**[[http://jasonwatmore.com/_content/images/jason-watmore.jpg](http://jasonwatmore.com/)JASON WATMORE'S BLOGA WEB DEVELOPER IN SYDNEY](http://jasonwatmore.com/)**

* [HOME](http://jasonwatmore.com/)
* [CODE SNIPPETS](http://jasonwatmore.com/page/code-snippets)
* [ARCHIVE](http://jasonwatmore.com/archive)
* [CONTACT](http://jasonwatmore.com/contact)

**SEPTEMBER 29 2016**

**Angular 2/5 User Registration and Login Example & Tutorial**

November 23 2017 - Updated to **Angular 5.0.3** for both Webpack and SystemJS versions.

After getting a lot of interest in a previous tutorial I posted on how to build a [User Registration and Login with Angular 1](http://jasonwatmore.com/post/2015/03/10/angularjs-user-registration-and-login-example-tutorial), and since Angular 2 Final was recently released I thought it was time to post an updated example built with Angular 2 and TypeScript.

The project is available on GitHub at <https://github.com/cornflourblue/angular2-registration-login-example>.

Webpack version is available at <https://github.com/cornflourblue/angular2-registration-login-example-webpack>.

Angular CLI version is available at <https://github.com/cornflourblue/angular2-registration-login-example-cli>.

The example is the boilerplate front end of a secure web application that I developed for a law firm in Sydney recently, it uses a fake backend that stores users in HTML5 local storage, to switch to using a real web service simply remove the fake backend providers in the app.module.ts file below the comment "// providers used to create fake backend".

Here it is in action:(See on Plunker at <http://plnkr.co/edit/9luTng?p=preview>)

Update History:

* 23 Nov 2017 - Updated to **Angular 5.0.3** for both Webpack and SystemJS versions
* 16 Sep 2017 - For the same example built with React and Redux check out [React + Redux - User Registration and Login Tutorial & Example](http://jasonwatmore.com/post/2017/09/16/react-redux-user-registration-and-login-tutorial-example)
* 11 May 2017 - For the same example with a real backend ASP.NET Core Web API check out [ASP.NET Core + Angular 2/4 - User Registration and Login Tutorial & Example](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-angular-24-user-registration-and-login-tutorial-example) (on my company blog - Point Blank Web Develpoment Sydney)
* 27 Apr 2017 - Updated to **Angular 4.1.0** for both Webpack and SystemJS versions
* 16 Mar 2017 - Created Webpack version of example - [Webpack version on GitHub](https://github.com/cornflourblue/angular2-registration-login-example-webpack" \t "_blank)
* 16 Mar 2017 - Updated to **Angular 2.4.9**
* 24 Feb 2017 - For the same example with a real backend MEAN Stack Web API check out [MEAN with Angular 2 - User Registration and Login Example & Tutorial](http://jasonwatmore.com/post/2017/02/22/mean-with-angular-2-user-registration-and-login-example-tutorial)
* 08 Dec 2016 - Updated Auth Guard and Login Component to redirect user back to previous / original url after login.
* 19 Nov 2016 - Updated to **Angular 2.2.1**

**Running the Angular 2/5 User Registration & Login Example**

This version of the example uses SystemJS as the module loader.

1. Install NodeJS (> v4) and NPM (> v3) from <https://nodejs.org/en/download/>, you can check the versions you have installed by running node -v and npm -v from the command line.
2. Download the project source code from <https://github.com/cornflourblue/angular2-registration-login-example>
3. Install all required npm packages by running npm install from the command line in the project root folder (where the package.json is located).
4. Start the application by running npm start from the command line in the project root folder.

**Running the Webpack Version of the Angular 2/5 Example**

This version of the example uses Webpack to bundle the angular 2 modules together and perform other build tasks, the structure is based on the [Angular 2 Webpack Introduction](https://angular.io/docs/ts/latest/guide/webpack.html) on the Angular 2 docs site.

The Webpack Dev Server is used as the local web server for this version.

1. Install NodeJS (> v4) and NPM (> v3) from <https://nodejs.org/en/download/>, you can check the versions you have installed by running node -v and npm -v from the command line.
2. Download the project source code from <https://github.com/cornflourblue/angular2-registration-login-example-webpack>
3. Install all required npm packages by running npm install from the command line in the project root folder (where the package.json is located).
4. Start the application by running npm start from the command line in the project root folder.
5. Browse to http://localhost:8080 to test your application.

**Running the Angular CLI Version of the Angular 2/5 Example**

This version is pretty much the same as the Webpack version above, I've just copied it into the project structure generated by Angular CLI (1.5.4) to make it easier for anybody that's using Angular CLI.

For more information about Angular CLI check out the official website at <https://cli.angular.io/>.

