ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Angular Framework

Χριστόδουλος Φραγκουδάκης, Μάρκος Καραμπάτσης

Προετοιμασία και βασικές ενέργειες

- Εγκατάσταση του Angular CLI

```
npm install -g @angular/cli@latest
```

- Δημιουργία ενός νέου Angular Project

```
ng new angular-introduction --standalone --skip-tests
```

- Επεμβάσεις στο αρχείο `ts.config.json`

```
{
...
"compilerOptions": {
    ...
    "baseUrl": "./",
    "strict": false,
    ...
}
...
}
```

Εκκίνηση του Angular Project

```
❯ ng serve
Initial chunk files | Names         | Raw size
polyfills.js        | polyfills     | 83.60 kB |
main.js             | main          |  1.67 kB |
styles.css          | styles        | 95 bytes |

                    | Initial total | 85.36 kB

Application bundle generation complete. [1.241 seconds]

Watch mode enabled. Watching for file changes...
➜  Local:   http://localhost:4200/
➜  press h + enter to show help
```

- Η εφαρμογή είναι διαθέσιμη στη διεύθυνση `http://localhost:4200/`

Online repository στο GitHub

Δημιουργία online repository στο GitHub

- (`angular-introduction`) και αποστολή του κώδικα

```
git remote add origin git@github.com:christodoulos/angular-introduction.git
git push -u origin main
```

- Δημιουργία του repository `<username>.github.io` αν δεν υπάρχει ήδη.

- Προσθήκη δυνατότητας deployment στις σελίδες gh-pages του GitHub

```
ng add angular-cli-ghpages
```

- Προσθήκη του *deploy* script στο αρχείο `package.json`

```
{
...
"scripts": {
    ...
    "deploy": "ng deploy --base-href=https://<username>.github.io/angular-introduction/"
}
...
}
```

Σελίδες gh-pages του GitHub

Αποστολή της εφαρμογής στις σελίδες gh-pages του GitHub

```
npm run deploy
```

- Η εφαρμογή είναι διαθέσιμη στη διεύθυνση `https://<username>.github.io/angular-introduction/`
- Ενεργοποίηση του GitHub Pages για το repository `<username>.github.io/angular-introduction`
- Η εφαρμογή είναι διαθέσιμη στη διεύθυνση `https://<username>.github.io/angular-introduction/`

Εγκατάσταση του bootstrap

- Εγκατάσταση του bootstrap

```
npm install bootstrap
```

- Επεξεργασία του αρχείου `angular.json`

```
{
...
"styles": [
    "src/styles.css",
    "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
...
}
```

- **Επανεκκίνηση του Angular Project** μετά από κάθε αλλαγή στο αρχείο `angular.json` είναι απαραίτητο να εκκινηθεί ξανά το Angular Project (^C και ξανά `ng serve`)

Εγκατάσταση του prettier

- Τοπική εγκατάσταση του `prettier` και δημιουργία του αρχείου `.prettierrc`

```
npm install --save-dev prettier
```

```
{
  "overrides": [
    {
      "files": "*.html",
      "options": {
        "parser": "angular"
      }
    }
  ]
}
```

One Way Binding

Βήμα 1: Απλή δέσμευση χαρακτηριστικών (one way binding)

- Χρήση του placeholder `{{ <atribute_name > }}` για τη δεσμευση του χαρακτηριστικού `attribute_name` στο template του component.
- Αν το χαρακτηριστικό της κλάσης είναι αντικείμενο τότε χρησιμοποιούμε τη γνωστή σύνταξη `{{ <object_name>.<attribute_name> }}`.

New Component

Βήμα 2: Δημιουργία νέου component

- Δημιουργία ενός νέου component με την εντολή `ng generate component components/person-table`.
- Μεταφορά του πίνακα από το `app.component.html` στο template του νέου component.
- Μεταφορά του χαρακτηριστικού `person` από την κλάση `AppComponent` στην κλάση `PersonTableComponent`.
- Συμπερίληψη της κλάσης `PersonTableComponent` στον πίνακα `imports` στην αρχικοποίηση του decorator στο αρχείο `app.component.ts`.
- Χρήση του νέου component στο template του `app.component.html` με την ετικέτα `<app-person-table></app-person-table>`.

