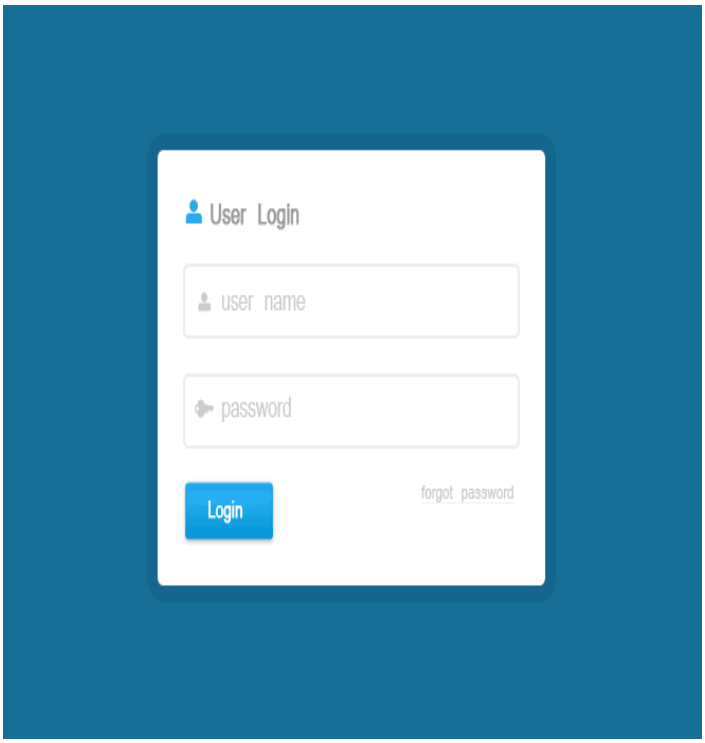


ANGULAR 4

Signup/Login Page design with output

ABSTRACT

A real Time experience with Login and Registration page design.

Sambit Kumar Padhy

Signup/Login Page Design

Running the Angular CLI Version of the Angular 4 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 min) 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`

```
Download the project source code from

https://gitlab.com/sambit567/Angular4-Signupform
```

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 `ng serve`from the command line in the project root folder.

