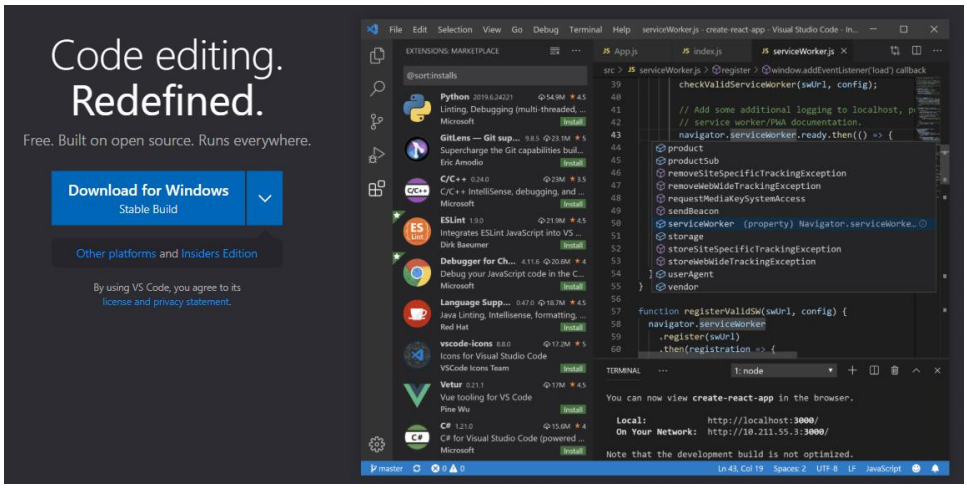
Uvod:

Ovaj tutorijal služi za objašnjavanje osnovnih koncepata Angular-a, front end frejmvorka baziranog na JavaScript-u. Svaki korak će biti detaljno objašnjen, a sama aplikacija će se zasnivati na primeru elektronskog super marketa, i to iz ugla osobe koja taj isti market vodi. Aplikacija će omogućiti pregled svih proizvoda, dodavanje novih proizvoda, brisanje već postojećih, i mnoge druge opcije koje će se nadograditi na inicijalnu postavku. Ceo kod aplikacije se može naći na sledem linku: https://github.com/Stevan-Radovanovic/AngularApp-Elab-E-Shop

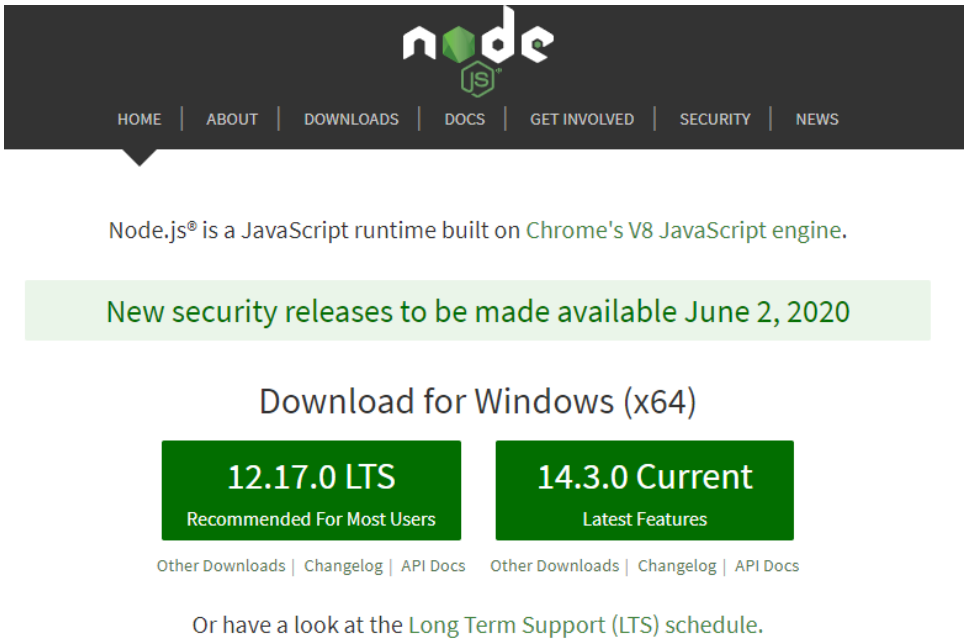
Korak 1: Instalacija Visual Studio Code-a

Okruženje koje ćemo koristiti je Visual Studio Code. Instalacija je laka, ne zauzima mnogo, i može se naći na oficijalnom sajtu.



Korak 2: Instalacija NodeJS-a

Osnovu naše aplikacije činiće npm, menadžer paketa koji se koristi u JavaScript-u, pa samim tim I u Angular-u. Da bismo dobili pristup npm-u, potrebno je instalirati NodeJS sa oficijalnog sajta. Instalacija je laka, i svodi se na praćenje instrukcija.



Korak 3: Instalacija Angular CLI-a i pravljenje novog projekta

Potrebno je otvoriti folder u okviru Visual Studio Code-a, i otvoriti terminal.

Pomoću npm-a, instaliraćemo Angular CLI - konzolu koja će znatno olakšati rad na projektu.

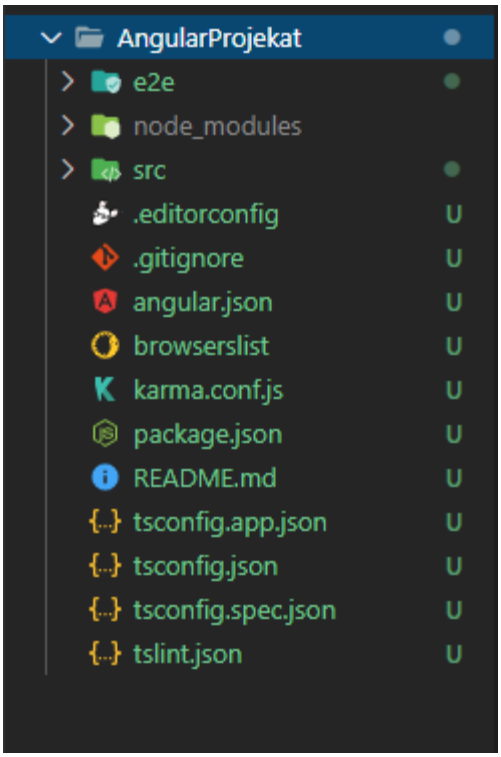
```
npm install -g @angular/cli
```

Nakon što se instalacija završi, pomoću sledeće komande, pravimo novi angular projekat, sa imenom po želji

```
ng new AngularProjekat
```

Na pitanje da li želimo da koristimo Angular Routing odgovaramo sa da (y), a kod odabira stylesheet-a, biramo CSS.

Struktura napravljenog projekta izgleda ovako:



Korak 4: Instaliranje Bulme

Napomena: Obavezno pomoću cd komande preći u novonapravljeni folder.

Za CSS framework koristićemo framework sa nazivom Bulma. Bulma je izgrađena na dobro poznatom BootStrap-u, i omogućuje nam mnogo lakše raspoređivanje i stilizovanje komponenti na ekranu. Komanda za instalaciju Bulme pomoću npm-a je sledeća:

```
npm install bulma
```

Da bismo uključili Bulmu u naš projekat, potrebno je da sledeću liniju ubacimo u naš angular.json fajl.


"assets": [
"src/favicon.ico",
"src/assets"
],
"styles": [
"src/styles.css",
"node_modules/bulma/css/bulma.min.css"
],
"scripts": []
},
"configurations": {
"production": {
"fileReplacements": [

Korak 5: Pokretanje aplikacije

Da bismo konstantno mogli da pratimo izmene na našoj aplikaciji, podići ćemo server na kome ćemo moći da posmatramo napravljene izmene, koji će raditi na localhost://4200. Komanda za pokretanje servera je sledeća:

```
ng serve
```

Sada smo spremni da počnemo pravi rad na aplikaciji!

Korak 6: Pravljenje početnih komponenti - Header i HomePage

Napomena: Obrisati sav početni kod iz App komponente

Pravimo dve početne komponente koje će činiti početnu osnovu naše aplikacije - Header i HomePage. Svaka komponenta se sastoji od tri dela - .html fajla, koji sadrži html kod, tj. raspored elemenata na ekranu, .css fajl, koji sadrži dizajn naše komponente, i .ts fajl koji sadrži svu logiku koja nam je neophodna.

Komande za pravljenje komponenti:

```
ng generate component header
```

```
ng generate component homePage
```

Za dizajn onoga što nam je neophodno u okviru komponenata koristićemo prednosti koje nam omogućava Bulma. Kod ćemo, po potrebi nadograđivati u toku rada.

Kod za komponentu Header:

```
<nav class="navbar is-warning" role="navigation" aria-label="main navigation">
  <div class="navbar-brand">
    <a class="navbar-item">
      <strong>Elab Prodavnica</strong>
    </a>
  </div>

  <div class="navbar-menu">
    <div class="navbar-start">
      <a class="navbar-item">
        Home
      </a>

      <a class="navbar-item">
        About Us
      </a>
    </div>
  </div>
</nav>
```

Početni kod ne uzima u obzir mobilne uređaje, samim tim ne implementira takozvano burger dugme. Dodatne funkcionalnosti će biti naknadno dodate.

Takođe, .css i .ts fajlovi za ovu komponentu nisu izmenjeni.

Kod za komponentu HomePage:

```
<hr />
<div class="columns is-multiline">
  <div *ngFor="let product of products" class="column is-4">
    <div class="card">
      <div class="card-image">
        <figure class="image is-4by3">
          <img src="{{ product.image }}" alt="Product image" />
        </figure>
      </div>
      <div class="columns">
        <div class="column is-7 is-offset-1">
          <strong>{{ product.title }}</strong>
        </div>
        <div class="column is-4">
          <i>{{ product.price }} RSD</i>
        </div>
      </div>
    </div>
  </div>
</div>
<hr />
```

```
.card {
    margin: auto;
    width: 80%
}
```

```
export class HomePageComponent implements OnInit {      You, 23 minutes ago • Added initial header and home page
  constructor() {}

  products: Product[] = [
    {
      id: '1',
      title: 'Voda',
      price: 40,
      image:
        'https://images.pexels.com/photos/416528/pexels-photo-416528.jpeg?auto=compress&cs=tinysrgb&dpr=1&w=500',
    },
    {
      id: '2',
      title: 'Sok',
      price: 60,
      image: 'https://nsnbc.me/wp-content/uploads/2019/07/fruit-juice.jpg',
    },
    {
      id: '3',
      title: 'Špagete',
      price: 80,
      image:
        'https://pbs.twimg.com/profile_images/423290202443251712/HYR23k2c.jpeg',
    },
    {
      id: '4',
      title: 'Makarone',
      price: 90,
      image: 'https://bonapeti.rs/files/lib/600x350/makaroni-vkusensos1.jpg',
    },
  ];

  ngOnInit(): void {}
}
```

Par veoma bitnih napomena:

Da bi ovaj kod radio, potrebno je napraviti interfejs Product, koji sadrži sve što jedan proizvod treba da sadrži, tj. id, naziv, URL slike, i cenu. Potrebno je napraviti fajl product.model.ts, i u njemu napisati sledeći kod:

```
export interface Product {
  id: string;
  title: string;
  image: string;
  price: number;
}
```

U okviru ts fajla pravimo takozvane dummy podatke, koji će nam služiti za rad dok ne napravimo bazu u kojoj će podaci biti čuvani.

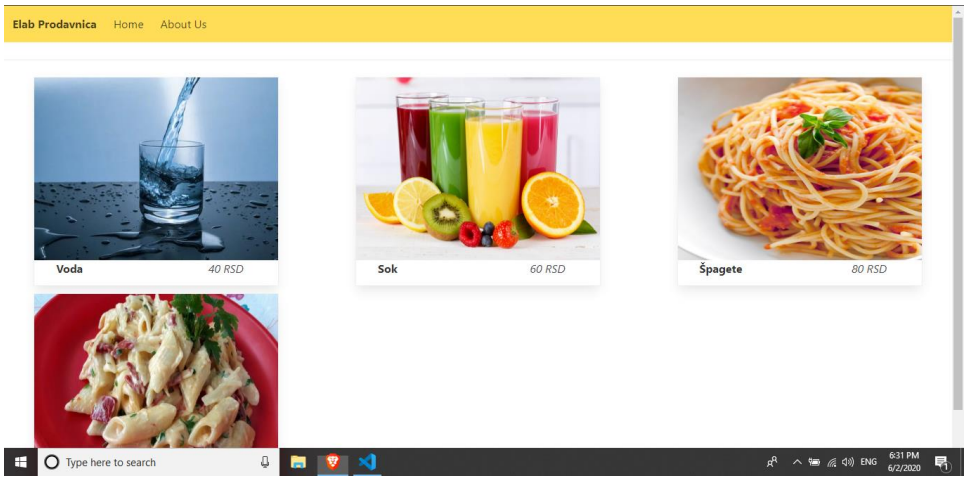
Sam izgled ove komponente je sačinjen od nekoliko ključnih koncepata Angulara i Bulme. Za svaki proizvod koristimo bulminu klasu Card, responzivnost je obezbeđena pomoću podele na kolone, tj. klase columns, a prolazak kroz svaki element liste omogućen je strukturalnom direktivom ngFor, koja u pozadini pravi for petlju za svaki element liste, i tako generiše novi neophodni html kod.

Tekst unutar dva para vitičastih zagrada je tekst koji će naknadno biti ubačen pomoću posebnog koncepta Angulara koji se naziva String interpolacija.

Da bismo videli ovaj kod u akciji, potrebno je da u html fajlu glavne app komponente ispišemo sledeći kod:

```
<app-header></app-header>
<app-home-page></app-home-page>
```

Naša aplikacija, koju trenutno možemo videti na localhost/4200, za početak izgleda ovako:



Slike su naravno, nasumično odabrane preko interneta.

Korak 7: Brisanje postojećih proizvoda

Da bismo mogli da obrišemo proizvode iz liste, potrebno je da u okviru svake kartice dodamo dugme za brisanje elementa. Potrebno je izmeniti kod iz card-content div-a, da bude kao kod na sledećoj slici:

```
<div class="card-content">
  <div class="columns" style="align-items: center;">
    <div class="column is-4 is-offset-1">
      <strong>{{ product.title }}</strong>
    </div>
    <div class="column is-3">
      <i>{{ product.price }} RSD</i>
    </div>
    <div class="column is-2">
      <a class="button is-danger" (click)="deleteProduct(product)"
        >Delete</a
      >
    </div>
  </div>
</div>
```

Kao što se može videti, na novo dugme smo stavili click listener - svaki put kada kliknemo na dato dugme, pokrenuće se funkcija koje je naznačena unutar znakova za navod. Potrebno je da istoimenu funkciju napravimo i u okviru .ts fajla.

```
deleteProduct(product: Product) {
  this.products = this.products.filter((prod) => {
    return prod !== product;
  });
  console.log(this.products);
}
```

Označeni proizvod ćemo izbaciti iz početne liste pomoću JavaScript funkcije za nizove - Filter.

Korak 8: Pravljenje nove stranice za dodavanje proizvoda

Za početak, napravićemo novu komponentu, u okviru koje ćemo dodavati nove produkte u postojeću listu.

```
ng g c newProduct
```

Pre nego što počnemo da uređujemo kod unutar komponente, potrebno je da napravimo početni skup ruta koje će omogućiti kretanje kroz aplikaciju, kao i dugme koje će nas dovesti do nove komponente.

Što se tiče dugmeta, ubacićemo ga u već postojeći heder:

```
<div class="navbar-menu">
  <div class="navbar-start">
    <a
      class="navbar-item"
      [routerLink]="['/home']"
      routerLinkActive="is-active"
    >
      Home
    </a>

    <a class="navbar-item">
      About Us      You, 3 hours ago
    </a>
  </div>

  <div class="navbar-end">
    <a
      class="navbar-item"
      [routerLink]="['/add']"
      routerLinkActive="is-active"
    >
      Add New
    </a>
  </div>
</div>
```

Pored dugmeta, može se uočiti i dodat kod vezan za rute. Sada ćemo postaviti rute za našu aplikaciju, izmenama u okviru app.routing.module.ts i app.component.html fajlova.

```
const routes: Routes = [
  {
    path: 'home',
    component: HomePageComponent,
  },
  {
    path: 'add',
    component: NewProductComponent,
  },
  {
    path: '',      You, 8 minutes ago • Uncommitted changes
    redirectTo: 'home',
    pathMatch: 'full',
  },
];

You, 8 minutes ago | 1 author (You)
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

```
<app-header></app-header>
<router-outlet></router-outlet>
```

Poslednja ruta služi za automatsku redirekciju u slučaju da se pristupi praznom url-u, što je ustaljena praksa u svim aplikacijama pravljenim u Angular-u.

Korak 9: Pravljenje servisa za proizvode.

Najbolji način za komunikaciju između komponenti u okviru Angular aplikacija su servisi, koji u sebi čuvaju zajedničke informacije. Napravićemo servis koristeći Angular CLI, i izmestiti listu proizvoda, kao i metodu za brisanje jednog proizvoda iz HomePage komponente u novonastali servis.

```
ng g service product
```

Takođe, da bismo u celoj aplikaciji redovno apdejtovali grafički interfejs, moramo da napravimo sistem za obaveštavanje svih komponenti o stanju podataka, preko koncepta BehaviourSubject-a.

Izmenjeni ts fajl HomePage komponente sada izgleda ovako:

```
export class HomePageComponent implements OnInit, OnDestroy {
  constructor(private service: ProductService) {}

  products: Product[] = [];
  subscription: Subscription;

  ngOnInit(): void {
    this.subscription = this.service.productSubject.subscribe(
      (products) => (this.products = products)
    );
  }

  deleteProduct(product: Product) {
    this.service.deleteProduct(product);
  }

  ngOnDestroy(): void {
    if (this.subscription) {
      this.subscription.unsubscribe();      You, a minute ago •
    }
  }
}
```

Koncept koji se ovde koristi je sastavni deo Angular-a, iako dolazi iz paketa koji nije napravljen od strane Angular developera. U pitanju je koncept Observable-a iz paketa rxjs. Pomoću njega, svaka komponenta može da se "sabskrajbuje" na promene koje dolaze iz servisa, i samim tim da apdejtuje svoj sadržaj pri svakoj izmeni. Da se pretplate ne bi gomilale, implementiramo ngOnDestroy, i u okviru njega brišemo pretplatu, i time oslobađamo memoriju.

Sam servis izgleda ovako:

```
import { Injectable } from '@angular/core';
import { Product } from './product.model';
import { BehaviorSubject } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class ProductService {
  products: Product[] = [ ···
  ];

  productSubject = new BehaviorSubject<Product[]>(this.products);