Component Input

Βήμα 3: Component Input

- Δημιουργία interface για τα δεδομένα τύπου `Person`

```
ng generate interface shared/interfaces/person
```

```
export interface Person {
  givenName: string;
  surName: string;
  age: number;
  email: string;
  address: string;
}
```

- Χρήση του interface `Person` ως τύπο του χαρακτηριστικού `person` στο component `PersonTableComponent`

Component Input

- Χρήση του decorator `@Input()` στο χαρακτηριστικό `person` τύπου `Person` ή `undefined` στο component `PersonTableComponent`
- Χρήση του `@if() {} @else {}` στο template του component `PersonTableComponent` για την υπό συνθήκη εμφάνιση των δεδομένων του χαρακτηριστικού `person`
- Η δέσμευση των χαρακτηριστικών της κλάσης `AppComponent` στο χαρακτηριστικό `person` του component `PersonTableComponent` γίνεται στο template του component `AppComponent`

```
<app-person-table [person]="person0"></app-person-table>
<!-- Χωρίς δέσμευση στο επόμενο -->
<app-person-table></app-person-table>
<app-person-table [person]="person1"></app-person-table>
```

@for

Βήμα 4: @for Template Directive

- Ορισμός χαρακτηριστικού `persons` τύπου `Person[]` στην κλάση `AppComponent` (πίνακας αντικειμένων τύπου `Person`)
- Χρήση του template directive `@for(obj of objects); track obj` για την εμφάνιση των δεδομένων του πίνακα `persons` με τη χρήση του component `PersonTableComponent`

```
@for (user of users; track user) {
<app-person-table [person]="user"></app-person-table>
}
```

Event binding

Βήμα 5:

- Δέσμευση μεθόδου της κλάσης (event handler) στο συμβάν `event` του template με χρήση του `(eventName)="onEventName($event)"`

```
<button (click)="onAddPerson()">Add Person</button>
```

- Χρήση του event `input` από ένα HTML input element για ανάγνωση της τιμής του στην κλάση και στη συνέχεια πέρασμα πίσω στο template με χρήση της απλής δέσμευση με το `{{ <atribute_name > }}`

```
<input type="text" (input)="onInput($event)" />
```

Routing

Βήμα 6: Routing

- Σκοπός μας είναι να κάνουμε επιλογές από το μενού στα αριστερά και τα component να εμφανίζονται στο χώρο δεξιά.
- Δημιουργία του Welcome component, αυτό που θα εμφανίζεται πρώτο όταν ξεκινήσει η εφαρμογή (χρησιμοποιεί κι ένα λογότυπο από το `/assets`):

```
ng g c welcome
```

Routing

Βήμα 6: Routing

- Στο αρχείο `app.routes.ts` ο πίνακας `routes` περιέχει αντικείμενα που είναι ο κατάλογος των path που εμφανίζονται στο μενού της εφαρμογής μαζί με το Angular component που αντιστοιχεί στο path.

```
import { Routes } from "@angular/router";
import { EventBindExampleComponent } from "src/app/components/event-bind-example/event-bind-example.component";
import { WelcomeComponent } from "./components/welcome/welcome.component";

export const routes: Routes = [
  { path: "event-bind-example", component: EventBindExampleComponent },
  { path: "welcome", component: WelcomeComponent },
  { path: "", redirectTo: "/welcome", pathMatch: "full" },
];
```

Routing

Βήμα 6: Routing

- Ήδη στο αρχείο `app.config.ts` ο κατάλογος των routes περνάει στο `provideRouter`:

```
import { ApplicationConfig } from "@angular/core";
import { provideRouter } from "@angular/router";

import { routes } from "./app.routes";

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes)],
};
```

Routing

Βήμα 6: Routing

- Το ακριβές σημείο στο template που θα εισάγονται τα component δηλώνεται με τη χρήση του tag `<router-outlet>` :

```
...
<span class="flex-grow-1 p-2 text-nowrap">
  <router-outlet></router-outlet>
</span>
...
```

Routing

Βήμα 6: Routing

Παράδειγμα ροής για μια επιλογή του χρήστη:

1. Ο χρήστης επιλέγει κάτι από το μενού που στην HTML το tag που αφορά την επιλογή συμπεριλαμβάνει την οδηγία `routerLink`, π.χ. στο `app.component.html` το tag `<span role="button" routerLink="event-bind-example">Event Bind Example</span>`.
2. Ο έλεγχος μεταβιβάζεται στο αρχείο `app.routes.ts` όπου γίνεται αναζήτηση στον πίνακα `routes` για την εύρεση του αντικειμένου που έχει τιμή στο χαρακτηριστικό `path` ίδια με την τιμή του `routerLink` στο tag από το βήμα 1.
3. Το URL αλλάζει σε αυτό που αντιστοιχεί στο path του αντικειμένου του βήματος 2.

