

Sklep Komputerowy

Spis treści

| | |
|------------------------------|----|
| Opis:..... | 2 |
| Strona główna | 3 |
| Navbar, routing | 4 |
| Firebase | 8 |
| Rejestracja użytkownika..... | 9 |
| Logowanie | 13 |
| Podsumowanie | 16 |

Opis:

Projekt zakładał stworzenie aplikacji internetowej służącej do obsługi internetowego sklepu komputerowego. Głównym celem było umożliwienie użytkownikom przeglądania i kupowanie dostępnych produktów, rejestracji, logowania oraz zarządzania danymi klientów. Projekt został zrealizowany głównie przy użyciu technologii Angular, Bootstrapa oraz Firebase.

W ramach projektu powstała strona główna sklepu, na której wyświetlany jest przykładowy wygląd. Wykorzystując Bootstrapa, udało się zapewnić responsywność treści, co zwiększa interaktywność strony.

Dodanie systemu routingu umożliwiło płynne przechodzenie między różnymi sekcjami strony, co poprawia ogólne doświadczenie użytkownika.

Formularze rejestracji i logowania zostały zintegrowane z usługą Firebase Authentication, co pozwoliło na bezpieczne przechowywanie danych użytkowników oraz autoryzację podczas logowania. Dane użytkowników, takie jak imię, nazwisko czy numer telefonu, są przechowywane w bazie danych Firebase Realtime Database. Podczas rejestracji nowego użytkownika, tworzony jest odpowiedni użytkownik w bazie danych Authentication oraz obiekt zawierający dane w Realtime Database.

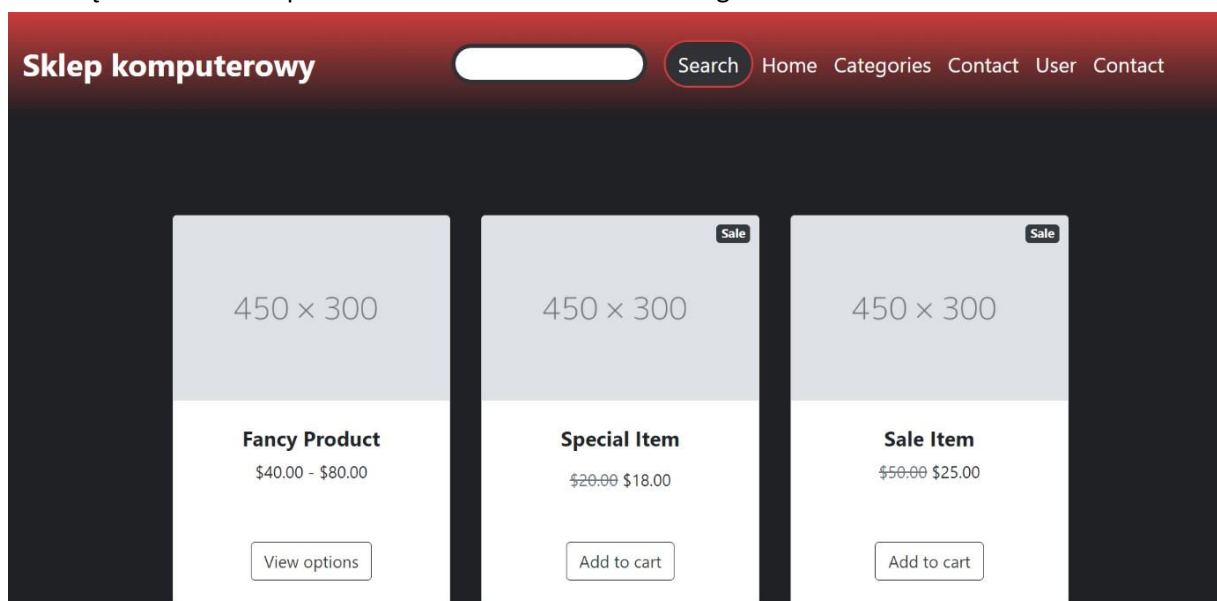
Ważnym elementem projektu jest również zabezpieczenie przy logowaniu i rejestracji. Dzięki zastosowaniu walidacji danych oraz funkcjom firebase do obsługi sesji, zapewniona jest poprawność procesu logowania i rejestracji.

