.ts`** and first import and declare everything you'll need:

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';
import { EventDetailPage } from '../event-detail/event-detail';
import { EventData } from '../../providers/event-data';

@Component({
  selector: 'page-event-list',
  templateUrl: 'event-list.html',
})
export class EventListPage {
  public eventList: any;
```

```
  constructor(public nav: NavController, public eventData: EventData) {
    this.nav = nav;
    this.eventData = eventData;

  }

}
```

- We are importing both the **`EventDetailPage`** & the **`EventData`** provider
- We are declaring a variable called **`eventList`** to hold our list of events.

Now we need to get that list of events from Firebase, so INSIDE your constructor call the **`getEventList()`** method from your EventData provider:

```
this.eventData.getEventList().on('value', snapshot => {
  let rawList = [];
  snapshot.forEach( snap => {
    rawList.push({
      id: snap.key,
      name: snap.val().name,
      price: snap.val().price,
      date: snap.val().date
    });
  });
  this.eventList = rawList;
});
```

We've done this before, we are just:

- Calling the **`getEventList()`** method from our provider.
- Going trough that list and adding it to an array.
- Setting our **`eventList`** variable to be equal to that array.

Now the first part of the html will work, it's going to show users a list of their events in the app.

But we're not done yet, we need to create the function to send users to the event detail, we'll do that right here:

```
goToEventDetail(eventId){
  this.nav.push(EventDetailPage, {
    eventId: eventId,
  });
}
```

We are receiving the entire event the user taps on, and when we push the user to that page we are also sending that event's ID as the **`eventId`** object.

Now all you need to do is go to **`event-detail.ts`** and receive that ID, for that you'll need to first import and declare **`NavParams`**:

```
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';

@Component({
  selector: 'page-event-detail',
  templateUrl: 'event-detail.html',
})
export class EventDetailPage {
  currentEvent: any;
  constructor(public nav: NavController, public navParams: NavParams) {
    this.navParams = navParams;
  }
}
```

Now, we're going to (inside the constructor) read the current event from the Firebase database:

```
this.eventData.getEventDetail(this.navParams.get('eventId'))
  .on('value', (snapshot) => {
    this.currentEvent = snapshot.val();
});
```

Now we can go into the **event-detail.html** and show all that data to our users:

```
<ion-header>
  <ion-navbar>
    <ion-title></ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <h2>
    {{currentEvent.name}}
  </h2>
  <p>Ticket: <strong>{{currentEvent.price | currency}}</strong></p>
  <p>Date: <strong>{{currentEvent.date}}</strong></p>
</ion-content>
```

I'm not spending much time or brain power with the UI of this page because it's going to get heavily modified in our next chapter.

And that's it if you felt this was too easy or the chapter was too simple that was the point, I wanted to create a small introduction to lists because we are going to get a little deeper in our next chapter and I wanted you to be ready for it.

And as always, if you run into any problems with the code just let me know :)

Chapter 5

Firebase Transactions

5.1 Create a revenue variable

We don't really need to create any more pages than there are, we already have everything we need from the previous chapter.

The first thing we'll need to do is to change some stuff from the previous chapter.

Go into **`event-data.ts`** and we'll update a couple of things there, first go ahead and to the **`createEvent()`** function add a revenue property and set it to the **`eventCost * (-1)`** this is going to be used for calculating real time revenue when adding new users.

```
createEvent(eventName: string, eventDate: string,
  eventPrice: number, eventCost: number): any {
  return this.eventList.push({
    name: eventName,
    date: eventDate,
    price: eventPrice,
    cost: eventCost,
    revenue: eventCost * -1
  }).then( newEvent => {
    this.eventList.child(newEvent.key).child('id').set(newEvent.key);
```

```
  });
}
```

And lastly you'll need a function to add the guests to the event so go to **`event-data.ts`** and add something simple, like:

```
addGuest(guestName, eventId, eventPrice): any {
  return this.eventList.child(eventId).child('guestList').push({
    guestName: guestName
  });
}
```

This way we are saving an event's guest list in case we need it for later.

Now we can focus on the **`event-detail`** folder, go to **`event-detail.ts`** and let's create an **`addGuest()`** function, we'll use it for adding new guests to our events, just go ahead and add it:

```
addGuest(guestName) {
  this.eventData.addGuest(guestName, this.currentEvent.id,
    this.currentEvent.price).then(() => {
    this.guestName = '';
  });
}
```

Nothing weird going on, we are just calling the addGuest function from our EventData provider, then passing it the **`guestName`**, our event's id and the event's ticket price (we'll use this last one for updating the revenue)

Since we need to add a lot of things into our event detail page, just forget about the html we wrote in **`event-detail.html`** in the previous chapter, right now today we're changing everything!

Go ahead and delete your old code, the file should look like this:

```
<ion-header>
  <ion-navbar>
    <ion-title></ion-title>
  </ion-navbar>
</ion-header>

<ion-content>

</ion-content>
```

Nothing crazy, just your regular newly generated ionic page.

The first thing we'll add is a row to show our revenue updating in real time (Yes, we'll get to the JS for that part soon) go ahead and add a row with a label and the event's revenue property:

```
<ion-row padding>
  <ion-col width-80>
    Current Revenue
  </ion-col>
  <ion-col width-20 [class.profitable]="currentEvent?.revenue > 0"
  [class.no-profit]="currentEvent?.revenue < 0">
    {{currentEvent?.revenue}}
  </ion-col>
</ion-row>
```

There we're adding the event's revenue, and we're telling Ionic, hey if the revenue is less than 0 add the **`no-profit`** css class, and if the revenue is greater than 0 go ahead and add the **`profitable`** css class.

By the way, **`profitable`** just adds green color and **`no-profit`** red color, in fact, take the css for this file, add it straight into **`event-detail.scss`**

```
.event-detail {
  ion-card-header {
    text-align: center;
  }

  .add-guest-form {
    button {
```

```
      margin-top: 16px;
    }
  }

  .profitable {
    color: #22BB22;
  }

  .no-profit {
    color: #FF0000;
  }
}
```

Now that we're back the second thing we'll add is basic event data, create a card that shows the event's name, date and ticket price:

```
<ion-card>
  <ion-card-header>
    {{currentEvent?.name}}
  </ion-card-header>
  <ion-card-content>
    <p>Ticket: <strong>{{currentEvent?.price | currency}}</strong></p>
    <p>Date: <strong>{{currentEvent?.date}}</strong></p>
  </ion-card-content>
</ion-card>
```

Very easy to understand, you are just showing your user's which event are they looking at.

And lastly we'll need a way to add our guests, create a text input for the guest's name and a button to send the info:

```
<ion-card class="add-guest-form">
  <ion-card-header>
    Add Guests
  </ion-card-header>
  <ion-card-content>
    <ion-list no-lines>
      <ion-item>
        <ion-label stacked>Name</ion-label>
        <ion-input [(ngModel)]="guestName" type="text"
```

```
          placeholder="What's your guest's name?">

        </ion-input>
      </ion-item>
      <span *ngIf="guestPicture">Picture taken!</span>

      <button ion-button color="primary" block
      (click)="addGuest(guestName)">
        Add Guest
      </button>
    </ion-list>
  </ion-card-content>
</ion-card>
```

This calls the `addGuest()` function from `event-detail.ts` and adds the guest to the event.

But Jorge, how does this updates the revenue for the event.

I wanted to leave this part for last because it needs a little explanation of why first.

I'm going to use a feature from Firebase called `transaction()` is a way to update data to ensure there's no corruption when being updated by multiple users.

For example, let's say Mary downloads the app, but she soon realizes that she needs some help at the front door, there are way to many guests and if she is the only one registering them it's going to take too long.

So she asks Kate, Mark and John for help, they download the app, login with Mary's password (Yeah, it could be a better idea to make it multi-tenant :P) and they start registering users too.

What happens in Mark & Kate both register new users, when Mark's click reads the revenue it was $300 so his app took those 300, added the $15 ticket price and the new revenue should be $315 right? Wrong!

Turns out that Kate registered someone else a millisecond earlier, so the revenue already was at $315 and Mark just set it to $315 again, you see the

problem here right?

This is where transactions come in, they update the data safely. The update function takes the current state of the data as an argument and returns the new desired state you would like to write.

If another client writes to the location before your new value is successfully written, your update function is called again with the new current value, and the write is retried.

And they are not event hard to write, just go ahead to **`event-data.ts`** and add a **`.then()`** function for the **`addGuest()`** it used to look like this:

```
addGuest(guestName, eventId, eventPrice): any {
  return this.eventList.child(eventId).child('guestList').push({
    guestName: guestName
  });
}
```

Now it should look like this:

```
addGuest(guestName, eventId, eventPrice): any {
  return this.eventList.child(eventId).child('guestList').push({
    guestName: guestName
  }).then(() => {
    this.eventList.child(eventId).child('revenue').transaction( (revenue) => {
      revenue += eventPrice;
      return event;
    });
  });
}
```

See how easy it is? The transaction takes the current state of the event and updates the revenue property.

I need to stop this chapter here, mainly because I ran out of coffee, see you in the next chapter!

Chapter 6

Firebase Storage

We already installed the Cordova plugin to use the Camera.

Now everything is ready to start working with the Camera Plugin, so go to **`event-detail.html`** and create a button to take the guest's picture, it doesn't have to be something complicated:

```
<span *ngIf="guestPicture">Picture taken!</span>
<button ion-button color="primary" block (click)="takePicture()">
  Take a Picture
</button>
```

Right after the name input I'm adding a message that says the picture was taken and it only shows if the guestPicture property exists (that will make more sense in the **`event-detail.ts`** file)

And then a button to call the **`takePicture()`** function, easy, right?

Now go to **`event-detail.ts`** and first import the Camera plugin:

```
import { Camera } from 'ionic-native';
```

After that you'll create a variable to hold the guest's picture, right before the **`constructor()`** add:

```
guestPicture: any;
```

And add that property as a parameter in the **`addGuest`** function:

Before:

```
addGuest(guestName) {
  this.eventData.addGuest(guestName, this.currentEvent.id,
    this.currentEvent.price).then(() => {
    this.guestName = '';
  });
}
```

Now:

```
addGuest(guestName) {
  this.eventData.addGuest(guestName, this.currentEvent.id,
    this.currentEvent.price, this.guestPicture).then(() => {
    this.guestName = '';
    this.guestPicture = null;
  });
}
```

We are passing the **`this.guestPicture`** variable to the **`addGuest()`** function on our **`EventData`** provider, don't worry if it gives you an error, the function isn't declare for those parameters and we'll fix that once we move to edit our provider.

Then we are setting **`this.guestPicture`** to null to make sure the message *"picture taken"* is shown.

Now we need to create the **`takePicture()`** function that's going to open the camera and allow us to take a picture of our guest, it's a long function so I'm going to paste it here and then explain the different parts of it:

```
takePicture(){
  Camera.getPicture({
    quality : 95,
    destinationType : Camera.DestinationType.DATA_URL,
    sourceType : Camera.PictureSourceType.CAMERA,
    allowEdit : true,
    encodingType: Camera.EncodingType.PNG,
    targetWidth: 500,
    targetHeight: 500,
    saveToPhotoAlbum: true
  }).then(imageData => {
    this.guestPicture = imageData;
  }, error => {
    console.log("ERROR -> " + JSON.stringify(error));
  });
}
```

The first part of the function is the call to the Camera plugin:

```
Camera.getPicture({
  quality : 95,
  destinationType : Camera.DestinationType.DATA_URL,
  sourceType : Camera.PictureSourceType.CAMERA,
  allowEdit : true,
  encodingType: Camera.EncodingType.PNG,
  targetWidth: 500,
  targetHeight: 500,
  saveToPhotoAlbum: true
})
```

There we are calling the Camera plugin and giving it a few options:

- **`quality`** => The quality we want our picture to have on a scale of 1 - 100.
- **`destinationType`** => This gives you the return type, DATA_URL is set to return a base64 string of the image, you can also return the image's URI.

- **sourceType** => We are telling it to open the camera, you can change this to get the image from the photo library.
- **allowEdit** => Allows users to edit the picture, mostly crop it.
- **encodingType**: We set the encoding for the image: png, jpg.
- **targetWidth** & **targetHeight** => This gives you the image size in px.
- **saveToPhotoAlbum** => This saves the image to the gallery after being taken.

The next part of the code is just setting that result to **this.guestPicture**.

This will now be send in the **addGuest()** function from above, so it's time to move to our provider and edit that.

Go to **event-data.ts** and create a **profilePicture** reference, first add the variable before the constructor:

```
public profilePictureRef: any;
```

Now create the reference inside the constructor, right after the eventList reference:

```
this.profilePictureRef = firebase.storage().ref('/guestProfile/');
```

Now we have a reference to our Firebase Storage under the guestProfile folder.

Next find the **addGuest()** function:

```
addGuest(guestName, eventId, eventPrice): any {
  return this.eventList.child(eventId).child('guestList').push({
    guestName: guestName
  }).then((newGuest) => {
    this.eventList.child(eventId).child('revenue').transaction( (revenue) => {
      revenue += eventPrice;
      return revenue;
    });
  });
}
```

The first thing to do here is to add the picture parameter:

```
addGuest(guestName, eventId, eventPrice, guestPicture = null)
```

I'm adding it and setting a default to null in case the guest doesn't want his picture taken.

Now we are going to add the code that takes the picture, saves it to Firebase Storage and then goes into the guest details and adds the URL for the picture we just saved:

```
if (guestPicture != null) {
  this.profilePictureRef.child(newGuest.key).child('profilePicture.png')
  .putString(guestPicture, 'base64', {contentType: 'image/png'})
  .then((savedPicture) => {
    this.eventList.child(eventId).child('guestList').child(newGuest.key)
    .child('profilePicture').set(savedPicture.downloadURL);
  });
}
```

We are creating a reference to our Firebase Storage:

`guestProfile/guestId/profilePicture.png`

And that's where we store our file, to store it we use the **`.putString()`** method passing the base64 string we got from the Camera Plugin.

After we upload the image to Firebase Storage we just create a database reference to the guest we just created and create a `profilePicture` property, then we set that property to the picture's download URL, you get that with:

```
savedPicture.downloadURL
```

And that's it, now you have a fully functional way of capturing pictures with your camera and uploading them to Firebase Storage.

See you in the next section!

Chapter 7

Security Rules

We're going to start preparing our app to go public, so the first thing we'll need to do is update our security rules in the server, we don't want people connecting to the app and having access to someone else's data.

7.1 Database Security

There's a comprehensive guide about security rules in Firebase Docs and I've kept them really simple for this app because I do believe that if you structure your data correctly they don't need to be hard.

So, to structure your security rules, you'll need to go to your firebase console:

console.firebase.google.com/project/YOURAPPNAMEHERE/database/data

By default the rules are there to allow access to only authenticated users:

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

```
  }
}
```

We need to set them so it also checks that the user trying to access the information is the **correct** user

```
{
  "rules": {
    "userProfile": {
      "$uid": {
        ".read": "auth != null && $uid === auth.uid",
        ".write": "auth != null && $uid === auth.uid"
      }
    }
  }
}
```

There we are saying that under the `userProfile` node there's going to be a variable called uid, when you add the $ sign in here it takes the value as a variable.

And we are saying that for a user to have read or write permissions under that node, their auth.uid needs to match the $uid variable.

In here auth is a variable that holds the authentication methods/properties.

There we ensure that only the user who owns the data can write/read it.

7.2 Storage Security

You should also set up rules for **Firebase Storage**, that way you can protect your users' files.

You'll need to go to:

console.firebase.google.com/project/YOURAPPGOESHERE/storage/rules

And this look a bit different:

```
// Grants a user access to a node matching their user ID
service firebase.storage {
  match /b/bill-tracker-e5746.appspot.com/o {
    // Files look like: "user/<UID>/path/to/file.txt"
    match /{userId}/{allPaths=**} {
      allow read, write: if request.auth.uid == userId;
    }
  }
}
```

It first looks that I'm pointing at the current cloud storage bucket, then that the user who uploaded the file is the only one who can read/write that file.

Chapter 8

Next Steps

By now I hope you have a greater understanding on how to work with Firebase 3 inside your Ionic 2 app.

I want you to do 1 thing.

Please send me an email letting me know that you finished the book, for real, I'd hate it if this was one of those books that sits in people's *TO READ* folder and they never get to it, so please, let me know that you did.