Routing

Βήμα 6: Routing

4. Στο πλαίσιο του `<router-outlet></router-outlet>` εμφανίζεται το component από το χαρακτηριστικό του αντικειμένου του βήματος 2.

- Δημιουργία των `ComponentInputExampleComponent` και `ForDirectiveExampleComponent` και προσθήκη στο μενού της εφαρμογής:

    i. Ενημέρωση του αρχείου `app.routes.ts`

    ii. Ενημέρωση του html μενού με τις κατάλληλες οδηγίες `routerLink`

Fancy App Menu με το

Βήμα 7: Fancy App Menu με το list-group του Bootstrap

- Δημιουργία νέου interface `MenuItem` στο αρχείο `shared/interfaces/menu-item.ts`:

```
export interface MenuItem {
  text: string; // Κείμενο που εμφανίζεται στο μενού
  routerLink: string; // Το path που αντιστοιχεί στο component
}
```

- Δημιουργία του component `ListGroupMenuComponent` με την εντολή:

```
ng g c components/list-group-menu
```

Fancy App Menu με το

Βήμα 7: Fancy App Menu με το list-group του Bootstrap

- Το μενού της εφαρμογής μας είναι ένας πίνακας αντικειμένων `MenuEntry`:

```
menu: MenuEntry[] = [
  { text: 'Component Input Example', routerLink: 'component-input-example' },
  { text: '@for Directive Example', routerLink: 'for-directive-example' },
  { text: 'Event Bind Example', routerLink: 'event-bind-example' },
];
```

Simple Datatable

Βήμα 8: Simple Datatable

- Χρήση του https://cobbl.io/ για να παράξουμε ένα πίνακα με πολλά δεδομένα τύπου `EPerson` που ορίζουμε στο `/shared/interfaces/person.ts`:

```
export interface EPerson {
givenName: string;
surName: string;
age: string;
email: string;
address: string;
education: string;
}
```

```
export const ManyPerson: EPerson[] = [
  {
    given_name: 'Sarah',
    surName: 'Howard',
    age: '41',
    email: 's.m.howard@yahoo.com',
    education: 'Some college, no degree',
  },
  ...
```

Simple Datatable

Βήμα 8: Simple Datatable

- Δημιουργία του `SimpleDataTableComponent`: λαμβάνει δεδομένα τύπου `EPerson` και τα εμφανίζει σε έναν πίνακα με δυνατότητα ταξινόμησης ανά στήλη
- Δημιουργία του `SimpleDataTableExampleComponent`: χρησιμοποιεί το `SimpleDataTableComponent`
- Ενημέρωση του μενού της εφαρμογής μας στο `app.routes.ts`:

```
...
{
    path: 'simple-data-table-example',
    component: SimpleDatatableExampleComponent,
}
...
```

Simple Datatable

Βήμα 8: Simple Datatable

- `list-group-menu.component.ts`:

```
...
{
  text: 'Simple Data Table Example',
  routerLink: 'simple-data-table-example',
}
...
```

- Εγκατάταση του `lodash-es`:

```
npm i lodash-es
npm i --save-dev @types/lodash-es
```

Component Output

Βήμα 9: Component Output

- Δημιουργία του `ComponentOutputExampleComponent` και ενημέρωση του μενού της εφαρμογής μας (στο `app.routes.ts` και στο `list-group-menu.component.ts`).
- Ενημέρωση του `SimpleDataTableComponent` ώστε να περνάει σαν έξοδο τη γραμμή του πίνακα που επιλέγεται με διπλό κλικ.
  - Χρήση του decorator `@Output()` στο χαρακτηριστικό `personClicked` τύπου `EPerson` στο `SimpleDataTableComponent`.
  - Το output είναι ένα `EventEmitter<T>` που μεταφέρει δεδομένα του συγκεκριμένου τύπου `<T>`.

Angular Material

Βήμα 10: Angular Material

- Εγκατάσταση του Angular Material και του Angular CDK:

```
❯ ng add @angular/material
ℹ Using package manager: npm
✓ Found compatible package version: @angular/material@17.3.2.
✓ Package information loaded.

The package @angular/material@17.3.2 will be installed and executed.
Would you like to proceed? Yes
✓ Packages successfully installed.
? Choose a prebuilt theme name, or "custom" for a custom theme: Indigo/Pink [ Preview:
https://material.angular.io?theme=indigo-pink ]
? Set up global Angular Material typography styles? No
? Include the Angular animations module? Include and enable animations
```

Angular Material

```
UPDATE package.json (1396 bytes)
✓ Packages installed successfully.
UPDATE src/app/app.config.ts (338 bytes)
UPDATE angular.json (3652 bytes)
UPDATE src/index.html (516 bytes)
UPDATE src/styles.css (181 bytes)
```

- Επέμβαση στο `PersonTableComponent` για να χειρίζεται δεδομένα είτε `Person` είτε `EPerson`.
- Επέμβαση στο `ComponentOutputExampleComponent` και αντικατάσταση του `alert` με το `dialog` του Angular Material (https://t.ly/JLFka).

Template Driven Forms

Βήμα 11: Template Driven Forms

- Δημιουργία των `EpersonTemplateDrivenFormComponent` και `TemplateDrivenFormExampleComponent`.
- Ενημέρωση του μενού της εφαρμογής μας (στο `app.routes.ts` και στο `list-group-menu.component.ts`).
- Επέμβαση στο `SimpleDatatableComponent` για την περίπτωση του κενού πίνακα.
- Χρήση του Angular Forms Module.

Template Driven Forms

Η φόρμα ορίζεται στο template και μεταφέρει δεδομένα στο component κατά την υποβολή της. Συνήθως τότε, ένα EventEmitter μεταφέρει τα δεδομένα στο component γονέα.

- Χρήση του `FormsModule` στον πίνακα imports του component (εμπλουτίζει τα templates με επιπλέον HTML markup ώστε να δημιουργούνται objects από τις φόρμες).
- `<form #form="ngForm">...</form>` ορίζει πως η HTML φόρμα δημιουργεί ένα αντικείμενο που είναι διαχειρίσιμο στα πλαίσια του template με τη μεταβλητή (template variable) `form`.
- Το αντικείμενο `form` περνά σαν όρισμα στο `onSubmit(form)` όταν συμβεί το event `onSubmit` (ελέγχεται από το κουμπί Submit που μπορεί να πατηθεί μόνο όταν η φόρμα είναι ορθά συμπληρωμένη (valid)).

Template Driven Forms

- Δίνουμε στο name του input το όνομα του χαρακτηριστικού του αντικειμένου που παράγει η φόρμα και σχετίζεται (το χαρακτηριστικό) με το συγκεκριμένο input. Το συγκεκριμένο χαρακτηριστικό συμμετέχει στο αντικείμενο μόνο αν συμπεριλάβουμε την οδηγία `ngModel`.
- Με το `#givenName="ngModel"` δηλώνουμε τη μεταβλητή template με όνομα `givenName` που είναι αντικείμενο που μπορεί να εξεταστεί για την ορθότητά του με το `givenName.errors` και να χρησιμοποιηθεί για την υπο συνθήκη εμφάνιση επεξηγηματικού κειμένου για το ενδεχόμενο λάθος ορθότητας.

Reactive Forms

Βήμα 12: Reactive Forms

- Ξεκινάμε με τα αντίστοιχα βήματα όπως στο βήμα 11.

Η φόρμα ορίζεται στο component και συνδέεται με τα input του template. Ένας click handler μεταφέρει τα δεδομένα στο component και στη συνέχεια ένα EventEmitter μεταφέρει τα δεδομένα στο component γονέα.

- Χρήση του `ReactiveFormsModule` στον πίνακα imports του component (εμπλουτίζει τα templates με επιπλέον HTML markup ώστε να μπορούν να συσχετιστούν με τα χαρακτηριστικά του component).
- Χρήση των κλάσεων `FormGroup` και `FormControl` για τη δόμηση του αντικειμένου που παράγεται από τη φόρμα. Χρήση των `Validators`.

Reactive Forms

- Δέσμευση του χαρακτηριστικού `form` του component με χρήση του `<form [formGroup]="form">...</form>`.
- Σύνδεση του input με το `FormControl` με χρήση του `formControlName`.
- Άμεση πρόσβαση στο πεδίο της φόρμας με το `form.get('όνομα πεδίου')`
- Κατά το Submit το χαρακτηριστικό `form` έχει ήδη τιμή στο component.

HTTP Client Service

Βήμα 13: HTTP Client Service

- Για να μπορέσουμε να χρησιμοποιήσουμε τον HTTP Client είναι απαραίτητη η επέμβαση στο `app.config.ts` :

```
import { ApplicationConfig } from "@angular/core";
import { provideRouter } from "@angular/router";

import { routes } from "./app.routes";
import { provideAnimationsAsync } from "@angular/platform-browser/animations/async";

import { provideHttpClient, withInterceptorsFromDi } from "@angular/common/http";

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes), provideAnimationsAsync(), provideHttpClient(withInterceptorsFromDi())],
};
```

HTTP Client Service

- Δημιουργία του `JokesService` με την εντολή:

```
ng generate service shared/services/jokes
```

- Συνηθίζουμε να ορίζουμε με `const` το URL του API που θα χρησιμοποιήσουμε:

```
const DAD_JOKES_API_URL = "https://icanhazdadjoke.com/";
const JACK_NORRIS_JOKES_API_URL = "https://api.chucknorris.io/jokes/random";
```

- Το service είναι μια κλάση της Typescript με τον decorator `@Injectable({providedIn: 'root'})` που επιτρέπει την ενσωμάτωση του service σε όλα τα Angular components με χρήση του `inject`.

HTTP Client Service

- Ο `HttpClient` είναι ένα έτοιμο Angular service που παρέχει τη δυνατότητα αποστολής HTTP requests και λήψης HTTP responses. Τα service της εφαρμογής μας ενσωματώνουν άμεσα τον `HttpClient` με τη χρήση του `inject`.
- Δημιουργία του `HttpClientExampleComponent` για την επίδειξη της λειτουργίας του `HttpClient` μέσω του `JokesService`:

```
ng g c components/http-client-example
```

- Έλεγχος του τύπου των δεδομένων που επιστρέφουν οι κλήσεις των API με το `console.log`.

HTTP Client Service

```
import { Component, inject } from "@angular/core";
import { JokesService } from "src/app/shared/services/jokes.service";

@Component({
  selector: "app-http-client-example",
  standalone: true,
  imports: [],
  templateUrl: "./http-client-example.component.html",
  styleUrl: "./http-client-example.component.css",
})
export class HttpClientExampleComponent {
  jokesService = inject(JokesService);

  ngOnInit(): void {
    this.jokesService.getDadJoke().subscribe((data) => {
      console.log(data);
    });
    this.jokesService.getChuckNorrisJoke().subscribe((data) => {
      console.log(data);
    });
  }
}
```

HTTP Client Service

- Δημιουργία των Interfaces `DadJoke` και `ChuckNorrisJoke` στο αρχείο `shared/interfaces/jokes.ts` :

```
export interface DadJoke {
  joke: string;
}

export interface ChuckNorrisJoke {
  value: string;
}
```

HTTP Client Service

- Χρήση των interfaces για casting στον HttpClient:

```
getDadJoke() {
    return this.http.get<DadJoke>(DAD_JOKES_API_URL, {
      headers: {
        Accept: 'application/json',
      },
    });
  }
  getChuckNorrisJoke() {
    return this.http.get<ChuckNorrisJoke>(JACK_NORRIS_JOKES_API_URL, {
      headers: {
        Accept: 'application/json',
      },
    });
  }
```

- Ενημέρωση του μενού της εφαρμογής μας

User Registration Form and Service

Βήμα 14: User Registration Form and Service

- Από το σημείο αυτό και στο εξής είναι απαραίτητο να έχετε εγκαταστήσει τη γλώσσα Python και να χρησιμοποιείτε το Python-Flask backend από το repository angular-introduction-backend.
- Δημιουργία των enviroments με την εντολή:

```
ng generate environments
```

- Ενημέρωση των αρχείων `environment.development.ts` και `environment.ts`

User Registration Form and Service

- Δημιουργία του `User` interface στο αρχείο `shared/interfaces/mongo-backend.ts`:

```
export interface User {
  givenName: string;
  surName: string;
  email: string;
  password: string;
}
```

- Δημιουργία του `UserService` με την εντολή:

```
ng generate service shared/services/user
```

User Registration Form and Service

- Η μέθοδος `registerUser` αποστέλλει στο backend τα πλήρη δεδομένα που αφορούν στην εγγραφή ενός νέου χρήστη
- Η μέθοδος `check_duplicate_email` ρωτά το backend αν το `email` που λαμβάνει σαν όρισμα χρησιμοποιείται ήδη σε κάποια εγγραφή στη βάση.
- Δημιουργία του `UserRegistrationComponent` που υλοποιεί μια reactive form για τη διαδικασία του registration:

1. Χρησιμοποιεί το `UserService` με τη χρήση του `inject`,
2. Αρχικοποιεί το `registrationStatus`
3. Ορίζει τη φόρμα του registration με δύο πεδία για το password που θα πρέπει να λάβουν από το χρήστη ακριβώς το ίδιο περιεχόμενο

User Registration Form and Service

4. Δεύτερο όρισμα στον ορισμό της φόρμας μέσω του `FormGroup` είναι ο **συνολικός validator** της φόρμας, στην περίπτωσή μας η μέθοδος της κλάσης που εξετάζει αν τα δύο password input συμπίπτουν.
5. Στην περίπτωση εντοπισμού λάθους σε κάποιο input, ο Validator επιστρέφει ένα object με ένα κλειδί ενδεικτικό του λάθους
6. Το κλειδί αυτό μπορεί μετά να χρησιμοποιηθεί στο template για να εμφανιστεί κατάλληλο μήνυμα λάθους

User Registration Form and Service

- Στην υποβολή της φόρμας χρησιμοποιείται το `UserService` για να υποβάλλει στο backend τα δεδομένα της φόρμας. Η εγγραφή στην απάντηση του backend ξεχωρίζει τις περιπτώσεις της απάντησης του backend με τα callbacks στα χαρακτηριστικά `next` και `error` :
  - `next` : το callback που καλείται όταν στο backend στείλει HTTP response `20*` .
  - `error` : το callback που καλείται όταν το backend στείλει HTTP reponse `40*` ή `50*` .
  - Ανάλογα θέτουμε το `registrationStatus` για να έχουμε τον αντίστοιχο έλεγχο στο template.
- Χρήση του backend για τον έλεγχο ύπαρξης duplicate email στη βάση και χρήση της πληροφορίας κατά το event `blur` για να γίνει το πεδίο email invalid.

User Authentication

Βήμα 15: User Authentication

- Δημιουργούμε το `RestrictedContentExampleComponent` που θα λειτουργήσει σαν την πρώτη "προστατευμένη" περιοχή της εφαρμογής μας, με την έννοια ότι θα είναι προσβάσιμη μόνο από χρήστες που έχουν εγγραφεί μέσω του registration και έχουν κάνει επιτυχή έλεγχο πρόσβασης. Ενημερώνουμε το `app.routes.ts` και το μενού στο `list-group-menu.component.ts` κατά τα γνωστά και προς το παρόν όλοι οι χρήστες έχουν πρόσβαση.
- Δημιουργούμε το `UserLoginComponent` που θα παρουσιάζει τη φόρμα που ζητά τα user credentials (email και password) με σκοπό να τα υποβάλλει σε μέθοδο του `UserService` που θα επικοινωνήσει με το backend για τον έλεγχό τους. Η τιμή της φόρμας μετατρέπεται στο τύπο

User Authentication

```
export interface Credentials {
  email: string;
  password: string;
}
```

καθώς αυτού του τύπου δεδομένα αναμένει το endpoint `/user/login/` στο backend. Η μέθοδος της υποβολής με POST στο backend βρίσκεται στο `user.service.ts` (σε περίπτωση επιτυχημένου ελέγχου πρόσβασης δημιουργείται στο backend και επιστρέφεται ένα **JWT token** στο χαρακτηριστικό του response `access_token`):

```
loginUser(credentials: Credentials) {
    return this.http.post<{ access_token: string }>(
      `${API_URL}/login`,
      credentials,
    );
}
```

User Authentication

- Η κλήση στο service γίνεται στο `onSubmit()` του `UserLoginComponent` όπου στο subscription της απάντησης του backend λαμβάνουμε το αποτέλεσμα του επιτυχημένου ελέγχου πρόσβασης στο callback του χαρακτηριστικού `next`. Καθώς αυτό που θα λάβουμε είναι ένα κωδικοποιημένο JWT token υπάρχει ανάγκη να το αποκωδικοποιήσουμε (αφού πρώτα το αποθηκεύσουμε στο `localStorage`):

```
npm i jwt-decode
```

- Τα `signals` στο Angular Framework 17+ επιτρέπουν τη δημιουργία μεταβλητών που, μέσω των services, **όλα τα component** έχουν πρόσβαση στην τελευταία τιμή που έχει ανατεθεί σε αυτές τις μεταβλητές. Επιπρόσθετα η δέσμευση ενός signal από κάποιο component δίνει τη δυνατότητα στο component να **αντιδρά αυτόματα!** στις αλλαγές του signal.