Dodatkowo, dla zalogowanych użytkowników, stworzono stronę domową, na której wyświetlane są podstawowe informacje o koncie.

Warto zauważyć, że projekt jest wciąż rozwijany, a w przyszłości planowane są dalsze rozszerzenia oraz dodanie nowych funkcji, mających na celu jeszcze lepsze dostosowanie aplikacji do potrzeb użytkowników.

Strona główna:

Strona główna jest oparta o Bootstrap. Wyświetlane są przykładowe kafelki które w przyszłości zostaną zamienione na produkty ładowane z Firebase Storage



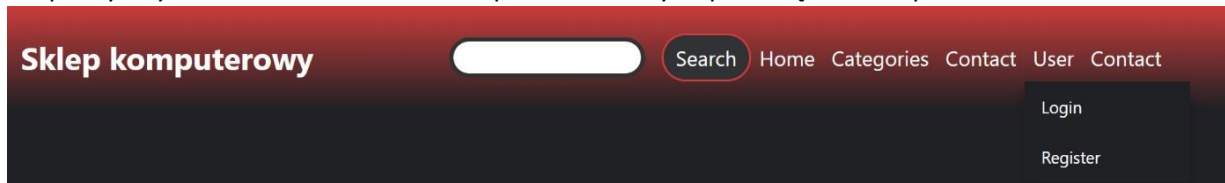
Rysunek 1 - strona główna

```
<> home.component.html x  TS user.class.ts  TS user-home.component.ts  TS firebase.auth.ts  <> user-home.component.html
1  <section class="py-5">
2  <div class="container px-4 px-lg-5 mt-5">
3    <div class="row gx-4 gx-lg-5 row-cols-2 row-cols-md-3 row-cols-xl-4 justify-content-center">
4      <div class="col mb-5">
5        <div class="card h-100">
6          <!-- Product image-->
7          
8          <!-- Product details-->
9          <div class="card-body p-4">
10             <div class="text-center">
11               <!-- Product name-->
12               <h5 class="fw-bolder">Fancy Product</h5>
13               <!-- Product price-->
14               $40.00 - $80.00
15             </div>
16           </div>
17           <!-- Product actions-->
18           <div class="card-footer p-4 pt-0 border-top-0 bg-transparent">
19             <div class="text-center"><a class="btn btn-outline-dark mt-auto" href="#">View options</a></div>
20           </div>
21         </div>
22       </div>
```

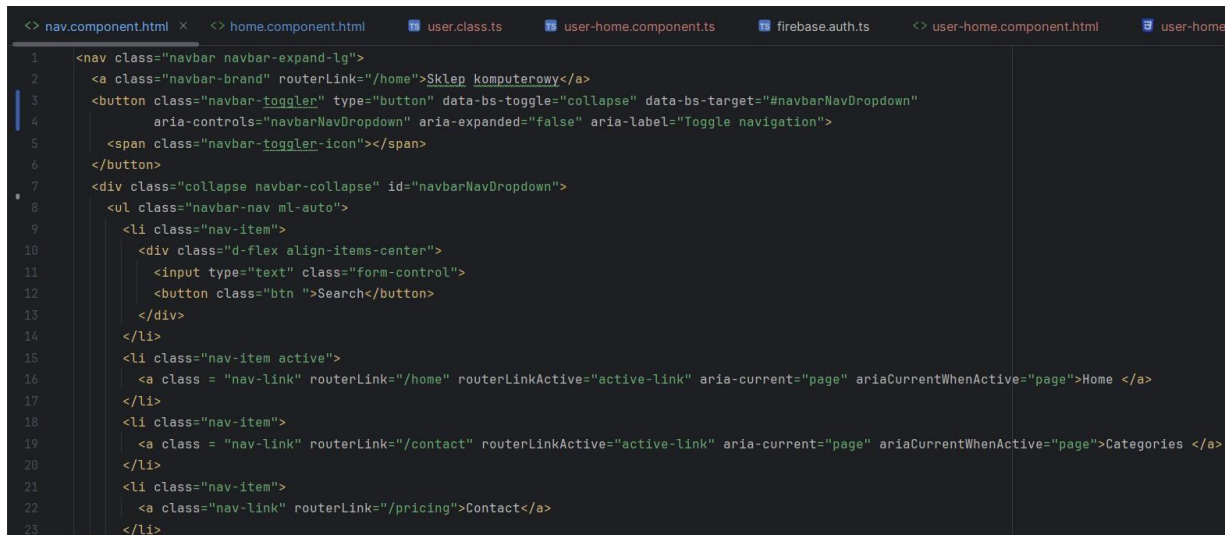
Rysunek 2- kod HTML strony głównej

Navbar, routing

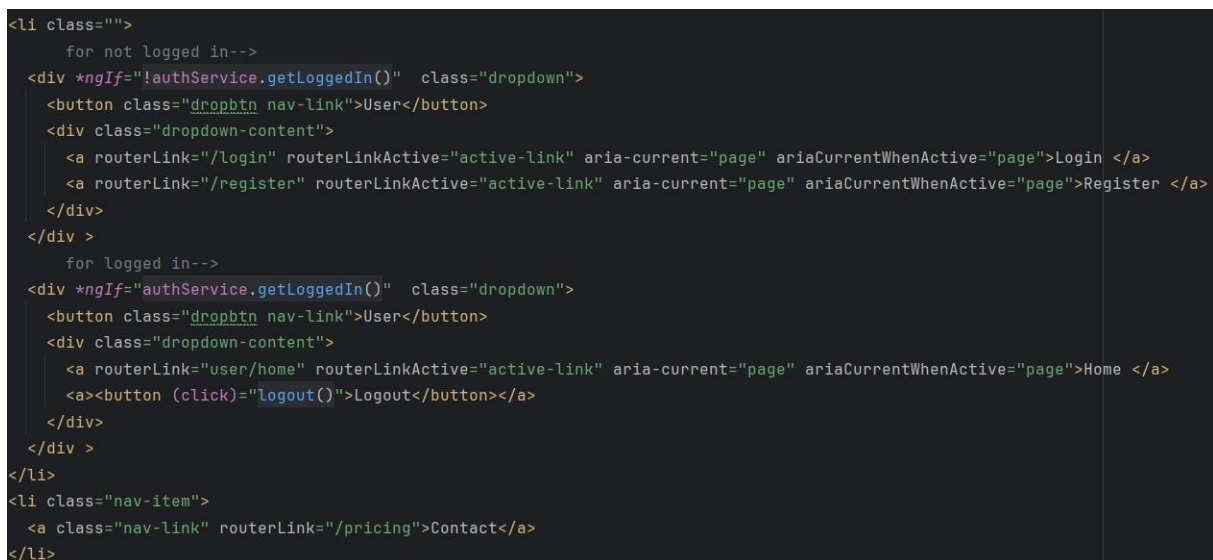
Rezponywny navbar również został zaimplementowany za pomocą bootstrapa.



Rysunek 3 - widok navbar



Rysunek 4 - HTML navbar



Rysunek 5- sprawdzanie czy użytkownik zalogowany

W celu ładowania podstron wykorzystałem wbudowany w Angulara system routingu. Obsługuje się go za pomocą atrybutu routerLink w linku do podstrony.

