# Angular Forms Exercise

## Setup Application

Create a new angular application. After that got to <https://bootswatch.com> and select a theme. Download it and **import** it into your angular project.

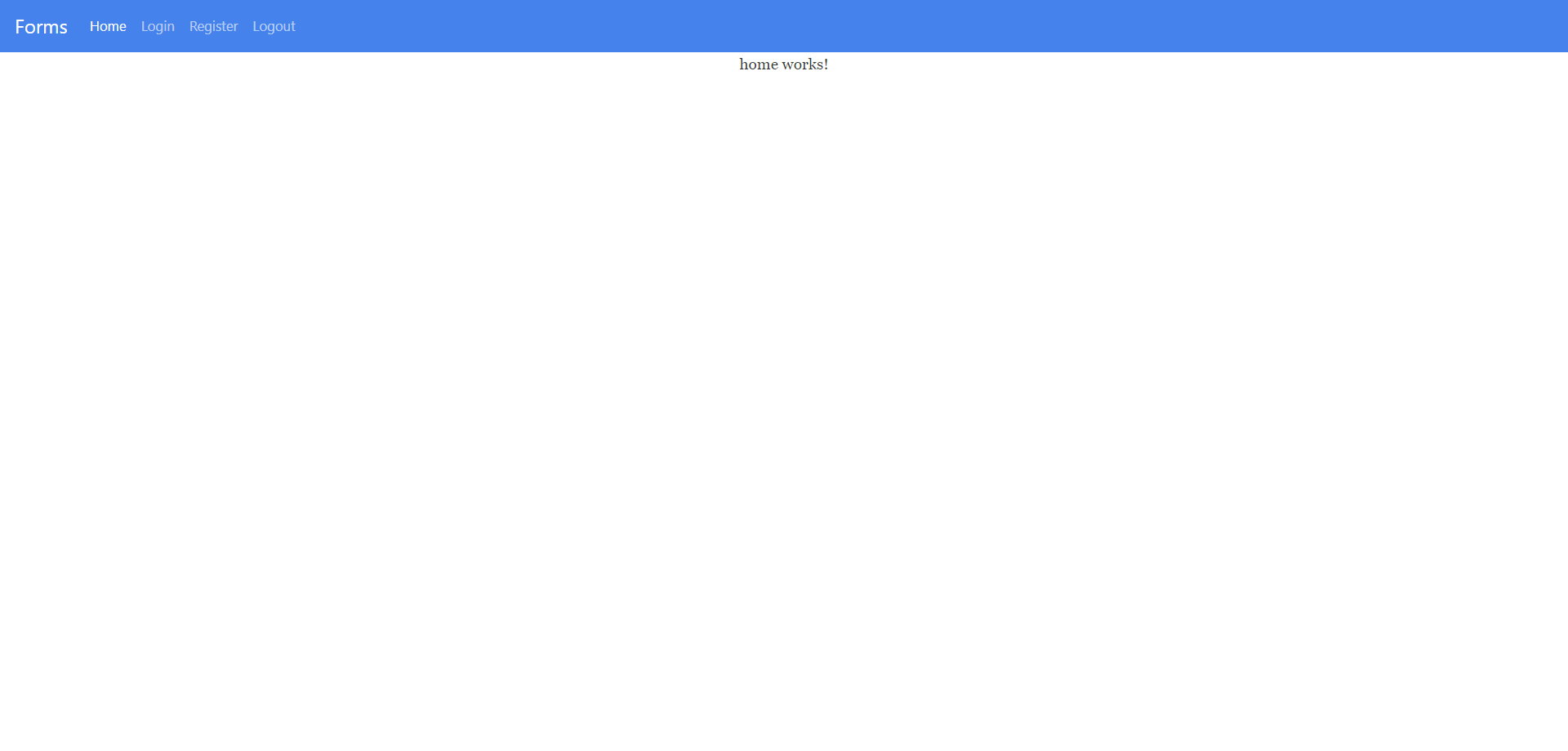
After our bootstrap theme is correctly imported we need a **navigation**. Create a **navigation component** and copy the following html in the template:

|  |
| --- |
| <nav class="navbar navbar-expand-lg navbar-dark bg-primary">  <a class="navbar-brand" href="#">Forms</a>  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarColor01" aria-controls="navbarColor01" aria-expanded="false" aria-label="Toggle navigation">  <span class="navbar-toggler-icon"></span>  </button>  <div class="collapse navbar-collapse" id="navbarColor01">  <ul class="navbar-nav mr-auto">  <li class="nav-item active">  <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>  </li>  <li class="nav-item">  <a class="nav-link">Login</a>  </li>  <li class="nav-item">  <a class="nav-link">Register</a>  </li>  <li class="nav-item">  <a class="nav-link">Logout</a>  </li>  </ul>  </div>  </nav> |

Create an **authentication** **folder** and inside generate a **login form** component and **a register form** component. We will also need a **models** folder to create our binding models for the forms. After that generate a **home component** outside the authentication folder.

We also need to setup the **routing module** and have routes that lead to home, login and register.

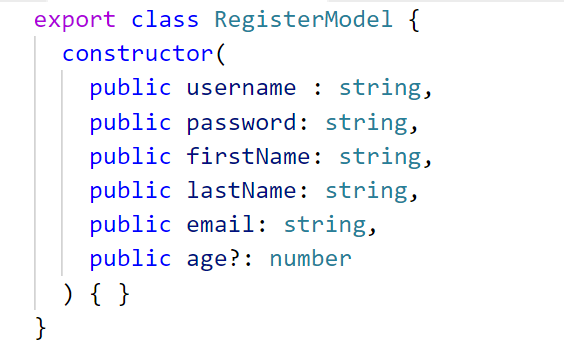
After all this is done your app should look like this and the route links should direct to the right components:



## Create Register Form

### Problem 2.1 Create a model

First off we need a **register model** that we can **bind** from the form and **send** to the API. A user should have a **username**, **password**, **first name**, **last name**, **email** and **age**. Create the model inside the models folder in authentication. It should look like this (age is optional):



After we have created our model go to the register component typescript file and **initialize** the newly created model inside it.

### Problem 2.2 Create the template

To visualize a registration form copy the following **html** inside the template:

|  |
| --- |
| <div class="container">  <h1>Registration Form</h1>  <form>  <div class="form-group">  <label for="username">Username</label>  <input type="text" class="form-control" id="username"  required>  </div>  <div class="form-group">  <label for="password">Password</label>  <input type="password" class="form-control" id="password"  required>  </div>  <div class="form-group">  <label for="confirmPassword">Confirm Password</label>  <input type="password" class="form-control" id="confirmPassword"  required>  </div>  <div class="form-group">  <label for="firstName">First Name</label>  <input type="text" class="form-control" id="firstName"  required>  </div>  <div class="form-group">  <label for="lastName">Last Name</label>  <input type="text" class="form-control" id="lastName"  required>  </div>  <div class="form-group">  <label for="email">Email Address</label>  <input type="text" class="form-control" id="email"  required>  </div>  <div class="form-group">  <label for="age">Age</label>  <input type="number" class="form-control" id="age">  </div>  <button type="submit" class="btn btn-success">Register</button>  </form>  </div> |

Our job is to add the necessary **directives**, **reference variables** and **attributes** in order to achieve a **two-way** data binding.

### Problem 2.3 Add validations

Your form should be able to **validate** all input fields and show an **error message** if the user entered incorrect information.

* Username must start with a **capital letter** and must contain **only** letters and digits.
* Password must contain **only letters and digits** and must be between **4** and **16** symbols.
* Both passwords must **match**.
* Names must contain **only letters** and start with a **capital letter**.
* There are various **email address patterns** in the internet. Find one and use it.
* Age is **optional** so it doesn’t matter.
* Disable the **register button** if something is invalid.

Attach a **submit function** to the form that we will implement later on.

## Create Login Form

Create a login form using the **same logic** as the previous one. Both input fields are **required**.

HTML template:

|  |
| --- |
| <div class="container">  <h1>Login Form</h1>  <form>  <div class="form-group">  <label for="username">Username</label>  <input type="text" class="form-control" id="username"  required>  </div>  <div class="form-group">  <label for="password">Password</label>  <input type="password" class="form-control" id="password"  required>  </div>  <button type="submit" class="btn btn-success">Login</button>  </form>  </div> |

## Authentication with Kinvey

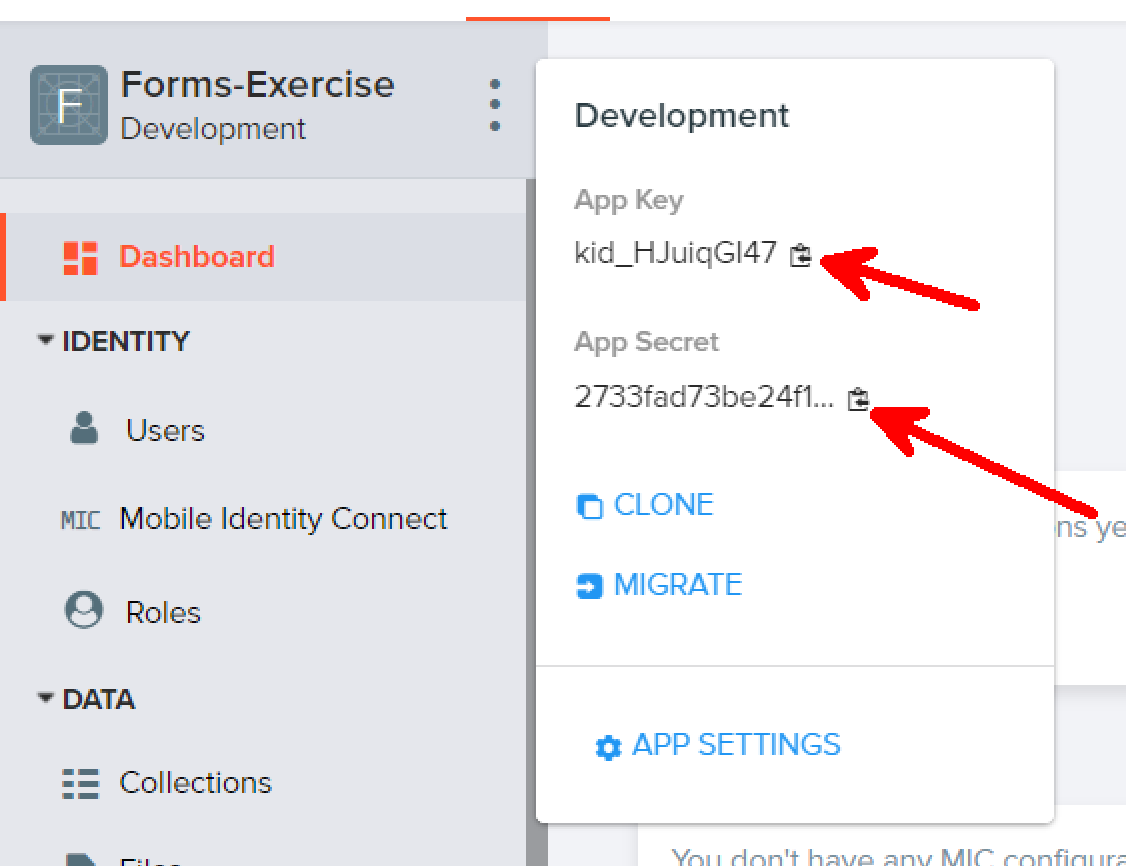
Register at <https://console.kinvey.com> and create a new app.

### Problem 4.1 Create Authentication Service

After you have created a new app we need an authentication service to register, login, logout.

Create an **authentication service** inside the **authentication folder** (don’t forget to import the HttpClientModule and provide the service).

In order to authenticate in Kinvey we need **App Key** and **App Secret**, which you can get from here:



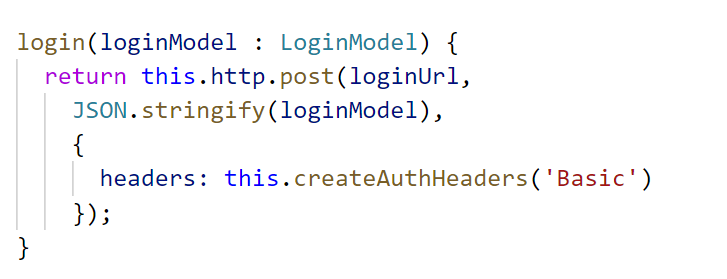
After that declare a couple of **url constants** in the service that we need in order to fetch data from the API. Don’t forget to add your app key and app secret.

|  |
| --- |
| const appKey = "" // APP KEY HERE;  const appSecret = "" // APP SECRET HERE;  const registerUrl = `https://baas.kinvey.com/user/${appKey}`;  const loginUrl = `https://baas.kinvey.com/user/${appKey}/login`;  const logoutUrl = `https://baas.kinvey.com/user/${appKey}/\_logout`; |

We need a function which retrieves **accurate http headers** according to which part of the application we want to reach. If the user is logging in or registering we need a **Basic authentication**. If the user is logging out or retrieving data that needs authentication we use **Kinvey authentication**.