  constructor() {}

  deleteProduct(product: Product) {
    this.products = this.products.filter((prod) => {
      return prod !== product;
    });
    this.productSubject.next(this.products);
  }
}
```

Korak 10: Dodavanje forme za unos novog proizvoda

Komponenta NewProduct će sadržati formu za unos novog elementa. U okviru Angular-a možemo razlikovati dva tipa forme - Template Driven forme i Reactive forme. Takozvane reaktivne forme daju više slobode programerima, i, iako su malo teže za savladati, u većini slučajeva predstavljaju bolji izbor.

Da bismo mogli da ih koristimo, u nasem app.module typescript fajlu ćemo FormsModule zameniti novim ReactiveForms modulom.

Na sledećim slikama se može videti css, html, i typescript deo NewProduct komponente. Rekativne forme se incijalizuju u ts fajlu, a zatim naknadno povezuju sa odgovarajućim elementima u html fajlu. Validatori nam omogućavaju da odredimo šta je sve potrebno da bi naša forma bila spremna za čuvanje. U našem slučaju, postavili smo da sva tri polja moraju da budu popunjena. Složeniji sistem validacije će biti implementiran kasnije.

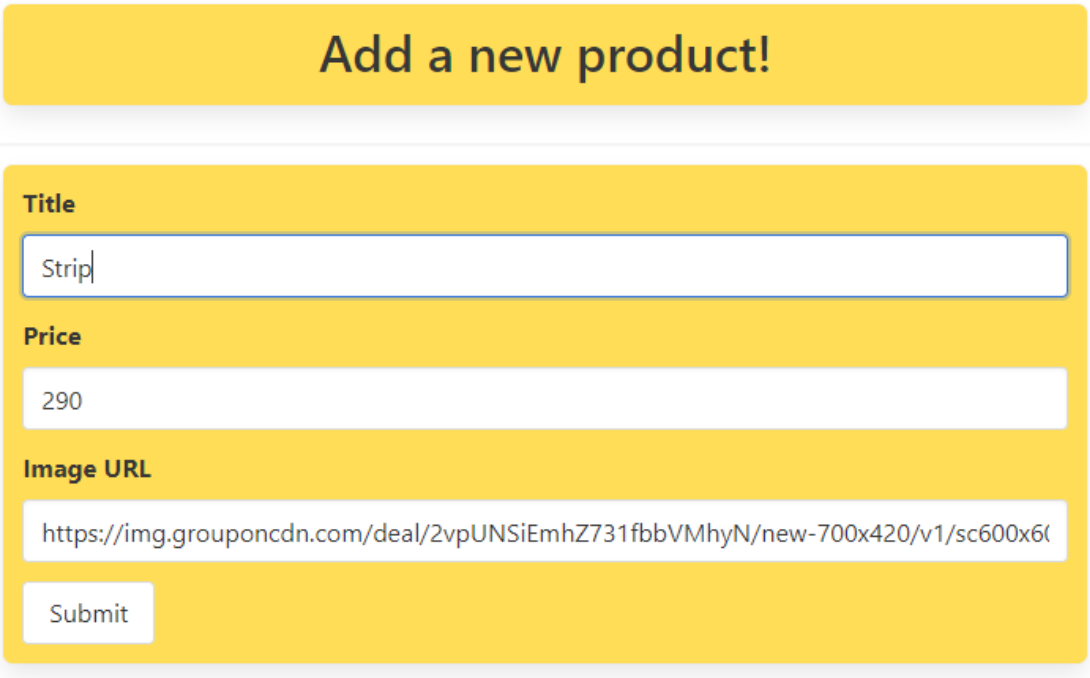
```
.box {
  background-color: hsl(48, 100%, 67%);
  margin: auto;
  max-width: 80%;
}
```

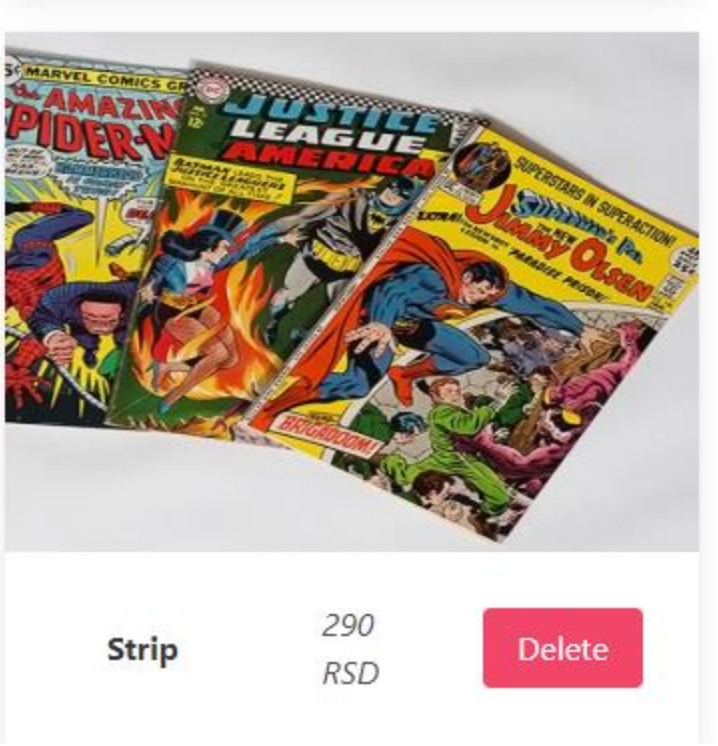
```
<hr />
<div class="columns">
  <div
    class="column is-6 is-offset-3 box"
    style="text-align: center; padding-bottom: 16px;"
  >
    <h1 class="title">
      Add a new product!
    </h1>
  </div>
</div>
<hr />
<div class="columns">
  <div class="column is-6 is-offset-3 box">
    <form [formGroup]="productForm" (submit)="onSubmit()">
      <div class="field">
        <div class="control">
          <label class="label">Title</label>
          <input
            type="text"
            class="input"
            placeholder="Enter the product name"
            formControlName="title"
          />
        </div>
      </div>
```

```
      <div class="field">
        <div class="control">
          <label class="label">Price</label>
          <input
            class="input"
            type="number"
            placeholder="Enter the product price"
            formControlName="price"
          />
        </div>
      </div>
      <div class="field">
        <div class="control">
          <label class="label">Image URL</label>
          <input
            class="input"
            type="text"      You, 16 minutes ago • Uncommitted changes
            placeholder="Enter the image url"
            formControlName="image"
          />
        </div>
      </div>
      <div class="control">
        <button [disabled]="!productForm.valid" type="submit" class="button">
          Submit
        </button>
      </div>
    </form>
  </div>
</div>
<hr />
```

```
export class NewProductComponent implements OnInit {
  constructor(private service: ProductService, private router: Router) {}

  productForm: FormGroup;

  initForm() {
    this.productForm = new FormGroup({
      title: new FormControl('', [Validators.required]),
      price: new FormControl('', [Validators.required]),
      image: new FormControl('', [Validators.required]),
    });
  }

  ngOnInit(): void {
    this.initForm();
  }

  onSubmit() {
    const tit = this.productForm.controls.title.value;
    const pr = this.productForm.controls.price.value;
    const im = this.productForm.controls.image.value;
    const product: Product = {
      id: (this.service.products.length + 1).toString(),
      image: im,
      price: pr,
      title: tit,
    };
    this.service.addNewProduct(product);
    this.router.navigateByUrl('/home');      You, 9 minutes ago • Uncommitted
  }
}
```

Forma vizuelno izgleda ovako, a novonastali proizvod vizuelno isto kao i svi postojeći. Slika je, naravno, nasumično odabrana.





Aplikacija sada ima većinu funkcionalnosti koje smo želeli, u njihovom osnovnom obliku. Doduše, susrećemo se sa jednim problemom - sve podatke kojima rapsolažemo možemo naći u okviru našeg koda. Cilj je da se lista koju trenutno čuvamo u našem servisu, izmestimo u neku bazu. Da bismo simulirali ovaj proces, i koristili REST API za prikazivanje i menjanje podataka, koristićemo JSONPlaceholder. Bazi možemo pristupiti preko sledećeg linka: https://my-json-server.typicode.com/Stevan-Radovanovic/FakeJSONdb/products

Korak 11: Komunikacija između aplikacije i baze preko REST API-ja

Da bismo mogli da komuniciramo sa bazom, potrebno je da izmenimo kod u našem servisu za proizvode. Koristićemo HttpClient, koji je ugrađen u Angular, i omogućava slanje svih HTTP zahteva na veoma lak način. Koristićemo koncept Observables-a, iz paketa rxjs, koji smo već koristili do sad.

```
getProducts() {
  this.http
    .get<Product[]>(
      'https://my-json-server.typicode.com/Stevan-Radovanovic/FakeJSONdb/products'
    )
    .subscribe((response) => {
      this.products = response;
      this.productSubject.next(this.products);
    });
}
```

```
deleteProduct(product: Product) {
  this.http
    .delete(
      'https://my-json-server.typicode.com/Stevan-Radovanovic/FakeJSONdb/products/' +
        product.id
    )
    .subscribe(() => {
      this.products = this.products.filter((prod) => {
        return prod !== product;      You, a few seconds ago • Uncommitted changes
      });
      this.productSubject.next(this.products);
    });
}
```

```
addNewProduct(product: Product) {
  this.http
    .post(
      'https://my-json-server.typicode.com/Stevan-Radovanovic/FakeJSONdb/products',
      product
    )
    .subscribe((response) => {
      this.products.push(product);
      this.productSubject.next(this.products);
    });
}
```

Da bi sve funkcionisalo kao ranije, potrebno je dodati još jednu sitnicu u app.component.ts. Ovaj dodatak će nam omogućiti da, pri svakom pokretanju aplikacije, povlačimo podatke iz baze.

```
constructor(private service: ProductService) {
  this.service.getProducts();
}
```

Aplikacija je sada funkcionalna!

Ceo kod se može naći na linku do repozitorijuma naznačenom u uvodnom pasusu. Naravno, postoje mnogi načini za dalje unapređivanje ove aplikacije, a ovde će biti nabrojani neki:

1. Dodavanje burger dugmeta u meniju za navigaciju

2. Dodavanje autorizacije tj. opcije za log in i register

3. Dodavanje loading spinera prilikom interakcije sa bazom

4. Pravljenje prave funkcionalne baze pomoću MongoDB, MySQL, ili nekog drugog servisa