Komponent nav zawiera w sobie funkcje sprawdzające czy użytkownik jest zalogowany i automatycznie wyświetla dostępne podstrony

```
nav.component.ts x <> nav.component.html <> home.component.html TS user.class.ts TS user-home.comp

11 @Component({
12   selector: 'app-nav',
13   templateUrl: './nav.component.html',
14   styleUrls: ['./nav.component.css'],
15   providers: [AuthenticationServiceComponent],
16 })
17 export class NavComponent {
18   no usages new *
19   constructor(
20     protected authService: AuthenticationServiceComponent,
21     private cdr: ChangeDetectorRef,
22     private appComponent: AppComponent,
23     private router: Router) {}
24
25   no usages new *
26   ngOnInit(){ // Funkcja ta pozwala sprawdzić status zalogowania użytkownika w czasie rzeczywistym
27     this.appComponent.loginStatusChange.subscribe( observerOrNext: async(status: boolean | null)=> {
28       this.authService.setLoggedIn(status || false);
29       this.cdr.detectChanges();
30     }
31   )
32 }
33
34 1+ usages new *
35 async logout(){
36   if(await UserSession.logout()){
37     await this.router.navigate( commands: ['login']);
38   }
39 }
```

```

1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { AuthGuard } from '../auth_guard/auth.guard';
4 import { UnauthGuard } from '../auth_guard/unauth.guard';
5
6 //import subpages
7 > import ...
12
13 const routes: Routes = [
14   {path: '', redirectTo: '/home', pathMatch: 'full'},
15   {path: 'home', component: HomeComponent},
16   {path: 'contact', component: ContactComponent},
17   {path: 'register', component: RegisterComponent, canActivate: [UnauthGuard]},
18   {path: 'login', component: LoginComponent, canActivate: [UnauthGuard]},
19   {path: 'user/home', component: UserHomeComponent, canActivate: [AuthGuard]}
20 ];
21
22 1+ usages new *
23 @NgModule({
24   imports: [RouterModule.forRoot(routes)],
25   exports: [RouterModule]
26 })
27 export class AppRoutingModule {

```

Rysunek 7 - algorytm odpowiedzialny za routowanie (zabezpieczenie - canActivate)

Dodanie atrybutu canActivate blokuje dostęp do strony user.home i przycisku logout

```

9 export class AuthGuard {
10   no usages
11   constructor(private authService: AuthenticationServiceComponent, private router: Router) {}
12   no usages
13   canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | UrlTree {
14     return this.checkLogin(route, state.url)
15   }
16
17   1+ usages
18   private checkLogin(route: ActivatedRouteSnapshot, url: string): boolean | UrlTree {
19     if(this.authService.isLoggedIn() || this.authService.isLoggedIn() == null){
20       return true;
21     }
22     else {
23       return this.router.createUrlTree([ '/login' ], { navigationExtras: { queryParams: { returnUrl: url } } });
24     }
25   }
26 }

```

Rysunek 6 - authGuard sprawdza czy użytkownik jest zalogowany

```

9   export class UnauthGuard {
    no usages
10   constructor(private authService: AuthenticationServiceComponent, private router: Router) {}
    no usages
11   canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | UrlTree{
12     return this.checkLogin(state.url)
13   }
14
15   1+ usages
16   private checkLogin(url: string): boolean | UrlTree{
17     if(!this.authService.getLoggedIn()){
18       return true;
19     }
20     else {
21       return this.router.createUrlTree( commands: ['/user'], navigationExtras: { queryParams: { returnUrl: url}});
22     }
23   }

```

Rysunek 8 - UnAuthGuard sprawdza czy użytkownik nie jest zalogowany

```

4   @Injectable({
5     providedIn: 'root'
6   })
7
8   export class AuthenticationServiceComponent {
9     loggedIn : boolean | null;
    1+ usages
10    public setLoggedIn(value: boolean){
11      this.loggedIn = value;
12    }
    1+ usages
13    public getLoggedIn(){
14      return this.loggedIn;
15    }
16
    1+ usages
17    constructor() {
18      this.loggedIn = false;
19    }
20  }

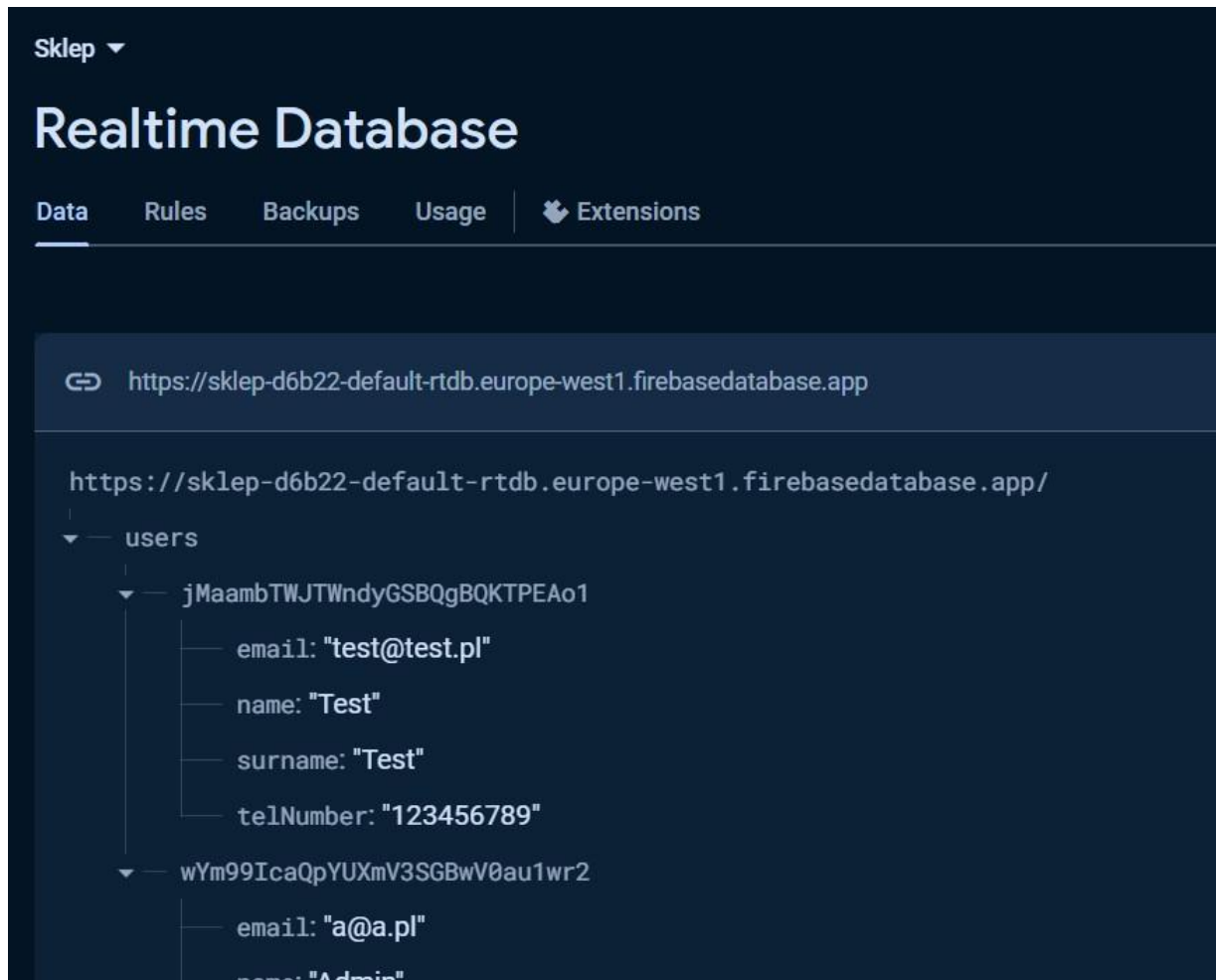
```

Rysunek 9 - klasa przechowująca status zalogowania

Status zalogowania użytkownika przechowywany jest w AuthenticationServiceComponent

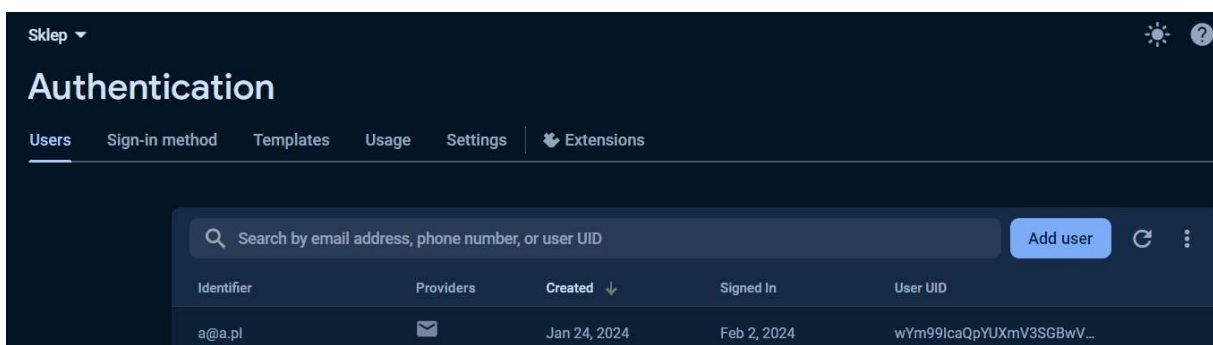
Firestore

Jako bazę danych wybrałem Firestore. W projekcie utworzyłem dwie bazy danych – Realtime Database i Authentication. W Realtime Database są przechowywane obiekty użytkowników kluczem podstawowym są id użytkowników pobrane z serwisu Authentication