User Authentication

- Θα χρησιμοποιήσουμε ένα signal για να υλοποιήσουμε την έννοια του ενεργού αυθεντικοποιημένου (authenticated) χρήστη που για την εφαρμογή μας θα είναι ένα αντικείμενο τύπου (`src/app/shared/interfaces/mongo-backend.ts`):

```
export interface LoggedInUser {
  fullname: string;
  email: string;
}
```

- Αν κανείς χρήστης δεν έχει κάνει login στην εφαρμογή, όπως όταν η εφαρμογή μόλις ξεκινά, τότε το signal θα περιέχει το `null` αλλιώς τον τύπο `LoggedInUser` (στο `src/app/shared/services/user.service.ts`):

```
user = signal<LoggedInUser | null>(null);
```

User Authentication

- Μπορούμε να παρακολουθούμε τις αλλαγές στα signals αν στο service χρησιμοποιήσουμε το `effect` που το callback που λαμβάνει εκτελείται αυτόματα σε κάθε αλλαγή των signals:

```
effect(() => {
  if (this.user()) {
    console.log("User logged in:", this.user().fullname);
  } else {
    console.log("No user logged in");
  }
});
```

User Authentication

- Είμαστε σε θέση να ολοκληρώσουμε το callback του subscription στην κλήση του backend για τον έλεγχο πρόσβασης:

```
this.userService.loginUser(credentials).subscribe({
  next: (response) => {
    const access_token = response.access_token;
    localStorage.setItem("access_token", access_token);
    const decodedTokenSubject = jwtDecode(access_token).sub as unknown as LoggedInUser;

    this.userService.user.set({
      fullname: decodedTokenSubject.fullname,
      email: decodedTokenSubject.email,
    });
    this.router.navigate(["restricted-content-example"]);
  },
  error: (error) => {
    console.error("Login error:", error);
    this.invalidLogin = true;
```

User Authentication

Μόλις το JWT token αποκωδικοποιηθεί τότε το payload ανατίθεται στη δομή του signal που αναπαριστά τον τρέχοντα αυθεντικοποιημένο χρήστη με χρήση της μεθόδου `.set(value:T)`. Τότε και μόνο τότε γίνεται αυτόματο redirection προς την προστατευμένη περιοχή της εφαρμογής μας.

- Επιπρόσθετα χρειάζεται η δημιουργία ενός route guard που θα επιβλέπει στο `app.routes.ts` πως μόνο οι εξουσιοδοτημένοι χρήστες έχουν πρόσβαση στο συγκεκριμένο component:

```
ng generate guard shared/guards/auth
```

User Authentication

Και στο `auth.guard.ts` :

```
export const authGuard: CanActivateFn = (route state)   => {
  const userService = inject(UserService);
  const router = inject(Router);

  if (userService.user()) {
    return true;
  }
  return router.navigate(['login']);
};
```

Ένας route guard, ή επιστρέφει true και άρα το component επιτρέπεται να εμφανιστεί στο `router-outlet` ή αλλιώς κάνει redirect στη φόρμα του login για να γίνει η διαδικασία του ελέγχου πρόσβασης.

User Authentication

- Με σκοπό να εμφανίζεται το ονοματεπώνυμο του loggedin χρήστη και για καλύτερη τακτοποίηση της εφαρμογής μας δημιουργούμε το `NavbarComponent` και χρησιμοποιούμε το signal του `UserService` για να χειριστούμε δυναμικά το template και να εμφανίζεται υπό συνθήκη το όνομα του χρήστη και επιπρόσθετα σε αυτή την περίπτωση, η προτροπή για αποσύνδεση (logout).
- Η λογική της αποσύνδεσης βρίσκεται στο `UserService`:

```
logoutUser() {
  this.user.set(null);
  localStorage.removeItem('access_token');
  this.router.navigate(['login']);
}
```

CRUD Example Scaffolding

Βήμα 16: CRUD Example Scaffolding

- Δημιουργία των component για το CRUD Example:

```
ng g c components/crud/crud-dashboard
ng g c components/crud/crud-navbar
ng g c components/crud/crud-create-example
ng g c components/crud/crud-read-example
ng g c components/crud/crud-update-example
ng g c components/crud/crud-delete-example
```

