Dokumentáció

Futtatás

A projekt futtatásához a következő függőségekre van szükség: `node.js, mongodb`

A projekt futtatása `docker` segítségével:

```
docker compose up
```

A projekt futtatása docker nélkül:

```
cd backend
npm i
npm start
cd ../frondend
npm i
npm start
```

A mongodb adatbázisnak futnia kell, alapből a `27017`-es porton.

A mongodb kapcsolódó URI-t a `/backend` mappában lévő `.env` fájlban lehet definiálni a `.env.sample` fájl alapján.

A felhasználói bejelentkezéshez és a backend hozzáféréshez generálni kell egy `ACCESS_TOKEN_SECRET` környezet változót és eltárolni a `.env` fájlban.

Bevezető

A beadandó témája egy alap nyílvántartó rendszer, amelyben a felhasználók termékek listájához képesek hozzáférni, látják, hogy melyik terméket melyik felhasználó adta hozzá a nyilvántartáshoz. A felhasználók képesek a maguk által hozzáadott termékeket szerkeszteni és törölni. A weboldal rendelkezik egy alap kezdőoldallal, amely a beadandó felada követelményeit tartalmazza. Minden oldal rendelkezik egy 'top-bar' komponensel, amely gombokat tartalmaz. A felhasználók be tudnak jelentkezni a bejelentkezés oldalon, ha nem rendelkeznek felhasználói fiókkal, akkor regisztrálhatnak a regisztáció oldalon. A projekt rendelkezik egy express backend-el, amely a mongodb adatbázis módosítását és lekérdezését kezeli, továbbá hitelesíti a felhasználókat. A frontend angular material komponenseket használ. A projekt összesen 9 komponensből áll: top-bar, welcome-screen, user-login, user-register, registry, product, product-add, user-page, centered-card. A projekt 3 service komponenst tartalmaz: product, user, auth. A projekt egy darab guard komponenst tartalmaz: auth.

kezdőlap

A kezdőlap tartalmaz egy centerd-card komponenst, amely tartalmazza a beadandó feladat követelményeit. A centered-card komponens az angular material card komponens `mat-card` a képernyő közepére igazítva.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-centered-card',
  template: `
    <div class="div-centered">
      <mat-card><ng-content></ng-content></mat-card>
    </div>
  `,
  styles: [
    `
      .div-centered {
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
      }
    `,
  ],
})
export class CenteredCardComponent {}
```

A `centered-card` komponens tartalmazza az `ng-content` komponenst, amellyel bármilyen egyéb html vagy egyéb komponenst lehet belevetíteni.

Bejelentkezés oldal

A bejelentkezési oldal egy centered-card komponensből áll, amely tartalmaz egy gombot, egy felhasználói regisztrációs `user-register` komponenshez linkelő linket és egy űrlapot tartalmaz, amely egy email és egy jelszó mezőkből áll. Az űrlap mezők Angular material input komponent (`mat-form-field`) használnak. A projektben található összes gomb, pedig Angular material gomb komponenst (`mat-button`). A jelszó mező tartalmaz egy gombot, amely lehetővé teszi a felhasználó számára, hogy láthassa a jelszavát. A bejelentkezési menű csak akkor engedi a felhasználót bejelentkezni, ha kitöltötte az email és jelszó mezőket. Ha a felhasználó nem tölti ezeket ki, akkor piros színű hibaüzenet fog megjelenni az input mezők fölött, valamint hibaüzenet fog megjelenni, ha az email nem valid formátummal rendelkezik. A 'bejelentkezés' gombra csak akkor lehet kattintani, ha az email és a jelszó mezők ki vannak töltve. A bejelentkezés gombra kattintva egy kérést küldünk a backend felé, amely tartalmazza az emailt és a jelszót. Ha a bejelentkezés sikeres volt, akkor egy `snackbar` komponens fogja a felhasználót informálni, hogy a bejelentkezés sikeres volt-e vagy sem. Sikeres bejelentkezés esetén válaszként érkezik egy `jwt` token, amely 7 napig érvényes és a böngésző lokális tárhelyében van eltárolva, valamint a felhasználó automatikusan navigálva lesz a kezdőlapra. A `top-bar` komponensen található gombok változnak attól függően, hogy melyik oldalon van a felhasználó. A

kezdőlapon látható egy 'nyilvántartás' gomb, amely a kezdőlapra navigálja a felhasználót. Ha a felhasználó nincs bejelentkezve, akkor a jobb oldalon látható lesz a bejelentkezés gomb. Abban az esetben, ha a felhasználó már bejelentkezett, láthatóak lesznek a termékek, kijelentkezés és felhasználó ikon gombok. A termékek gomb a nyílvántartását tartalmazó `registry` komponenshez navigálja a felhasználót. A kijelentkezés gomb kijelentkezteti a felhasználót. A felhasználó ikon, pedig a felhasználó oldal, `user-page` komponenshez navigálja a felhasználót. A bejeletkezés oldalon csak a kezdőlapra vezető 'nyilvántartás' gomb látható.