1. Install NodeJS (> v6.9) and NPM (> v3) from <https://nodejs.org/en/download/>, you can check the versions you have installed by running node -v and npm -v from the command line.
2. Install Angular CLI by running npm install -g @angular/cli
3. Download the project source code from <https://github.com/cornflourblue/angular2-registration-login-example-cli>
4. Install all required npm packages by running npm install from the command line in the project root folder (where the package.json is located).
5. Start the application by running ng servefrom the command line in the project root folder.
6. Browse to http://localhost:4200 to test your application.

**Angular 2/5 Project Structure**

I used the [Angular 2 quickstart](https://angular.io/docs/ts/latest/quickstart.html) project as a base for the application, it's written in TypeScript and uses systemjs for loading modules. If you're new to angular 2 I'd recommend checking out the quickstart as it provides details on the project tooling and configuration files which aren't covered in this post.

The project and code structure mostly follows the recommendations in the official [Angular 2 style guide](https://angular.io/styleguide), with my own tweaks here and there.

Each feature has it's own folder (home & login), other code such as services, models, guards etc are placed in folders prefixed with an underscore to easily differentiate them and group them together at the top of the folder structure.

Here's the project structure:

* app
  + \_directives
    - [alert.component.html](http://jasonwatmore.com/#alert-component-html)
    - [alert.component.ts](http://jasonwatmore.com/#alert-component-ts)
    - index.ts
  + \_guards
    - [auth.guard.ts](http://jasonwatmore.com/#auth-guard-ts)
    - index.ts
  + \_helpers
    - [fake-backend.ts](http://jasonwatmore.com/#fake-backend-ts)
    - index.ts
  + \_models
    - [user.ts](http://jasonwatmore.com/#user-ts)
    - index.ts
  + \_services
    - [alert.service.ts](http://jasonwatmore.com/#alert-service-ts)
    - [authentication.service.ts](http://jasonwatmore.com/#authentication-service-ts)
    - index.ts
    - [user.service.ts](http://jasonwatmore.com/#user-service-ts)
  + home
    - [home.component.html](http://jasonwatmore.com/#home-component-html)
    - [home.component.ts](http://jasonwatmore.com/#home-component-ts)
    - index.ts
  + login
    - index.ts
    - [login.component.html](http://jasonwatmore.com/#login-component-html)
    - [login.component.ts](http://jasonwatmore.com/#login-component-ts)
  + register
    - index.ts
    - [register.component.html](http://jasonwatmore.com/#register-component-html)
    - [register.component.ts](http://jasonwatmore.com/#register-component-ts)
  + [app.component.html](http://jasonwatmore.com/#app-component-html)
  + [app.component.ts](http://jasonwatmore.com/#app-component-ts)
  + [app.module.ts](http://jasonwatmore.com/#app-module-ts)
  + [app.routing.ts](http://jasonwatmore.com/#app-routing-ts)
  + [main.ts](http://jasonwatmore.com/#main-ts)
* app.css
* index.html
* package.json
* system.config.js
* tsconfig.json

Below are brief descriptions and the code for the main files of the example application, all files are available in the github project linked at the top of the post.

**Angular 2/5 Alert Component Template**

**Path: /app/\_directives/alert.component.html**

The alert component template contains the html for displaying alert messages at the top of the page.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1 | <div \*ngIf="message" [ngClass]="{ 'alert': message, 'alert-success': message.type === 'success', 'alert-danger': message.type === 'error' }">{{message.text}}</div> |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Alert Component**

**Path: /app/\_directives/alert.component.ts**

The alert component passes alert messages to the template whenever a message is received from the alert service. It does this by subscribing to the alert service's getMessage() method which returns an Observable.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import { Component, OnInit } from '@angular/core';    import { AlertService } from '../\_services/index';    @Component({      moduleId: module.id,      selector: 'alert',      templateUrl: 'alert.component.html'  })    export class AlertComponent {      message: any;        constructor(private alertService: AlertService) { }        ngOnInit() {          this.alertService.getMessage().subscribe(message => { this.message = message; });      }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Auth Guard**

**Path: /app/\_guards/auth.guard.ts**

The auth guard is used to prevent unauthenticated users from accessing restricted routes, in this example it's used in app.routing.ts to protect the home page route. For more information about angular 2 guards you can check out [this post](http://blog.thoughtram.io/angular/2016/07/18/guards-in-angular-2.html) on the thoughtram blog.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import { Injectable } from '@angular/core';  import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';    @Injectable()  export class AuthGuard implements CanActivate {        constructor(private router: Router) { }        canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {          if (localStorage.getItem('currentUser')) {              // logged in so return true              return true;          }            // not logged in so redirect to login page with the return url          this.router.navigate(['/login'], { queryParams: { returnUrl: state.url }});          return false;      }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Fake Backend Provider**

**Path: /app/\_helpers/fake-backend.ts**

The fake backend provider enables the example to run without a backend / backendless, it uses HTML5 local storage for storing registered user data and provides fake implementations for authentication and CRUD methods, these would be handled by a real api and database in a production application.