- Ενημέρωση του κεντρικού μενού της εφαρμογής για το path `crud-example` προς το `CrudDashboardComponent`
- Οι επιλογές του CRUD γίνονται από το μενού του `CrudNavbarComponent`

CRUD Example Scaffolding

Βήμα 17: CRUD Create Example

- Ασχολούμαστε με το `CrudCreateExampleComponent`. Χρησιμοποιούμε το `FormArray` για να δημιουργήσουμε αντικείμενα με δυναμικό αριθμό objects σε κάποιο χαρακτηριστικό τους. Συγκεκριμένα εδώ δίνουμε τη δυνατότητα εισαγωγής δυναμικού αριθμού από τηλεφωνικούς αριθμούς (κινητό, εργασία, σπίτι).
- Πρέπει να χρησιμοποιήσουμε ένα χαρακτηριστικό της κλάσης που να είναι reference στο `FormArray` για να μπορέσουμε αποτελεσματικά να έχουμε δυναμική εισαγωγή πεδίων από το template.

CRUD Example Scaffolding

```
...
phoneNumbers = this.form.get('phoneNumbers') as FormArray;

addPhoneNumber() {
  this.phoneNumbers.push(
    new FormGroup({
      number: new FormControl('', Validators.required),
      type: new FormControl('', Validators.required),
    }),
  );
}

removePhoneNumber(index: number) {
  this.phoneNumbers.removeAt(index);
}
...
```

CRUD Example Scaffolding

και στο template:

```
...
<div formArrayName="phoneNumbers">
  @for (phone of phoneNumbers.controls; let i = $index; track i) {
  <div [formGroupName]="i" class="d-flex gap-3 align-items-center">
    <mat-form-field>
      <mat-label>Phone Number</mat-label>
      <input matInput type="text" formControlName="number" />
      @if (phone.get("number").invalid && phone.get("number").touched) {
      <mat-error class="text-wrap"> Ο αριθμός του τηλεφώνου είναι απαραίτητος </mat-error>
      }
    </mat-form-field>
    <mat-form-field>
      <mat-label>Type</mat-label>
      <mat-select formControlName="type">
        <mat-option value="Mobile">Κινητό</mat-option>
        <mat-option value="Work">Εργασία</mat-option>
        <mat-option value="Home">Σπίτι</mat-option>
      </mat-select>
      @if (phone.get("type").invalid && phone.get("type").touched) {
      <mat-error class="text-wrap"> Πρέπει να διαλέξετε κάποιον τύπο τηλεφώνου </mat-error>
      }
    </mat-form-field>
    @if (phoneNumbers.controls.length > 1 && i !== 0) {
    <mat-icon role="button" (click)="removePhoneNumber(i)" class="cursor-pointer">delete</mat-icon>
    } @if (phoneNumbers.controls[i].valid) {
    <mat-icon role="button" (click)="addPhoneNumber()" class="cursor-pointer">add</mat-icon>
    }
  </div>
  }
```

CRUD Example Scaffolding

- Δημιουργία του `auth-interceptor.service.ts` :

```
ng generate service shared/services/auth-interceptor
```

Το interceptor είναι ένα Angular service που μπορεί να επεξεργαστεί τα HTTP requests ή τα HTTP responses πριν αυτά φτάσουν στον server ή μετά την απάντηση του server. Στην περίπτωσή μας το interceptor είναι υπεύθυνο για την προσθήκη του JWT token στο header των HTTP requests που απαιτούν αυθεντικοποίηση.

CRUD Example Scaffolding

```
import { HttpHandler, HttpInterceptor, HttpRequest } from "@angular/common/http";
import { Injectable } from "@angular/core";

@Injectable()
export class AuthInterceptorService implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler) {
    const authToken = localStorage.getItem("access_token");
    // console.log('authToken', authToken);
    if (!authToken) {
      return next.handle(req);
    }

    const authRequest = req.clone({
      headers: req.headers.set("Authorization", "Bearer " + authToken),
    });
    return next.handle(authRequest);
  }
}
```

CRUD Example Scaffolding

- Ενημέρωση του `app.config.ts`:

```
...
import {
  HTTP_INTERCEPTORS,
  provideHttpClient,
  withInterceptorsFromDi,
} from '@angular/common/http';
import { AuthInterceptorService } from './shared/services/auth-interceptor.service';

export const appConfig: ApplicationConfig = {
  providers: [
    ...,
    {
      provide: HTTP_INTERCEPTORS,
      useClass: AuthInterceptorService,
      multi: true,
    },
```