Logging in requires a **body**, **url** and **http headers**. The function should look like this:

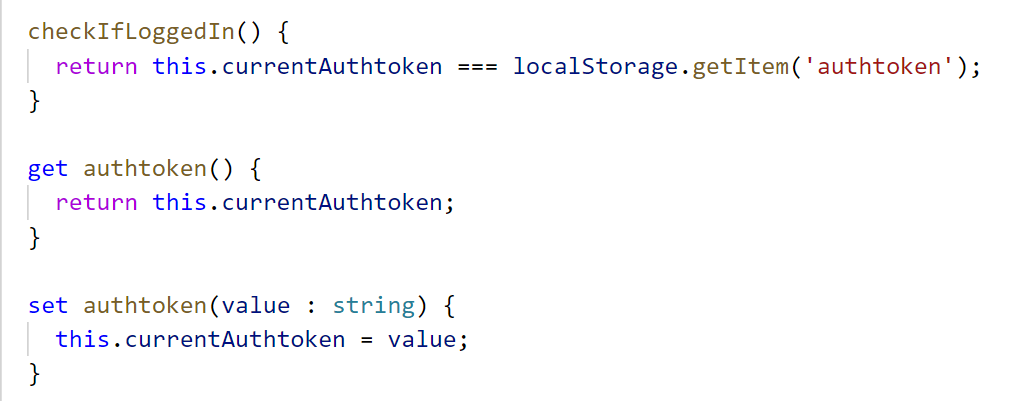


Registration is the **same** only this time we send the registration model and use the **register url**.

Logging out requires that you send an **empty body** and use **Kinvey** authentication.

****

The last piece of the puzzle is to add a **current authtoken** field. Best to make it **private** and only be able to modify it inside the service (create a **getter** and **setter** for it). Also create a function which checks if the **user is logged in**. It compares the current authtoken to the one in local storage.

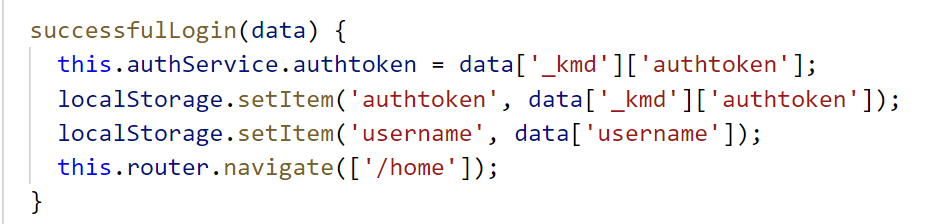


### Problem 4.2 Complete Register Component

Now that we have a service inject it into the register component and complete the **submit function**. A successful register should **redirect** to the **home page** (use Angular Router) and in case of an error show the **error description** in the html template.

### Problem 4.3 Complete Login Component

Complete the submit function inside the login component. A successful login should **save the authtoken** inside **local storage** and **redirect** to the home page and in case of an error show the **error description** in the html template. This is how you set the authentication token:



### Problem 4.4 Logout

Inside the **navigation component** create a click handler. Every time the user clicks logout we should **call** the logout function from the service and **clear** the local storage.

### Problem 4.5 Hide/Show navigation links

Think of a way to show **login/register** only for **anonymous** users and show **home/logout** only for **authenticated** users.

### Problem 4.6 Authentication Guards

Create a guard to allow access to the home page only for **authenticated users**.

import { Kinvey } from 'kinvey-angular2-sdk';

Kinvey.init({

instanceId: 'undefined',

appKey: 'kid\_rJLDV1MZB',

appSecret: '698d3c4260f442c4b340a60c3847a3c1'

});