5. Browse to http://localhost:4200 to test your application.

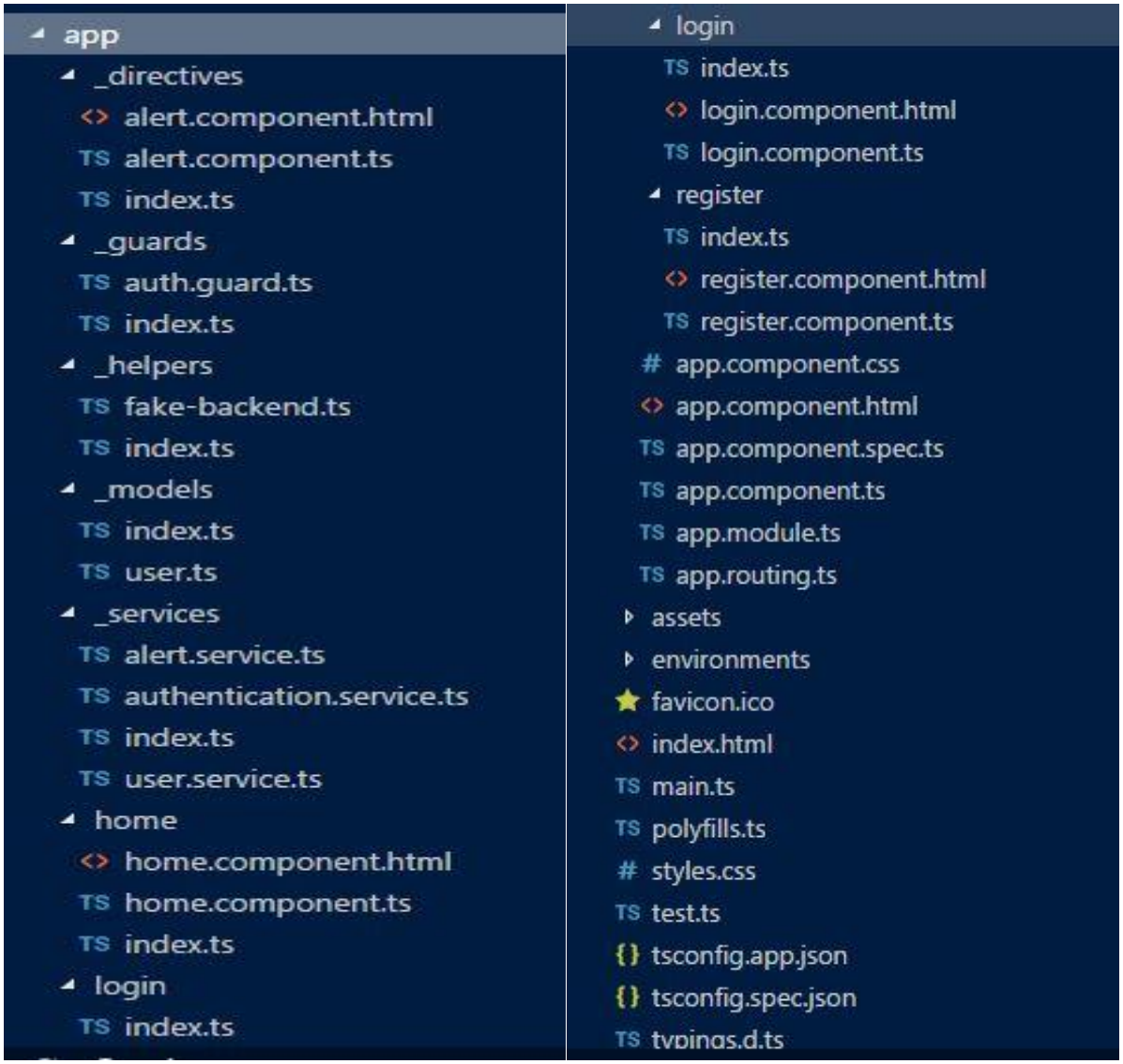
Project Structure

I used the Angular 4 quick start 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 quick start 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 4 style guide, with my own tweaks here and there.

Each feature has its 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:



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

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.

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

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.

```
alert.component.ts
import { Component, OnInit } from '@angular/core';

import { AlertService } from '../_services/index';

@Component({
    moduleId: module.id.toString(),
    selector: 'alert',
    templateUrl: 'alert.component.html'
})

export class AlertComponent {
    message: any;

    constructor(private alertService: AlertService) { }

    ngOnInit() {
        this.alertService.getMessage().subscribe(message => { this.message = message; });
    }
}
```

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.

```
TS auth.guard.ts ×
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;
    }
}
```

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 4 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.

TS fake-backend.ts ×

```
1   import { Http, BaseRequestOptions, Response, ResponseOptions, RequestMethod, XHRBackend, RequestOptions } from '@angular/h
2   import { MockBackend, MockConnection } from '@angular/http/testing';
3
4   export function fakeBackendFactory(backend: MockBackend, options: BaseRequestOptions, realBackend: XHRBackend) {
5       // array in local storage for registered users
6       let users: any[] = JSON.parse(localStorage.getItem('users')) || [];
7
8       // configure fake backend
9       backend.connections.subscribe((connection: MockConnection) => {
10          // wrap in timeout to simulate server api call
11          setTimeout(() => {
12
13              // authenticate
14              if (connection.request.url.endsWith('/api/authenticate') && connection.request.method === RequestMethod.Post)
15                  // get parameters from post request
16                  let params = JSON.parse(connection.request.getBody());
17
18                  // find if any user matches login credentials
19                  let filteredUsers = users.filter(user => {
20                      return user.username === params.username && user.password === params.password;
21                  });
22
23                  if (filteredUsers.length) {
24                      // if login details are valid return 200 OK with user details and fake jwt token
25                      let user = filteredUsers[0];
26                      connection.mockRespond(new Response(new ResponseOptions({
27                          status: 200,
28                          body: {
29                              id: user.id,
30                              username: user.username,
31                              firstName: user.firstName,
32                              lastName: user.lastName,
34                              token: 'fake-jwt-token'
35                          }
36              }));
37      } else {
38          // else return 400 bad request
39          connection.mockError(new Error('Username or password is incorrect'));
40      }
41
42      return;
43  }
44
45  // get users
46  if (connection.request.url.endsWith('/api/users') && connection.request.method === RequestMethod.Get) {
47      // check for fake auth token in header and return users if valid, this security is implemented server side in a re
48      if (connection.request.headers.get('Authorization') === 'Bearer fake-jwt-token') {
49          connection.mockRespond(new Response(new ResponseOptions({ status: 200, body: users })));
50      } else {
51          // return 401 not authorised if token is null or invalid
52          connection.mockRespond(new Response(new ResponseOptions({ status: 401 })));
53      }
54
55      return;
56  }
57
58  // get user by id
59  if (connection.request.url.match(/\/api\/users\/\d+$/) && connection.request.method === RequestMethod.Get) {
60      // check for fake auth token in header and return user if valid, this security is implemented server side in a rea
61      if (connection.request.headers.get('Authorization') === 'Bearer fake-jwt-token') {
62          // find user by id in users array
63          let urlParts = connection.request.url.split('/');
64          let id = parseInt(urlParts[urlParts.length - 1]);
65          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;
        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
        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);
```

```
129     let requestOptions = new RequestOptions({
130         method: connection.request.method,
131         headers: connection.request.headers,
132         body: connection.request.getBody(),
133         url: connection.request.url,
134         withCredentials: connection.request.withCredentials,
135         responseType: connection.request.responseType
136     });
137     realHttp.request(connection.request.url, requestOptions)
138         .subscribe((response: Response) => {
139             connection.mockRespond(response);
140         },
141         (error: any) => {
142             connection.mockError(error);
143         });
144
145   500);
146
147
148
149   new Http(backend, options);
150
151
152   fakeBackendProvider = {
153   fake backend in place of Http service for backend-less development
154   : Http,
155   ory: fakeBackendFactory,
156   MockBackend, BaseRequestOptions, XHRBackend]
```