It uses the Angular 2 MockBackend to replace the default backend used by the Http service, the MockBackend enables you to intercept http requests made within the application and provide fake responses, it's also used for unit testing.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156 | import { Http, BaseRequestOptions, Response, ResponseOptions, RequestMethod, XHRBackend, RequestOptions } from '@angular/http';  import { MockBackend, MockConnection } from '@angular/http/testing';    export function fakeBackendFactory(backend: MockBackend, options: BaseRequestOptions, realBackend: XHRBackend) {      // array in local storage for registered users      let users: any[] = JSON.parse(localStorage.getItem('users')) || [];        // configure fake backend      backend.connections.subscribe((connection: MockConnection) => {          // wrap in timeout to simulate server api call          setTimeout(() => {                // authenticate              if (connection.request.url.endsWith('/api/authenticate') && connection.request.method === RequestMethod.Post) {                  // get parameters from post request                  let params = JSON.parse(connection.request.getBody());                    // find if any user matches login credentials                  let filteredUsers = users.filter(user => {                      return user.username === params.username && user.password === params.password;                  });                    if (filteredUsers.length) {                      // if login details are valid return 200 OK with user details and fake jwt token                      let user = filteredUsers[0];                      connection.mockRespond(new Response(new ResponseOptions({                          status: 200,                          body: {                              id: user.id,                              username: user.username,                              firstName: user.firstName,                              lastName: user.lastName,                              token: 'fake-jwt-token'                          }                      })));                  } else {                      // else return 400 bad request                      connection.mockError(new Error('Username or password is incorrect'));                  }                    return;              }                // get users              if (connection.request.url.endsWith('/api/users') && connection.request.method === RequestMethod.Get) {                  // check for fake auth token in header and return users if valid, this security is implemented server side in a real application                  if (connection.request.headers.get('Authorization') === 'Bearer fake-jwt-token') {                      connection.mockRespond(new Response(new ResponseOptions({ status: 200, body: users })));                  } else {                      // return 401 not authorised if token is null or invalid                      connection.mockRespond(new Response(new ResponseOptions({ status: 401 })));                  }                    return;              }                // get user by id              if (connection.request.url.match(/\/api\/users\/\d+$/) && connection.request.method === RequestMethod.Get) {                  // check for fake auth token in header and return user if valid, this security is implemented server side in a real application                  if (connection.request.headers.get('Authorization') === 'Bearer fake-jwt-token') {                      // find user by id in users array                      let urlParts = connection.request.url.split('/');                      let id = parseInt(urlParts[urlParts.length - 1]);                      let matchedUsers = users.filter(user => { return user.id === id; });                      let user = matchedUsers.length ? matchedUsers[0] : null;                        // respond 200 OK with user                      connection.mockRespond(new Response(new ResponseOptions({ status: 200, body: user })));                  } else {                      // return 401 not authorised if token is null or invalid                      connection.mockRespond(new Response(new ResponseOptions({ status: 401 })));                  }                    return;              }                // create user              if (connection.request.url.endsWith('/api/users') && connection.request.method === RequestMethod.Post) {                  // get new user object from post body                  let newUser = JSON.parse(connection.request.getBody());                    // validation                  let duplicateUser = users.filter(user => { return user.username === newUser.username; }).length;                  if (duplicateUser) {                      return connection.mockError(new Error('Username "' + newUser.username + '" is already taken'));                  }                    // save new user                  newUser.id = users.length + 1;                  users.push(newUser);                  localStorage.setItem('users', JSON.stringify(users));                    // respond 200 OK                  connection.mockRespond(new Response(new ResponseOptions({ status: 200 })));                    return;              }                // delete user              if (connection.request.url.match(/\/api\/users\/\d+$/) && connection.request.method === RequestMethod.Delete) {                  // check for fake auth token in header and return user if valid, this security is implemented server side in a real application                  if (connection.request.headers.get('Authorization') === 'Bearer fake-jwt-token') {                      // find user by id in users array                      let urlParts = connection.request.url.split('/');                      let id = parseInt(urlParts[urlParts.length - 1]);                      for (let i = 0; i < users.length; i++) {                          let user = users[i];                          if (user.id === id) {                              // delete user                              users.splice(i, 1);                              localStorage.setItem('users', JSON.stringify(users));                              break;                          }                      }                        // respond 200 OK                      connection.mockRespond(new Response(new ResponseOptions({ status: 200 })));                  } else {                      // return 401 not authorised if token is null or invalid                      connection.mockRespond(new Response(new ResponseOptions({ status: 401 })));                  }                    return;              }                // pass through any requests not handled above              let realHttp = new Http(realBackend, options);              let requestOptions = new RequestOptions({                  method: connection.request.method,                  headers: connection.request.headers,                  body: connection.request.getBody(),                  url: connection.request.url,                  withCredentials: connection.request.withCredentials,                  responseType: connection.request.responseType              });              realHttp.request(connection.request.url, requestOptions)                  .subscribe((response: Response) => {                      connection.mockRespond(response);                  },                  (error: any) => {                      connection.mockError(error);                  });            }, 500);        });        return new Http(backend, options);  };    export let fakeBackendProvider = {      // use fake backend in place of Http service for backend-less development      provide: Http,      useFactory: fakeBackendFactory,      deps: [MockBackend, BaseRequestOptions, XHRBackend]  }; |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2 User Model**

**Path: /app/\_models/user.ts**

The user model is a small class that defines the properties of a user.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | export class User {      id: number;      username: string;      password: string;      firstName: string;      lastName: string;  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Alert Service**

**Path: /app/\_services/alert.service.ts**

The alert service enables any component in the application to display alert messages at the top of the page via the alert component.

It has methods for displaying success and error messages, and a getMessage() method that returns an Observable that is used by the alert component to subscribe to notifications for whenever a message should be displayed.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | import { Injectable } from '@angular/core';  import { Router, NavigationStart } from '@angular/router';  import { Observable } from 'rxjs';  import { Subject } from 'rxjs/Subject';    @Injectable()  export class AlertService {      private subject = new Subject<any>();      private keepAfterNavigationChange = false;        constructor(private router: Router) {          // clear alert message on route change          router.events.subscribe(event => {              if (event instanceof NavigationStart) {                  if (this.keepAfterNavigationChange) {                      // only keep for a single location change                      this.keepAfterNavigationChange = false;                  } else {                      // clear alert                      this.subject.next();                  }              }          });      }        success(message: string, keepAfterNavigationChange = false) {          this.keepAfterNavigationChange = keepAfterNavigationChange;          this.subject.next({ type: 'success', text: message });      }        error(message: string, keepAfterNavigationChange = false) {          this.keepAfterNavigationChange = keepAfterNavigationChange;          this.subject.next({ type: 'error', text: message });      }        getMessage(): Observable<any> {          return this.subject.asObservable();      }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Authentication Service**

**Path: /app/\_services/authentication.service.ts**

The authentication service is used to login and logout of the application, to login it posts the users credentials to the api and checks the response for a JWT token, if there is one it means authentication was successful so the user details including the token are added to local storage.

The logged in user details are stored in local storage so the user will stay logged in if they refresh the browser and also between browser sessions until they logout. If you don't want the user to stay logged in between refreshes or sessions the behaviour could easily be changed by storing user details somewhere less persistent such as session storage or in a property of the authentication service.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import { Injectable } from '@angular/core';  import { Http, Headers, Response } from '@angular/http';  import { Observable } from 'rxjs/Observable';  import 'rxjs/add/operator/map'    @Injectable()  export class AuthenticationService {      constructor(private http: Http) { }        login(username: string, password: string) {          return this.http.post('/api/authenticate', JSON.stringify({ username: username, password: password }))              .map((response: Response) => {                  // login successful if there's a jwt token in the response                  let user = response.json();                  if (user && user.token) {                      // store user details and jwt token in local storage to keep user logged in between page refreshes                      localStorage.setItem('currentUser', JSON.stringify(user));                  }                    return user;              });      }        logout() {          // remove user from local storage to log user out          localStorage.removeItem('currentUser');      }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 User Service**

**Path: /app/\_services/user.service.ts**

The user service contains a standard set of CRUD methods for managing users, it contains a jwt() method that's used to add the JWT token from local storage to the Authorization header of each http request.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | import { Injectable } from '@angular/core';  import { Http, Headers, RequestOptions, Response } from '@angular/http';    import { User } from '../\_models/index';    @Injectable()  export class UserService {      constructor(private http: Http) { }        getAll() {          return this.http.get('/api/users', this.jwt()).map((response: Response) => response.json());      }        getById(id: number) {          return this.http.get('/api/users/' + id, this.jwt()).map((response: Response) => response.json());      }        create(user: User) {          return this.http.post('/api/users', user, this.jwt()).map((response: Response) => response.json());      }        update(user: User) {          return this.http.put('/api/users/' + user.id, user, this.jwt()).map((response: Response) => response.json());      }        delete(id: number) {          return this.http.delete('/api/users/' + id, this.jwt()).map((response: Response) => response.json());      }        // private helper methods        private jwt() {          // create authorization header with jwt token          let currentUser = JSON.parse(localStorage.getItem('currentUser'));          if (currentUser && currentUser.token) {              let headers = new Headers({ 'Authorization': 'Bearer ' + currentUser.token });              return new RequestOptions({ headers: headers });          }      }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Home Component Template**

**Path: /app/home/home.component.html**

The home component template contains html and angular 2 template syntax for displaying a simple welcome message, a list of users and a logout link.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <div class="col-md-6 col-md-offset-3">      <h1>Hi {{currentUser.firstName}}!</h1>      <p>You're logged in with Angular 2!!</p>      <h3>All registered users:</h3>      <ul>          <li \*ngFor="let user of users">              {{user.username}} ({{user.firstName}} {{user.lastName}})              - <a (click)="deleteUser(user.id)">Delete</a>          </li>      </ul>      <p><a [routerLink]="['/login']">Logout</a></p>  </div> |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Home Component**

**Path: /app/home/home.component.ts**

The home component gets the current user from local storage and all users from the user service, and makes them available to the template.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | import { Component, OnInit } from '@angular/core';    import { User } from '../\_models/index';  import { UserService } from '../\_services/index';    @Component({      moduleId: module.id,      templateUrl: 'home.component.html'  })    export class HomeComponent implements OnInit {      currentUser: User;      users: User[] = [];        constructor(private userService: UserService) {          this.currentUser = JSON.parse(localStorage.getItem('currentUser'));      }        ngOnInit() {          this.loadAllUsers();      }        deleteUser(id: number) {          this.userService.delete(id).subscribe(() => { this.loadAllUsers() });      }        private loadAllUsers() {          this.userService.getAll().subscribe(users => { this.users = users; });      }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Login Component Template**

**Path: /app/login/login.component.html**

The login component template contains a login form with username and password fields. It displays validation messages for invalid fields when the submit button is clicked. On submit the login() method is called as long as the form is valid.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <div class="col-md-6 col-md-offset-3">      <h2>Login</h2>      <form name="form" (ngSubmit)="f.form.valid && login()" #f="ngForm" novalidate>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !username.valid }">              <label for="username">Username</label>              <input type="text" class="form-control" name="username" [(ngModel)]="model.username" #username="ngModel" required />              <div \*ngIf="f.submitted && !username.valid" class="help-block">Username is required</div>          </div>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !password.valid }">              <label for="password">Password</label>              <input type="password" class="form-control" name="password" [(ngModel)]="model.password" #password="ngModel" required />              <div \*ngIf="f.submitted && !password.valid" class="help-block">Password is required</div>          </div>          <div class="form-group">              <button [disabled]="loading" class="btn btn-primary">Login</button>              <img \*ngIf="loading" src="data:image/gif;base64," />              <a [routerLink]="['/register']" class="btn btn-link">Register</a>          </div>      </form>  </div> |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Login Component**

**Path: /app/login/login.component.ts**

The login component uses the authentication service to login and logout of the application. It automatically logs the user out when it initializes (ngOnInit) so the login page can also be used to logout.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | import { Component, OnInit } from '@angular/core';  import { Router, ActivatedRoute } from '@angular/router';    import { AlertService, AuthenticationService } from '../\_services/index';    @Component({      moduleId: module.id,      templateUrl: 'login.component.html'  })    export class LoginComponent implements OnInit {      model: any = {};      loading = false;      returnUrl: string;        constructor(          private route: ActivatedRoute,          private router: Router,          private authenticationService: AuthenticationService,          private alertService: AlertService) { }        ngOnInit() {          // reset login status          this.authenticationService.logout();            // get return url from route parameters or default to '/'          this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';      }        login() {          this.loading = true;          this.authenticationService.login(this.model.username, this.model.password)              .subscribe(                  data => {                      this.router.navigate([this.returnUrl]);                  },                  error => {                      this.alertService.error(error);                      this.loading = false;                  });      }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Register Component Template**

**Path: /app/register/register.component.html**

The register component template contains a simplae registration form with fields for first name, last name, username and password. It displays validation messages for invalid fields when the submit button is clicked. On submit the register() method is called if the form is valid.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <div class="col-md-6 col-md-offset-3">      <h2>Register</h2>      <form name="form" (ngSubmit)="f.form.valid && register()" #f="ngForm" novalidate>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !username.valid }">              <label for="firstName">First Name</label>              <input type="text" class="form-control" name="firstName" [(ngModel)]="model.firstName" #firstName="ngModel" required />              <div \*ngIf="f.submitted && !firstName.valid" class="help-block">First Name is required</div>          </div>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !username.valid }">              <label for="lastName">Last Name</label>              <input type="text" class="form-control" name="lastName" [(ngModel)]="model.lastName" #lastName="ngModel" required />              <div \*ngIf="f.submitted && !lastName.valid" class="help-block">Last Name is required</div>          </div>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !username.valid }">              <label for="username">Username</label>              <input type="text" class="form-control" name="username" [(ngModel)]="model.username" #username="ngModel" required />              <div \*ngIf="f.submitted && !username.valid" class="help-block">Username is required</div>          </div>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !password.valid }">              <label for="password">Password</label>              <input type="password" class="form-control" name="password" [(ngModel)]="model.password" #password="ngModel" required />              <div \*ngIf="f.submitted && !password.valid" class="help-block">Password is required</div>          </div>          <div class="form-group">              <button [disabled]="loading" class="btn btn-primary">Register</button>              <img \*ngIf="loading" src="data:image/gif;base64," />              <a [routerLink]="['/login']" class="btn btn-link">Cancel</a>          </div>      </form>  </div> |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Register Component**

**Path: /app/register/register.component.ts**

The register component has a single register() method that creates a new user with the user service when the register form is submitted.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | import { Component } from '@angular/core';  import { Router } from '@angular/router';    import { AlertService, UserService } from '../\_services/index';    @Component({      moduleId: module.id,      templateUrl: 'register.component.html'  })    export class RegisterComponent {      model: any = {};      loading = false;        constructor(          private router: Router,          private userService: UserService,          private alertService: AlertService) { }        register() {          this.loading = true;          this.userService.create(this.model)              .subscribe(                  data => {                      // set success message and pass true paramater to persist the message after redirecting to the login page                      this.alertService.success('Registration successful', true);                      this.router.navigate(['/login']);                  },                  error => {                      this.alertService.error(error);                      this.loading = false;                  });      }  } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 App Component Template**

**Path: /app/app.component.html**

The app component template is the root component template of the application, it contains a router-outlet directive for displaying the contents of each view based on the current route, and an alert directive for displaying alert messages from anywhere in the system.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <!-- main app container -->  <div class="jumbotron">      <div class="container">          <div class="col-sm-8 col-sm-offset-2">              <alert></alert>              <router-outlet></router-outlet>          </div>      </div>  </div> |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 App Component**

**Path: /app/app.component.ts**

The app component is the root component of the application, it defines the root tag of the app as <app></app> with the selector property.

The moduleId property is set to allow a relative path to be used for the templateUrl.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import { Component } from '@angular/core';    @Component({      moduleId: module.id,      selector: 'app',      templateUrl: 'app.component.html'  })    export class AppComponent { } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 App Module**

**Path: /app/app.module.ts**

The app module defines the root module of the application along with metadata about the module. For more info about angular 2 modules check out [this page](https://angular.io/docs/ts/latest/guide/ngmodule.html) on the official docs site.

This is where the fake backend provider is added to the application, to switch to a real backend simply remove the providers located under the comment "// providers used to create fake backend".

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | import { NgModule }      from '@angular/core';  import { BrowserModule } from '@angular/platform-browser';  import { FormsModule }    from '@angular/forms';  import { HttpModule } from '@angular/http';    // used to create fake backend  import { fakeBackendProvider } from './\_helpers/index';  import { MockBackend, MockConnection } from '@angular/http/testing';  import { BaseRequestOptions } from '@angular/http';    import { AppComponent }  from './app.component';  import { routing }        from './app.routing';    import { AlertComponent } from './\_directives/index';  import { AuthGuard } from './\_guards/index';  import { AlertService, AuthenticationService, UserService } from './\_services/index';  import { HomeComponent } from './home/index';  import { LoginComponent } from './login/index';  import { RegisterComponent } from './register/index';    @NgModule({      imports: [          BrowserModule,          FormsModule,          HttpModule,          routing      ],      declarations: [          AppComponent,          AlertComponent,          HomeComponent,          LoginComponent,          RegisterComponent      ],      providers: [          AuthGuard,          AlertService,          AuthenticationService,          UserService,            // providers used to create fake backend          fakeBackendProvider,          MockBackend,          BaseRequestOptions      ],      bootstrap: [AppComponent]  })    export class AppModule { } |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 App Routing**

**Path: /app/app.routing.ts**

The app routing file defines the routes of the application, each route contains a path and associated component. The home route is secured by passing the AuthGuard to the canActivate property of the route.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import { Routes, RouterModule } from '@angular/router';    import { HomeComponent } from './home/index';  import { LoginComponent } from './login/index';  import { RegisterComponent } from './register/index';  import { AuthGuard } from './\_guards/index';    const appRoutes: Routes = [      { path: '', component: HomeComponent, canActivate: [AuthGuard] },      { path: 'login', component: LoginComponent },      { path: 'register', component: RegisterComponent },        // otherwise redirect to home      { path: '\*\*', redirectTo: '' }  ];    export const routing = RouterModule.forRoot(appRoutes); |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Angular 2/5 Main (Bootstrap) File**

**Path: /app/main.ts**

The main file is the entry point used by angular to launch and bootstrap the application.

[?](http://jasonwatmore.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';    import { AppModule } from './app.module';    platformBrowserDynamic().bootstrapModule(AppModule); |

[Back to top](http://jasonwatmore.com/#projectstructure)

**Recommended Books on Angular 2/5**

* [Angular 2 Cookbook - Second Edition](https://www.amazon.com/gp/product/1785881922/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=1785881922&linkCode=as2&tag=jasonwatmore-20&linkId=62e30a6d3d356a2ae92c6e1ac85f9ba4) by Matt Frisbie (a developer at Google)
* [Angular 2 Development with TypeScript](https://www.amazon.com/gp/product/1617293121/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=1617293121&linkCode=as2&tag=jasonwatmore-20&linkId=9733d68da7d94b5334324509b264c25e) by Yakov Fain and Anton Moiseev

**Web Development Sydney**

Feel free to [drop me a line](http://jasonwatmore.com/contact) if you're looking for a web developer in Sydney Australia, I also provide remote contracting services if you're outside Sydney.

Tags: [Angular 2](http://jasonwatmore.com/posts/tag/angular-2), [Angular 4](http://jasonwatmore.com/posts/tag/angular-4), [Angular 5](http://jasonwatmore.com/posts/tag/angular-5), [TypeScript](http://jasonwatmore.com/posts/tag/typescript), [Login](http://jasonwatmore.com/posts/tag/login), [Registration](http://jasonwatmore.com/posts/tag/registration), [Authentication and Authorization](http://jasonwatmore.com/posts/tag/authentication-and-authorization)

Share: [Facebook](http://facebook.com/sharer.php?u=http://jasonwatmore.com/post/2016/09/29/angular-2-user-registration-and-login-example-tutorial) [Twitter](http://twitter.com/intent/tweet?url=http://jasonwatmore.com/post/2016/09/29/angular-2-user-registration-and-login-example-tutorial&text=Angular%202/5%20User%20Registration%20and%20Login%20Example%20&%20Tutorial) [Google+](https://plus.google.com/share?url=/post/2016/09/29/angular-2-user-registration-and-login-example-tutorial)

**More Angular 2 Posts**

* [Angular 2/5 - Alert (Toaster) Notifications](http://jasonwatmore.com/post/2017/06/25/angular-2-4-alert-toaster-notifications)
* [Upgrading from Angular 2 to Angular 4](http://jasonwatmore.com/post/2017/05/20/upgrading-from-angular-2-to-angular-4)
* [Angular 2/5 - Router Animation Tutorial & Example](http://jasonwatmore.com/post/2017/04/19/angular-2-4-router-animation-tutorial-example)
* [Angular 2 - Refresh Without 404 in Node & IIS](http://jasonwatmore.com/post/2017/02/24/angular-2-refresh-without-404-in-node-iis)
* [MEAN with Angular 2/4 - User Registration and Login Example & Tutorial](http://jasonwatmore.com/post/2017/02/22/mean-with-angular-2-user-registration-and-login-example-tutorial)
* [Angular 2/5 - Custom Modal Window / Dialog Box](http://jasonwatmore.com/post/2017/01/24/angular-2-custom-modal-window-dialog-box)
* [Angular 2 - Redirect to Previous URL after Login with Auth Guard](http://jasonwatmore.com/post/2016/12/08/angular-2-redirect-to-previous-url-after-login-with-auth-guard)
* [Angular 2/5 - Communicating Between Components with Observable & Subject](http://jasonwatmore.com/post/2016/12/01/angular-2-communicating-between-components-with-observable-subject)
* [Angular 2 - MockBackend Example for Backendless Development](http://jasonwatmore.com/post/2016/11/24/angular-2-mockbackend-example-for-backendless-development)
* [Angular 1 vs Angular 2+ - Comparing Examples to Learn Angular 2+](http://jasonwatmore.com/post/2016/11/20/angular-1-vs-angular-2-comparing-examples-to-learn-angular-2)
* [Angular 2/5 - Pagination Example with Logic like Google](http://jasonwatmore.com/post/2016/08/23/angular-2-pagination-example-with-logic-like-google)
* [Angular 2/5 JWT Authentication Example & Tutorial](http://jasonwatmore.com/post/2016/08/16/angular-2-jwt-authentication-example-tutorial)

**ABOUT**

I'm a web developer in Sydney Australia and the technical lead at [Point Blank Development](https://www.pointblankdevelopment.com.au/), I've been building websites and web applications in Sydney since 1998.

Find me on:

**MONTHS**

* 2017
  + [December](http://jasonwatmore.com/posts/2017/12) (1)
  + [September](http://jasonwatmore.com/posts/2017/09) (1)
  + [June](http://jasonwatmore.com/posts/2017/06) (1)
  + [May](http://jasonwatmore.com/posts/2017/05) (1)
  + [April](http://jasonwatmore.com/posts/2017/04) (1)
  + [March](http://jasonwatmore.com/posts/2017/03) (2)
  + [February](http://jasonwatmore.com/posts/2017/02) (2)
  + [January](http://jasonwatmore.com/posts/2017/01) (1)
* 2016
* 2015
* 2014
* 2013
* 2012
* 2011

**TAGS**

* [Angular 2](http://jasonwatmore.com/posts/tag/angular-2)
* [Angular 4](http://jasonwatmore.com/posts/tag/angular-4)
* [Angular 5](http://jasonwatmore.com/posts/tag/angular-5)
* [Angular Directive](http://jasonwatmore.com/posts/tag/angular-directive)
* [Angular UI Router](http://jasonwatmore.com/posts/tag/angular-ui-router)
* [AngularJS](http://jasonwatmore.com/posts/tag/angularjs)
* [Animation](http://jasonwatmore.com/posts/tag/animation)
* [ASP.NET](http://jasonwatmore.com/posts/tag/aspnet)
* [ASP.NET Web API](http://jasonwatmore.com/posts/tag/aspnet-web-api)
* [Authentication and Authorization](http://jasonwatmore.com/posts/tag/authentication-and-authorization)
* [Bootstrap](http://jasonwatmore.com/posts/tag/bootstrap)
* [C#](http://jasonwatmore.com/posts/tag/c)
* [Chai](http://jasonwatmore.com/posts/tag/chai)
* [CKEditor](http://jasonwatmore.com/posts/tag/ckeditor)
* [CSS3](http://jasonwatmore.com/posts/tag/css3)
* [DDD](http://jasonwatmore.com/posts/tag/ddd)
* [Design Patterns](http://jasonwatmore.com/posts/tag/design-patterns)
* [Dynamic LINQ](http://jasonwatmore.com/posts/tag/dynamic-linq)
* [ELMAH](http://jasonwatmore.com/posts/tag/elmah)
* [ES6](http://jasonwatmore.com/posts/tag/es6)
* [Exchange](http://jasonwatmore.com/posts/tag/exchange)
* [Facebook](http://jasonwatmore.com/posts/tag/facebook)
* [Fluent NHibernate](http://jasonwatmore.com/posts/tag/fluent-nhibernate)
* [Google Analytics](http://jasonwatmore.com/posts/tag/google-analytics)
* [Google API](http://jasonwatmore.com/posts/tag/google-api)
* [Google Maps API](http://jasonwatmore.com/posts/tag/google-maps-api)
* [Google Plus](http://jasonwatmore.com/posts/tag/google-plus)
* [Heroku](http://jasonwatmore.com/posts/tag/heroku)
* [HTML5](http://jasonwatmore.com/posts/tag/html5)
* [HTTP](http://jasonwatmore.com/posts/tag/http)
* [IIS](http://jasonwatmore.com/posts/tag/iis)
* [Insecure Content](http://jasonwatmore.com/posts/tag/insecure-content)
* [Instagram API](http://jasonwatmore.com/posts/tag/instagram-api)
* [Ionic Framework](http://jasonwatmore.com/posts/tag/ionic-framework)
* [iOS](http://jasonwatmore.com/posts/tag/ios)
* [iPhone](http://jasonwatmore.com/posts/tag/iphone)
* [JavaScript](http://jasonwatmore.com/posts/tag/javascript)
* [jQuery](http://jasonwatmore.com/posts/tag/jquery)
* [LINQ](http://jasonwatmore.com/posts/tag/linq)
* [Login](http://jasonwatmore.com/posts/tag/login)
* [MEAN Stack](http://jasonwatmore.com/posts/tag/mean-stack)
* [Mocha](http://jasonwatmore.com/posts/tag/mocha)
* [Modal](http://jasonwatmore.com/posts/tag/modal)
* [MongoDB](http://jasonwatmore.com/posts/tag/mongodb)
* [Moq](http://jasonwatmore.com/posts/tag/moq)
* [MVC](http://jasonwatmore.com/posts/tag/mvc)
* [MVC5](http://jasonwatmore.com/posts/tag/mvc5)
* [ngMock](http://jasonwatmore.com/posts/tag/ngmock)
* [NHibernate](http://jasonwatmore.com/posts/tag/nhibernate)
* [Ninject](http://jasonwatmore.com/posts/tag/ninject)
* [NodeJS](http://jasonwatmore.com/posts/tag/nodejs)
* [Pagination](http://jasonwatmore.com/posts/tag/pagination)
* [Pinterest](http://jasonwatmore.com/posts/tag/pinterest)
* [React](http://jasonwatmore.com/posts/tag/react)
* [Redmine](http://jasonwatmore.com/posts/tag/redmine)
* [Redux](http://jasonwatmore.com/posts/tag/redux)
* [Registration](http://jasonwatmore.com/posts/tag/registration)
* [Repository](http://jasonwatmore.com/posts/tag/repository)
* [Security](http://jasonwatmore.com/posts/tag/security)
* [Sinon](http://jasonwatmore.com/posts/tag/sinon)
* [SinonJS](http://jasonwatmore.com/posts/tag/sinonjs)
* [TDD](http://jasonwatmore.com/posts/tag/tdd)
* [Twitter](http://jasonwatmore.com/posts/tag/twitter)
* [TypeScript](http://jasonwatmore.com/posts/tag/typescript)
* [Umbraco](http://jasonwatmore.com/posts/tag/umbraco)
* [Unit of Work](http://jasonwatmore.com/posts/tag/unit-of-work)
* [Unit Testing](http://jasonwatmore.com/posts/tag/unit-testing)
* [URL Rewrite](http://jasonwatmore.com/posts/tag/url-rewrite)
* [Validation](http://jasonwatmore.com/posts/tag/validation)
* [Windows Server 2008](http://jasonwatmore.com/posts/tag/windows-server-2008)

**Powered by [MEANie](http://jasonwatmore.com/post/2016/10/29/meanie-mean-stack-blogging-platform)**

© 2017 JasonWatmore.com

Sponsored by