```
onSubmit() {
  this.loginSubscription = this.authService
    .login(this.loginForm.value)
    .subscribe({
      next: () => {
        this.snackBar.open('Sikeres bejelentkezés', 'ok', { duration: 3000 });
        this.router.navigate(['/']);
      },
      error: (error) => {
        const code = error.status;
        if (code !== 500) {
          this.snackBar.open(error.error.message, 'ok', { duration: 3000 });
        }
      },
    });
}
```

Felhasználó hitelesítő (auth service)

A felhasználó hitelesítést kezelő `AuthService` szolgáltatás a bejelentkezés, kijelentkezés és a hitelesített felhasználó kezelésével foglalkozik. Az email és jelszó alapján visszatér egy `jwt` access tokennel, amelyet a böngésző helyi tárhelyében tárol. A kijelentkezés során ezt a tokent kitörli a helyi tárhelyből. A jelenleg bejelentkezett felhasználó adatait a `jwt` token tartalmazza.

```
login(loginRequest: LoginRequest): Observable<LoginResponse> {
    return this.http
      .post<LoginResponse>(`http://localhost:3000/auth/signin`, loginRequest)
      .pipe(
        tap((res: LoginResponse) => {
          this.setAccessToken(res.accessToken);
          this.token = res.accessToken;
          this.currentUserSubject.next(this.getLoggedInUser());
        }),
        catchError((err) => {
          console.error(err);
          throw err;
        })
```

```
    );
  }
```

A szolgáltatás akkor jelzi, hogy a felhasználó be van jelentkezve, ha rendelkezik access tokennel és a token még nem járt le. Ha a token lejárt, akkor újra be kell jelentkeznie a felhasználónak.

```
isLoggedIn() {
  return this.token && this.jwtService.isTokenExpired(this.token);
}
```

Felhasználó regisztráció

A felhasználó regisztráció komponens (`user-register`) egy `centered-card` komponensben található űrlapot és egy gombot tartalmaz. Az űrlapban kötelezően meg kell adni egy felhasználónevet, emailt és jelszót, valamint opcionálisan meg lehet adni telefonszámot, vezetéknevet és keresztnevet. Az űrlap piros hibaüzenettel jelzi, ha az email nem megfelelő formátummal rendelkezik, ha a felhasználónév hiányzik, a jelszó rövidebb, mint 8 karakter vagy a jelszó megerősítése hibás. A gomb csak akkor lesz elérhető, ha az email, felhasználónév és jelszó mezők helyesen ki vannak töltve. Sikeres regisztráció eredményét egy `snackbar` komponens fogja megjeleníteni. A backend hibát fog jelezni, ha az email vagy a felhasználónév már használatban van. A regisztrációs oldalon a `top-bar` komponensen megjelenik a bejelentkezés gomb. Sikeres bejelentkezés esetén a felhasználót automatikusan visszanavigálja a kezdőlapra.

```
onSubmit() {
  this.userRegisterSubscription = this.userService
    .register(this.registerForm.value)
    .subscribe({
      next: () => {
        this.snackBar.open('Sikeres regisztráció', 'ok', { duration: 3000 });
        this.router.navigate(['/']);
      },
      error: (error) => {
        const code = error.status;

        if (code !== 500) {
          if (error.error?.message) {
            this.snackBar.open(error.error.message, 'ok', { duration: 3000 });
          } else {
            console.error(error);
          }
        }
      },
    });
}
```

Nyilvántartás

A nyilvűntartás komponensben egy táblázat (`mat-table`) és egy lapozó (`mat-paginator`) komponens található. A táblázat a termékek nevét, márkáját, árát, a terméket hozzáadó felhasználó nevét és a termék kategóriáját tartalmazza. Ha a felhasználó rákattint egy termékre, akkor elnavigál a termék oldalára (`product` komponens). A fenti `paginator` komponensben a felhasználó kiválaszthatja, hogy egy oldalon hány termék jelenjen meg és léphet a következő vagy előző oldalra. A komponens minden lapozás során lekérdezi az aktuális kiválasztott számú terméket az adatbázisból. A nyílvántartás oldal url query paramétereket is tartalmaz, amellyel meg lehet adni, hogy hanyadik oldalt nyissa meg és, hogy hány termék jelenjen meg. A nyilvántartás oldalt csak bejelentkezett felhasználók láthatják. A `top-bar` gombok közül megjelent a hozzáad gomb, amire kattintva elnavigál a `product-add` komponenshez.

```
loadProducts(event: any) {
  const { pageIndex, pageSize } = event;
  this.getProductsSubscription = this.productService
    .getProducts(pageIndex * pageSize, pageSize)
    .subscribe({
      next: (products) => (this.dataSource = products),
      error: (error) => console.error(error),
    });

  this.setUrlQueryParams(pageIndex, pageSize);
}

ngOnInit() {
  this.queryParamSubscription = this.route.queryParams.subscribe((params) => {
    this.currentPage = params['page'] ?? 0;
    this.pageSize = params['size'] ?? 10;
  });

  this.getProductsSubscription2 = this.productService
    .getProducts(0, this.pageSize)
    .subscribe({
      next: (products) => (this.dataSource = products),
      error: (error) => console.error(error),
    });

  this.getProductsCountSubscription = this.productService
    .getProductsCount()
    .subscribe({
      next: (products) => (this.productsLength = products.length),
      error: (error) => console.error(error),
    });
```

```
}
```

Termék szolgáltatás

A termék szolgáltatás a termékek kezelésére szolgál. A termék szolgáltatás segítségével lehet a backendről lekérdezni a termékeket, lehetőség van továbbá a termékek frissítésére és törlésére is. A termék szolgáltatás függvényei `Observable` adatfolyamot hoznak létre, amelyre a komponensek feltudnak íratkozni.

```
deleteProduct(id: string) {
  return this.http.delete<{ message: string }>(
    `http://localhost:3000/products/${id}`
  );
}

addProduct(product: {
  name?: string | null;
  brand?: string | null;
  price?: number | null;
  category?: string | null;
  registeredBy?: string;
}) {
  return this.http.post<{ message: string }>(
    `http://localhost:3000/products/add`,
    {
      product: product,
    }
  );
}

updateProduct(product: Product) {
  return this.http.put<{ message: string }>(
    `http://localhost:3000/products/${product._id}`,
    {
      product: {
        name: product.name ?? undefined,
        brand: product.brand ?? undefined,
        price: product.price ?? undefined,
        category: product.category ?? undefined,
      },
    }
  );
}

getProduct(id: string) {
  return this.http.get<Product>(`http://localhost:3000/products/${id}`);
```

```
}

getProducts(index: number, count: number) {
  return this.http.get<Product[]>(
    `http://localhost:3000/products/lower/${index}/${count}`
  );
}

getProductsCount() {
  return this.http.get<{ length: number }>(
    'http://localhost:3000/products/count'
  );
}
```

Termék oldal

A termék oldal tartalmazza a termék adatait, valamint lehetővé teszi a termék hozzáadójának, hogy szerkessze vagy törölje az adott terméket. A szerkesztés gombra kattintva a termék adatait lehet szerkeszteni, a kattintás után a szerkesztés gomb 'mentés' gombra változik. Az űrlapot vissza lehet állítani a mégse gombra kattintva.