User Model

Path: /app/_models/user.ts

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

```
TS user.ts
1   export class User {
2       id: number;
3       username: string;
4       password: string;
5       firstName: string;
6       lastName: string;
7       Email: string;
8   }
```

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.

```
TS alert.service.ts ×
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();
    }
}
```

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 SAM 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.

```
TS authentication.service.ts ×
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));
                }
            });
    }

    logout() {
        // remove user from local storage to log user out
        localStorage.removeItem('currentUser');
    }
}
```

User Service

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

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

```
TS user.service.ts ×
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.sam()).map((response: Response) => response.json());
    }

    getById(id: number) {
        return this.http.get('/api/users/' + id, this.sam()).map((response: Response) => response.json());
    }

    create(user: User) {
        return this.http.post('/api/users', user, this.sam()).map((response: Response) => response.json());
    }

    update(user: User) {
        return this.http.put('/api/users/' + user.id, user, this.sam()).map((response: Response) => response.json());
    }

    delete(id: number) {
        return this.http.delete('/api/users/' + id, this.sam()).map((response: Response) => response.json());
    }

    // private helper methods

    private sam() {
        // 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 });
        }
    }
}
```

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.

```
<> home.component.html ×
<div class="col-md-6 col-md-offset-3">
    <h1>Hi {{currentUser.firstName}}!</h1>
    <p>Wel Come to Angular 4 !!</p>
    <h3>All registered users:</h3>
    <ul>
        <li *ngFor="let user of users">
            {{user.username}} ({{user.firstName}} {{user.lastName}} {{user.Email}})
            - <a (click)="deleteUser(user.id)">Delete</a>
        </li>
    </ul>
    <p><a [routerLink]="['/login']">Logout</a></p>
</div>
```

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.

```
TS home.component.ts ×
import { Component, OnInit } from '@angular/core';

import { User } from '../_models/index';
import { UserService } from '../_services/index';

@Component({
    moduleId: module.id.toString(),
    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; });
    }
}
```

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.

```
login.component.html ×
<div class="card" style="width: 30rem;">
<div class="card-body">
    <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="
            <a [routerLink]="['/register']" class="btn btn-link">Register</a>
        </div>
    </form>
  </div>
</div>
```

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.

```
TS login.component.ts ×
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';

import { AlertService, AuthenticationService } from '../_services/index';

@Component({
    moduleId: module.id.toString(),
    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;
                });
    }
}

```

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.

register.component.html

```html
<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="Email">Mail ID</label>
                <input type="text" class="form-control" name="Email" [(ngModel)]="model.Email" #lastName="ngModel" required />
                <div *ngIf="f.submitted && !Email.valid" class="help-block">Email ID 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="
            <a [routerLink]="['/login']" class="btn btn-link">Cancel</a>
        </div>
    </form>
</div>
```

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.

register.component.ts

```typescript
import { AlertService, UserService } from '../_services/index';

@Component({
    moduleId: module.id.toString(),
    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 => {
                    this.alertService.success('Registration successful', true);
                    this.router.navigate(['/login']);
                },
                error => {
                    this.alertService.error(error);
                    this.loading = false;
                });
    }
}
```

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.

```
app.component.html
<!-- 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>

<!-- credits -->
<div class="text-center copyright">
    <p>
        <a href="https://gitlab.com/sambit567/Angular4-Signupform" target="_top">
            Angular 4 User Registration and Login Example </a>
    </p>
    <p>
            Copyright&copy; <a href="https://www.linkedin.com/in/sambit567" target="_top">Sambit Kumar Padhy</a>

    </p>
</div>
```

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.

```
app.component.ts
import { Component } from '@angular/core';

import '../assets/app.css';

@Component({
    moduleId: module.id.toString(),
    selector: 'app',
    templateUrl: 'app.component.html'
})

export class AppComponent { }
```

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 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".

```
TS app.module.ts ×
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 { }
```

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 can Activate property of the route.

```
TS app.routing.ts ×
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);
```

Main (Bootstrap) File

Path: /app/main.ts

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

```
TS main.ts ×
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

OUTPUT:

Login Page: