Лабораторна робота №6

Тема:Створення проекту Список справ з використанням Angular
Мета:Навчитись створювати проекти з використанням Angular.

1. Angular 7 Tutorial: Install or Update Angular 7 CLI and Create Application

To install or update Angular 7 CLI, type this command in the Terminal or Node Command Line.

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

Now, you have the latest version of Angular CLI.

```
ng --version

Angular CLI: 7.0.1
Node: 8.12.0
OS: darwin x64
Angular:
...

Package                      Version
------------------------------------------------------
@angular-devkit/architect    0.10.1
@angular-devkit/core         7.0.1
@angular-devkit/schematics   7.0.1
@schematics/angular          7.0.1
@schematics/update           0.10.1
rxjs                         6.3.3
typescript                   3.1.3
```

Next, create a new Angular 7 Web Application using this Angular CLI command.

```
ng new angular7-crud
```

If you get the question like below, choose `Yes` and `SCSS` (or whatever you like to choose).

```
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? SCSS
```

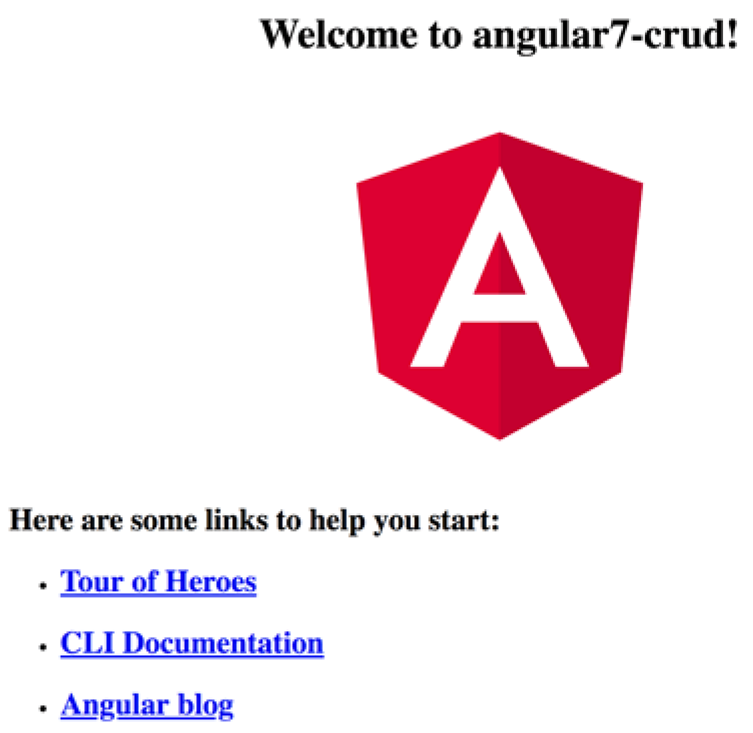
Next, go to the newly created Angular 7 project folder.

```
cd angular7-crud
```

Type this command to run the Angular 7 application using this command.

```
ng serve
```

Open your browser then go to this address `localhost:4200`, you should see this Angular 7 page.



Welcome to angular7-crud!

Here are some links to help you start:

- Tour of Heroes
- CLI Documentation
- Angular blog

2. Angular 7 Tutorial: Create Routes for Navigation between Angular Pages/Component

The Angular 7 routes already added when we create new Angular 7 application in the previous step. Before configure the routes, type this command to create a new Angular 7 components.

```
ng g component products
ng g component product-detail
ng g component product-add
ng g component product-edit
```

Open `src/app/app.module.ts` then you will see those components imported and declared in `@NgModule` declarations. Next, open and edit `src/app/app-routing.module.ts` then add this imports.

```
import { ProductsComponent } from './products/products.component';
import { ProductDetailComponent } from './product-detail/product-detail.component';
import { ProductAddComponent } from './product-add/product-add.component';
import { ProductEditComponent } from './product-edit/product-edit.component';
```

Add these arrays to the existing routes constant.

```
const routes: Routes = [
  {
    path: 'products',
    component: ProductsComponent,
    data: { title: 'List of Products' }
  },
  {
    path: 'product-details/:id',
    component: ProductDetailComponent,
    data: { title: 'Product Details' }
  },
  {
    path: 'product-add',
    component: ProductAddComponent,
    data: { title: 'Add Product' }
  },
  {
    path: 'product-edit/:id',
    component: ProductEditComponent,
    data: { title: 'Edit Product' }
  },
  { path: '',
    redirectTo: '/products',
    pathMatch: 'full'
  }
];
```

Open and edit `src/app/app.component.html` and you will see existing router outlet. Next, modify this HTML page to fit the CRUD page.

```
<div style="text-align:center">
  <img width="150" alt="Angular Logo" src="data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAyNTAgMjUwIj4KICAgIDxwYXRoIGZpbGw9IiNERDAwMzEiIGQ9Ik0xMjUgMzBMMzEuOSA2My4ybDE0LjIgMTIzLjFMMTI1IDIzMGw3OC45LTQzLjcgMTQuMi0xMjMuMXoiIC8+CiAgICA8cGF0aCBmaWxsPSIjQzMwMDJGIiBkPSJNMTI1IDMwdjIyLjItLjFWMjMwbDc4LjktNDMuNyAxNC4yLTEyMy4xTDEyNSAzMHoiIC8+CiAgICA8cGF0aCAgZmlsbD0iI0ZGRkZGRiIgZD0iTTEyNSA1Mi4xTDY2LjggMTgyLjZoMjEuN2wxMS43LTI5LjJoNDkuNGwxMS43IDI5LjJIMTgzTDEyNSA1Mi4xem0xNyA4My4zaC0zNGwxNy00MC45IDE3IDQwLjl6IiAvPgogIDwvc3ZnPg==">
</div>

<div class="container">
  <router-outlet></router-outlet>
</div>
```

Open and edit `src/app/app.component.scss` then replace all SASS codes with this.

```
.container {
  padding: 20px;
}
```

3. Angular 7 Tutorial: Create Service for Accessing RESTful API

Before creating a service for RESTful API access, first, we have to install or register `HttpClientModule`. Open and edit `src/app/app.module.ts` then add this import.

```
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
```

Add it to `@NgModule` imports after `BrowserModule`.

```
imports: [
  BrowserModule,
  FormsModule,
  HttpClientModule,
  AppRoutingModule
],
```

We will use type specifier to get a typed result object. For that, create a new Typescript file `src/app/product.ts` then add this lines of Typescript codes.

```
export class Product {
  id: number;
  prod_name: string;
  prod_desc: string;
  prod_price: number;
  updated_at: Date;
}
```

Next, generate an Angular 7 service by typing this command.

```
ng g service api
```

Next, open and edit `src/app/api.service.ts` then add this imports.

```
import { Observable, of, throwError } from 'rxjs';
import { HttpClient, HttpHeaders, HttpErrorResponse } from '@angular/common/http';
import { catchError, tap, map } from 'rxjs/operators';
import { Product } from './product';
```

Add these constants before the `@Injectable`.

```
const httpOptions = {
  headers: new HttpHeaders({'Content-Type': 'application/json'})
};
const apiUrl = "/api/v1/products";
```

Inject `HttpClient` module to the constructor.

```
constructor(private http: HttpClient) { }
```

Add the error handler function.

```
private handleError<T> (operation = 'operation', result?: T) {
  return (error: any): Observable<T> => {

    // TODO: send the error to remote logging infrastructure
    console.error(error); // log to console instead

    // Let the app keep running by returning an empty result.
    return of(result as T);
  };
}
```

Add all CRUD (create, read, update, delete) functions of products data.

```
getProducts (): Observable<Product[]> {
  return this.http.get<Product[]>(apiUrl)
    .pipe(
      tap(heroes => console.log('fetched products')),
      catchError(this.handleError('getProducts', []))
    );
}

getProduct(id: number): Observable<Product> {
  const url = `${apiUrl}/${id}`;
  return this.http.get<Product>(url).pipe(
    tap(_ => console.log(`fetched product id=${id}`)),
    catchError(this.handleError<Product>(`getProduct id=${id}`))
  );
}

addProduct (product): Observable<Product> {
  return this.http.post<Product>(apiUrl, product, httpOptions).pipe(
    tap((product: Product) => console.log(`added product w/ id=${product.id}`)),
    catchError(this.handleError<Product>('addProduct'))
  );
}
```

```
updateProduct (id, product): Observable<any> {
  const url = `${apiUrl}/${id}`;
  return this.http.put(url, product, httpOptions).pipe(
    tap(_ => console.log(`updated product id=${id}`)),
    catchError(this.handleError<any>('updateProduct'))
  );
}

deleteProduct (id): Observable<Product> {
  const url = `${apiUrl}/${id}`;

  return this.http.delete<Product>(url, httpOptions).pipe(
    tap(_ => console.log(`deleted product id=${id}`)),
    catchError(this.handleError<Product>('deleteProduct'))
  );
}
```

4. Angular 7 Tutorial: Display List of Products using Angular Material

We will display the list of products that get via API Service. For that, open and edit `src/app/products/products.component.ts` then add this imports.

```
import { ApiService } from '../api.service';
```

Next, inject the API Service to the constructor.

```
constructor(private api: ApiService) { }
```

Next, for user interface (UI) we will use Angular 6 Material and CDK. There's a CLI for generating a Material component like Table as a component, but we will create or add the Table component from scratch to existing component. Type this command to install Angular Material.

```
ng add @angular/material
```

If there are questions like below, just use the default answer.

```
? Enter a prebuilt theme name, or "custom" for a custom theme: purple-green
? Set up HammerJS for gesture recognition? Yes
? Set up browser animations for Angular Material? Yes
```

We will register all required Angular Material components or modules to `src/app/app.module.ts`. Open and edit that file then add this imports.

```
import {
  MatInputModule,
  MatPaginatorModule,
  MatProgressSpinnerModule,
  MatSortModule,
  MatTableModule,
  MatIconModule,
  MatButtonModule,
  MatCardModule,
  MatFormFieldModule } from "@angular/material";
```

Also, modify `FormsModule` import to add `ReactiveFormsModule`.

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
```

Register the above modules to `@NgModule` imports.

```
imports: [
  BrowserModule,
  FormsModule,
  HttpClientModule,
  AppRoutingModule,
  ReactiveFormsModule,
  BrowserAnimationsModule,
  MatInputModule,
  MatTableModule,
  MatPaginatorModule,
  MatSortModule,
  MatProgressSpinnerModule,
  MatIconModule,
  MatButtonModule,
  MatCardModule,
  MatFormFieldModule
],
```

Next, back to `src/app/products/products.component.ts` then add this imports.

```
import { Product } from '../product';
```

Declare the variables of Angular Material Table Data Source before the constructor.

```
displayedColumns: string[] = ['prod_name', 'prod_price'];
data: Product[] = [];
isLoadingResults = true;
```

Modify the `ngOnInit` function to get list of products immediately.

```
ngOnInit() {
  this.api.getProducts()
    .subscribe(res => {
      this.data = res;
      console.log(this.data);
      this.isLoadingResults = false;
    }, err => {
      console.log(err);
      this.isLoadingResults = false;
    });
}
```

Next, open and edit `src/app/products/products.component.html` then replace all HTML tags with this Angular Material tags.

```
<div class="example-container mat-elevation-z8">
  <div class="example-loading-shade"
       *ngIf="isLoadingResults">
    <mat-spinner *ngIf="isLoadingResults"></mat-spinner>
  </div>
  <div class="button-row">
     <a mat-flat-button color="primary" [routerLink]="['/product-add']"><mat-icon>add</mat-
icon></a>
  </div>
  <div class="mat-elevation-z8">
    <table mat-table [dataSource]="data" class="example-table"
           matSort matSortActive="prod_name" matSortDisableClear matSortDirection="asc">

      <!-- Product Name Column -->
      <ng-container matColumnDef="prod_name">
        <th mat-header-cell *matHeaderCellDef>Product Name</th>
        <td mat-cell *matCellDef="let row">{{row.prod_name}}</td>
      </ng-container>

      <!-- Product Price Column -->
      <ng-container matColumnDef="prod_price">
```

```
        <th mat-header-cell *matHeaderCellDef>Product Price</th>
        <td mat-cell *matCellDef="let row">$ {{row.prod_price}}</td>
      </ng-container>

      <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
      <tr mat-row *matRowDef="let row; columns: displayedColumns;" [routerLink]="['/product-
details/', row.id]"></tr>
    </table>
  </div>
</div>
```

Finally, to make a little UI adjustment, open and edit `src/app/products/products.component.css` then add this CSS codes.

```
/* Structure */
.example-container {
  position: relative;
  padding: 5px;
}

.example-table-container {
  position: relative;
  max-height: 400px;
  overflow: auto;
}

table {
  width: 100%;
}

.example-loading-shade {
  position: absolute;
  top: 0;
  left: 0;
  bottom: 56px;
  right: 0;
  background: rgba(0, 0, 0, 0.15);
  z-index: 1;
  display: flex;
  align-items: center;
  justify-content: center;
}

.example-rate-limit-reached {
  color: #980000;
  max-width: 360px;
  text-align: center;
}

/* Column Widths */
.mat-column-number,
.mat-column-state {
  max-width: 64px;
}
```

```
.mat-column-created {
  max-width: 124px;
}

.mat-flat-button {
  margin: 5px;
}
```

5. Angular 7 Tutorial: Show and Delete Product Details using Angular Material

To show product details after click or tap on the one of a row inside the Angular Material table, open and edit `src/app/product-detail/product-detail.component.ts` then add this imports.

```
import { ActivatedRoute, Router } from '@angular/router';
import { ApiService } from '../api.service';
import { Product } from '../product';
```

Inject above modules to the constructor.

```
constructor(private route: ActivatedRoute, private api: ApiService, private router: Router) { }
```

Declare the variables before the constructor for hold product data that get from the API.

```
product: Product = { _id: '', prod_name: '', prod_desc: '', prod_price: null, updated_at: null };
isLoadingResults = true;
```

Add a function for getting Product data from the API.

```
getProductDetails(id) {
  this.api.getProduct(id)
    .subscribe(data => {
      this.product = data;
      console.log(this.product);
      this.isLoadingResults = false;
    });
}
```

Call that function when the component is initiated.

```
ngOnInit() {
  this.getProductDetails(this.route.snapshot.params['id']);
}
```

Add this function for delete product.

```
deleteProduct(id) {
  this.isLoadingResults = true;
  this.api.deleteProduct(id)
    .subscribe(res => {
        this.isLoadingResults = false;
        this.router.navigate(['/products']);
      }, (err) => {
        console.log(err);
        this.isLoadingResults = false;
      }
    );
}
```

For the view, open and edit `src/app/product-detail/product-detail.component.html` then replace all HTML tags with this.

```
<div class="example-container mat-elevation-z8">
  <div class="example-loading-shade"
       *ngIf="isLoadingResults">
    <mat-spinner *ngIf="isLoadingResults"></mat-spinner>
  </div>
  <div class="button-row">
    <a mat-flat-button color="primary" [routerLink]="['/products']"><mat-icon>list</mat-icon></a>
  </div>
  <mat-card class="example-card">
    <mat-card-header>
      <mat-card-title><h2>{{product.prod_name}}</h2></mat-card-title>
      <mat-card-subtitle>{{product.prod_desc}}</mat-card-subtitle>
    </mat-card-header>
    <mat-card-content>
      <dl>
        <dt>Product Price:</dt>
        <dd>{{product.prod_price}}</dd>
        <dt>Updated At:</dt>
        <dd>{{product.updated_at | date}}</dd>
      </dl>
    </mat-card-content>
    <mat-card-actions>
      <a mat-flat-button color="primary" [routerLink]="['/product-edit', product._id]"><mat-icon>edit</mat-icon></a>
      <a mat-flat-button color="warn" (click)="deleteProduct(product._id)"><mat-icon>delete</mat-icon></a>
    </mat-card-actions>
  </mat-card>
</div>
```

Finally, open and edit `src/app/product-detail/product-detail.component.css` then add this lines of CSS codes.

```
/* Structure */
.example-container {
  position: relative;
  padding: 5px;
}

.example-loading-shade {
  position: absolute;
  top: 0;
  left: 0;
  bottom: 56px;
  right: 0;
  background: rgba(0, 0, 0, 0.15);
  z-index: 1;
  display: flex;
  align-items: center;
  justify-content: center;
}

.mat-flat-button {
  margin: 5px;
}
```

6. Angular 7 Tutorial: Add a Product using Angular Material

To create a form for adding a Product, open and edit `src/app/product-add/product-add.component.ts` then add this imports.

```
import { Router } from '@angular/router';
import { ApiService } from '../api.service';
import { FormControl, FormGroupDirective, FormBuilder, FormGroup, NgForm, Validators }
from '@angular/forms';
```

Inject above modules to the constructor.

```
constructor(private router: Router, private api: ApiService, private formBuilder: FormBuilder) {
}
```

Declare variables for the Form Group and all of the required fields inside the form before the constructor.

```
productForm: FormGroup;
prod_name:string='';
prod_desc:string='';
prod_price:number=null;
updated_at:Date=null;
isLoadingResults = false;
```

Add initial validation for each field.

```
ngOnInit() {
  this.productForm = this.formBuilder.group({
    'prod_name' : [null, Validators.required],
    'prod_desc' : [null, Validators.required],
    'prod_price' : [null, Validators.required],
    'updated_at' : [null, Validators.required]
  });
}
```

Create a function for submitting or POST product form.

```
onFormSubmit(form:NgForm) {
  this.isLoadingResults = true;
  this.api.addProduct(form)
    .subscribe(res => {
        let id = res['_id'];
        this.isLoadingResults = false;
        this.router.navigate(['/product-details', id]);
      }, (err) => {
        console.log(err);
        this.isLoadingResults = false;
      });
}
```

Next, open and edit `src/app/product-add/product-add.component.html` then replace all HTML tags with this.

```
<div class="example-container mat-elevation-z8">
  <div class="example-loading-shade"
       *ngIf="isLoadingResults">
    <mat-spinner *ngIf="isLoadingResults"></mat-spinner>
  </div>
  <div class="button-row">
      <a mat-flat-button color="primary" [routerLink]="['/products']"><mat-icon>list</mat-
icon></a>
  </div>
  <mat-card class="example-card">
    <form [formGroup]="productForm" (ngSubmit)="onFormSubmit(productForm.value)">
```

```
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Product Name" formControlName="prod_name"
               [errorStateMatcher]="matcher">
        <mat-error>
                                                  <span *ngIf="!productForm.get('prod_name').valid &&
productForm.get('prod_name').touched">Please enter Product Name</span>
        </mat-error>
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Product Desc" formControlName="prod_desc"
               [errorStateMatcher]="matcher">
        <mat-error>
                                                  <span *ngIf="!productForm.get('prod_desc').valid &&
productForm.get('prod_desc').touched">Please enter Product Description</span>
        </mat-error>
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Product Price" formControlName="prod_price"
               [errorStateMatcher]="matcher">
        <mat-error>
                                                  <span *ngIf="!productForm.get('prod_price').valid &&
productForm.get('prod_price').touched">Please enter Product Price</span>
        </mat-error>
      </mat-form-field>
      <div class="button-row">
                         <button type="submit" [disabled]="!productForm.valid" mat-flat-button
color="primary"><mat-icon>save</mat-icon></button>
      </div>
    </form>
  </mat-card>
</div>
```

Finally, open and edit `src/app/product-add/product-add.component.css` then add this CSS codes.

```
/* Structure */
.example-container {
  position: relative;
  padding: 5px;
}

.example-form {
  min-width: 150px;
  max-width: 500px;
  width: 100%;
}

.example-full-width {
  width: 100%;
}

.example-full-width:nth-last-child() {
  margin-bottom: 10px;
}
```

```
.button-row {
  margin: 10px 0;
}

.mat-flat-button {
  margin: 5px;
}
```

7. Angular 7 Tutorial: Edit a Book using Angular Material

We have put an edit button inside the Product Detail component for call Edit page. Now, open and edit `src/app/product-edit/product-edit.component.ts` then add this imports.

```
import { Router, ActivatedRoute } from '@angular/router';
import { ApiService } from '../api.service';
import { FormControl, FormGroupDirective, FormBuilder, FormGroup, NgForm, Validators }
from '@angular/forms';
```

Inject above modules to the constructor.

```
constructor(private router: Router, private route: ActivatedRoute, private api: ApiService, private
formBuilder: FormBuilder) { }
```

Declare the Form Group variable and all of the required variables for the product form before the constructor.

```
productForm: FormGroup;
_id:string='';
prod_name:string='';
prod_desc:string='';
prod_price:number=null;
isLoadingResults = false;
```

Next, add validation for all fields when the component is initiated.

```
ngOnInit() {
  this.getProduct(this.route.snapshot.params['id']);
  this.productForm = this.formBuilder.group({
    'prod_name' : [null, Validators.required],
    'prod_desc' : [null, Validators.required],
    'prod_price' : [null, Validators.required]
  });
}
```

Create a function for getting product data that filled to each form fields.

```
getProduct(id) {
  this.api.getProduct(id).subscribe(data => {
    this._id = data._id;
    this.productForm.setValue({
      prod_name: data.prod_name,
      prod_desc: data.prod_desc,
      prod_price: data.prod_price
    });
  });
}
```

Create a function to update the product changes.

```
onFormSubmit(form:NgForm) {
  this.isLoadingResults = true;
  this.api.updateProduct(this._id, form)
    .subscribe(res => {
        let id = res['_id'];
        this.isLoadingResults = false;
        this.router.navigate(['/product-details', id]);
      }, (err) => {
        console.log(err);
        this.isLoadingResults = false;
      }
    );
}
```

Add a function for handling show product details button.

```
productDetails() {
  this.router.navigate(['/product-details', this._id]);
}
```

Next, open and edit `src/app/product-edit/product-edit.component.html` then replace all HTML tags with this.

```
<div class="example-container mat-elevation-z8">
  <div class="example-loading-shade"
       *ngIf="isLoadingResults">
    <mat-spinner *ngIf="isLoadingResults"></mat-spinner>
  </div>
  <div class="button-row">
    <a mat-flat-button color="primary" (click)="productDetails()"><mat-icon>info</mat-icon></a>
  </div>
  <mat-card class="example-card">
    <form [formGroup]="productForm" (ngSubmit)="onFormSubmit(productForm.value)">
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Product Name" formControlName="prod_name"
               [errorStateMatcher]="matcher">
        <mat-error>
          <span *ngIf="!productForm.get('prod_name').valid && productForm.get('prod_name').touched">Please enter Product Name</span>
        </mat-error>
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Product Desc" formControlName="prod_desc"
               [errorStateMatcher]="matcher">
        <mat-error>
          <span *ngIf="!productForm.get('prod_desc').valid && productForm.get('prod_desc').touched">Please enter Product Description</span>
        </mat-error>
      </mat-form-field>
      <mat-form-field class="example-full-width">
        <input matInput placeholder="Product Price" formControlName="prod_price"
               [errorStateMatcher]="matcher">
        <mat-error>
          <span *ngIf="!productForm.get('prod_price').valid && productForm.get('prod_price').touched">Please enter Product Price</span>
        </mat-error>
      </mat-form-field>
      <div class="button-row">
```

```
            <button type="submit" [disabled]="!productForm.valid" mat-flat-button color="primary"><mat-icon>save</mat-icon></button>
      </div>
    </form>
  </mat-card>
</div>
```

Finally, open and edit `src/app/product-edit/product-edit.component.css` then add this lines of CSS codes.

```
/* Structure */
.example-container {
  position: relative;
  padding: 5px;
}

.example-form {
  min-width: 150px;
  max-width: 500px;
  width: 100%;
}

.example-full-width {
  width: 100%;
}

.example-full-width:nth-last-child() {
  margin-bottom: 10px;
}

.button-row {
  margin: 10px 0;
}

.mat-flat-button {
  margin: 5px;
}
```

8. Angular 7 Tutorial: Run and Test the Angular 7 CRUD Web Application

Now, it's a time for testing the Angular 7 CRUD Web Application. First, we have to run MongoDB server in another Terminal tab.

```
mongod
```

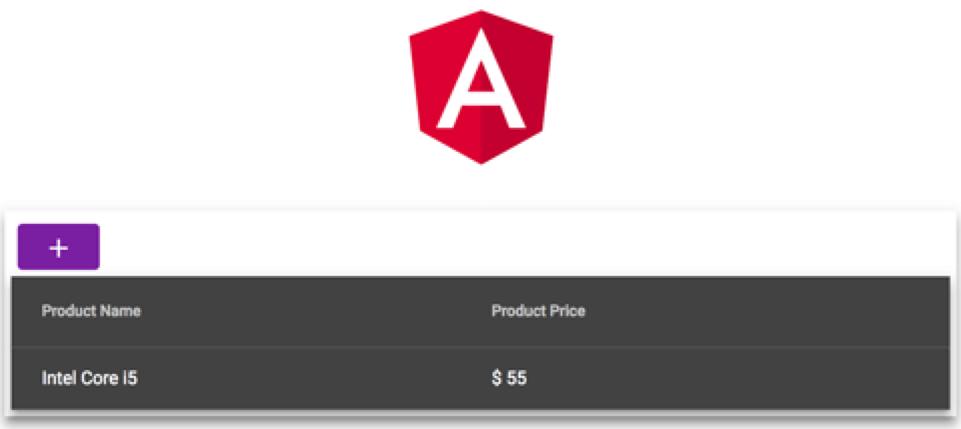
Open the other Terminal tab again then run the cloned Express.js API.

```
npm start
```

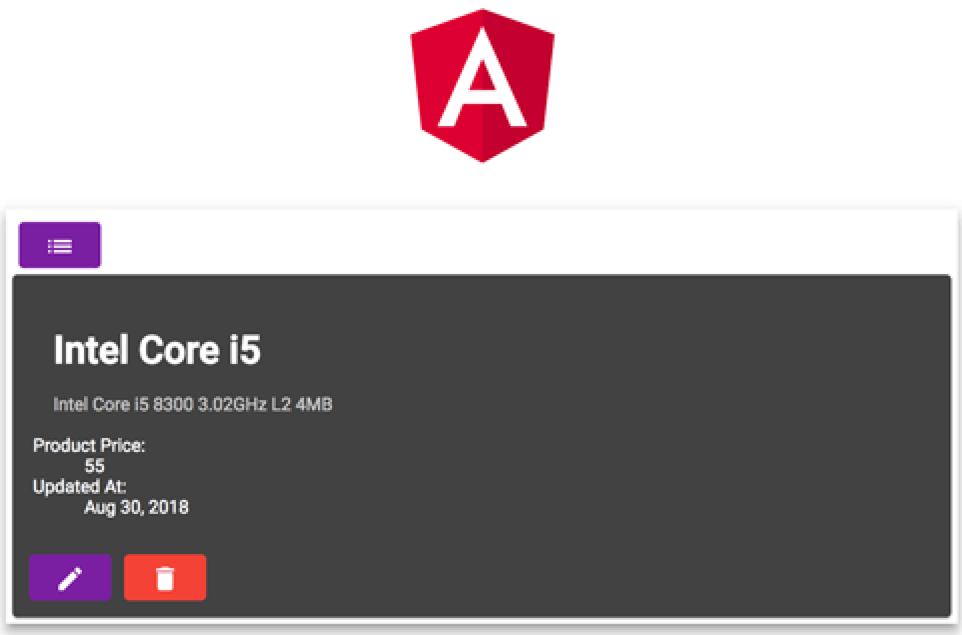
Back to the current Terminal tab, then run the Angular 7 Web Application.

```
ng serve
```

In the browser go to this URL `localhost:4200` and here the whole application looks like.



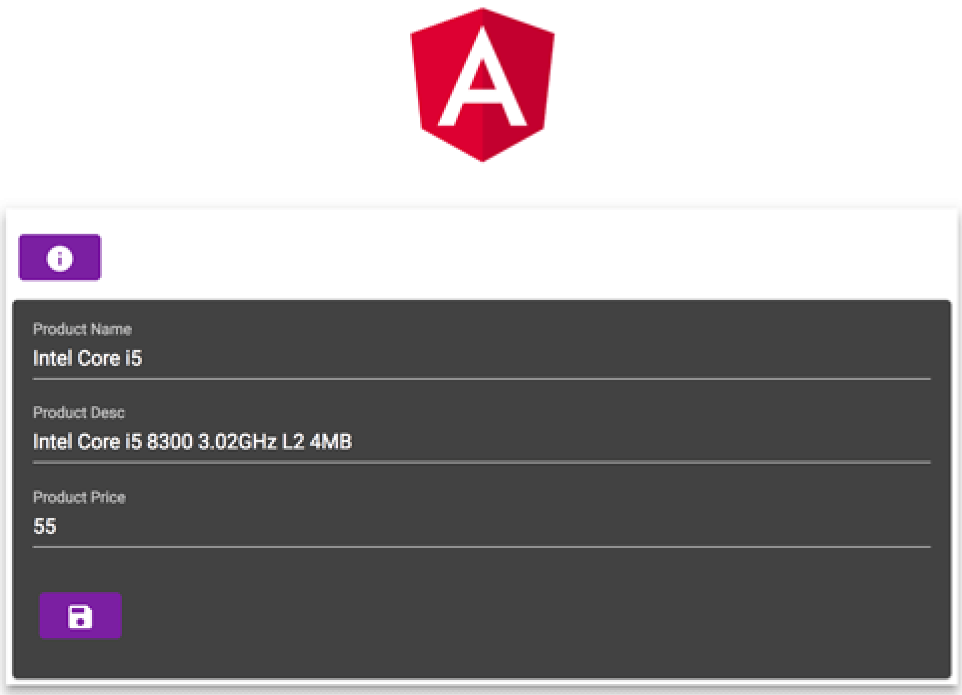
Product Name	Product Price
Intel Core i5	$ 55



Intel Core i5

Intel Core i5 8300 3.02GHz L2 4MB

Product Price:
55
Updated At:
Aug 30, 2018



Product Name
Intel Core i5

Product Desc
Intel Core i5 8300 3.02GHz L2 4MB

Product Price
55

That it's, we have finished the Angular 7 Tutorial: Building CRUD Web Application. If you can't follow the steps of the tutorial, you can compare it with the working source code from our GitHub.

Завдання для виконання

1. Розгорнути проект та підключити модулі.
2. Створити компоненти та сервіси, як показано в інструкції