```
export class ProductComponent implements OnInit, OnDestroy {
  productId!: string;
  product!: Product;
  edit: boolean;
  action: string;
  productModifyForm!: FormGroup;
  matcher = new FormErrorStateMatcher();
  currentUserId!: string | undefined | null;
  paramsSubscription!: Subscription;
  productSubscription!: Subscription;
  currentUserSubscription!: Subscription;
  updateProductSubscription!: Subscription;

  constructor(
    private route: ActivatedRoute,
    private router: Router,
    private productService: ProductService,
    private snackBar: MatSnackBar,
    private authService: AuthService
  ) {
    this.edit = false;
    this.action = 'Szerkeszt';

    this.productModifyForm = new FormGroup({
```

```
    name: new FormControl(null, [Validators.required]),
    brand: new FormControl(null, [Validators.required]),
    price: new FormControl(null, [Validators.required]),
    category: new FormControl(null, [Validators.required]),
  });

  this.productModifyForm.disable();
}

isOwner() {
  return this.product && this.product.registeredBy === this.currentUserId;
}

onDelete() {
  this.productService.deleteProduct(this.productId).subscribe({
    next: () => {
      this.snackBar.open('Termék törölve', 'ok', { duration: 3000 });
      this.router.navigate(['/registry']);
    },
    error: (error) => console.error(error),
  });
}

cancel() {
  this.productModifyForm.reset(this.product);
  this.toggleEditMode();
}

toggleEditMode() {
  if (!this.edit) {
    this.action = 'Mentés';
    this.edit = true;
    this.productModifyForm.enable();
  } else {
    this.action = 'Szerkeszt';
    this.edit = false;
    this.productModifyForm.disable();
  }
}

onSubmit() {
  this.updateProductSubscription = this.productService
    .updateProduct({ _id: this.productId, ...this.productModifyForm.value })
    .subscribe({
      next: () => {
        this.snackBar.open('Termék frissítve', 'ok', { duration: 3000 });
```

```
          this.router.navigate(['/registry']);
        },
        error: (error) => console.error(JSON.stringify(error, null, 2)),
      });
  }

  ngOnInit() {
    this.paramsSubscription = this.route.params.subscribe((params) => {
      this.productId = params['id'];
    });
    this.productSubscription = this.productService
      .getProduct(this.productId)
      .subscribe({
        next: (data) => {
          this.product = data;
          this.productModifyForm.patchValue(data);
        },
        error: (error) => console.error(error),
      });

    this.currentUserSubscription = this.authService.currentUser$.subscribe({
      next: (user) => (this.currentUserId = user?.id),
      error: (error) => console.error(error),
    });
  }

  ngOnDestroy(): void {
    this.paramsSubscription.unsubscribe();
    this.productSubscription.unsubscribe();
    this.currentUserSubscription.unsubscribe();

    this.updateProductSubscription?.unsubscribe();
  }
}
```

Termék hozzáadása oldal

Lehetővé teszi a bejelentkezett felhasználók számára, hogy terméket hozzanak létre.

```
onSubmit() {
  this.currentUserSubscription = this.authService.currentUser$.subscribe({
    next: (user) => {
      console.log(user);
      this.addProductSubscription = this.productService
        .addProduct({
```

```
          ...this.productAddForm.value,
          registeredBy: user?.id,
        })
        .subscribe({
          next: () => {
            this.snackBar.open('Termék hozzáadva', 'ok', { duration: 3000 });
            this.router.navigate(['/registry']);
          },
          error: (error) => console.error(error),
        });
    },
    error: (error) => console.error(error),
  });
  this.router.navigate(['/registry']);
}
```

Felhasználó oldal

A jelenleg bejelentkezet felhasználó adatait jeleníti meg.

```
<app-centered-card class="user-page">
  <mat-card-title>Felhasználó adatok</mat-card-title>
  <mat-card-content>
    <p>{{ user?.name }} - {{ user?.lastName }} {{ user?.firstName }}</p>
    <p>{{ user?.email }}</p>
  </mat-card-content>
</app-centered-card>
```

Felhasználó szolgáltatás

A felhasználó szolgáltatás `userService`, lehetővé felhasználók létrehozását és a felhasználók nevének lekérdezésére id alapján.

```
export class UserService {
  constructor(private http: HttpClient) {}

  getUserNameById(id: string) {
    return this.http.get<{ name: string }>(`http://localhost:3000/users/${id}`);
  }

  register(data: RegisterData) {
    const registerRequest = {
      user: data,
    };
    const headers = new HttpHeaders({
      'Content-Type': 'application/json',
    });
```

```
    return this.http.post<{ message: string }>(
      'http://localhost:3000/users/add/',
      registerRequest,
      {
        headers,
      }
    );
  }
}
```