Rysunek 8 - Realtime Database Firebase

Serwis Firebase Authentication przechowuje dane do logowania użytkowników: email, hasło i uid użytkownika (hasło nie jest dostępne).



Rysunek 9 - Authentication Firebase

Rejestracja użytkownika

The image shows a registration form with a dark background and white input fields. The form is outlined with a red border. It contains the following fields and labels:

- Name: Input field with placeholder text "Name".
- Surname: Input field with placeholder text "Surname".
- Email address: Input field with placeholder text "Enter email".
- Telephone number: Input field with placeholder text "123456789".
- Password: Input field with placeholder text "Password".
- Repeat password: Input field with placeholder text "Repeat password".
- Submit: A button labeled "Submit".
- Form Status: INVALID

Rysunek 10 - wygląd formularza rejestracji

W pliku `register.component` tworzę obiekt `FormBuilder` który posłuży mi do pobierania i walidacji danych z formularza

```
register.component.ts x <> register.component.html app-r...

24 @Component({
25   standalone: true,
26   selector: 'app-register',
27   templateUrl: './register.component.html',
28   styleUrls: ['./register.component.css'],
29   imports: [ReactiveFormsModule, NgIf],
30 })
31
32 export class RegisterComponent {
33
34   db = new DataBase();
35   no usages new *
36   constructor(private formBuilder: FormBuilder) {}
37
38   nameError:string = '';
39   surnameError:string = '';
40   emailError:string = '';
41   telError:string = '';
42   password1Error:string = '';
43   password2Error:string = '';
```

Rysunek 11 - "Budowanie" formularza

```
TS register.component.ts x <> register.component.html TS app-routing.module.ts TS login.c
139   registerForm = this.formBuilder.group( controls: {
140     name: ['', [Validators.required, this.nameValidation()]],
141     surname: ['', [Validators.required, this.surnameValidation()]],
142     email: ['', [Validators.required, this.emailValidation()]],
143     telNumber: ['', [Validators.required, this.telNumberValidation()]],
144     password1: ['', [Validators.required, this.password1Validation()]],
145     password2: ['', [Validators.required, this.password2Validation()]],
146
147   });
148
149   1+ usages  Jakub *
149   validate(){ //walidacja formularza
150     if(this.registerForm.valid){ //jeżeli jest poprawny zarejestruj użytkownika
151       registerUser(
152         name: this.registerForm.get('name')?.value ?? '',
153         surname: this.registerForm.get('surname')?.value ?? '',
154         email: this.registerForm.get('email')?.value ?? '',
155         telNumber: this.registerForm.get('telNumber')?.value ?? '',
156         password: this.registerForm.get('password1')?.value ?? ''
157       );
158     }
159   }
```

Rysunek 12 - pobieranie danych i walidacja

```

108 password1Validation(): ValidatorFn {
109     return (control: AbstractControl): ValidationErrors | null => {
110         const password1Re: RegExp = /^(?=.*\d)(?=.*[A-Z])(?=.*[!@#$%^&*])[0-9A-Za-z!@#$%^&*]{12,}$/;
111         const test = password1Re.test(control.value);
112
113         if (!test) {
114             this.password1Error = "Niepoprawny hasło"
115             return { forbiddenName: { value: control.value } };
116
117         } else {
118             this.password1Error='';
119             return null;
120         }
121     };
122 }

```

Rysunek 14 - przykładowa funkcja walidująca hasło

Funkcje walidujące pola formularza

| | |
|--|---|
| Name <input type="text" value="a"/> <small>Imię musi mieć więcej niż 2 litery</small> | Surname <input type="text" value="a"/> <small>Nazwisko musi mieć więcej niż 2 litery</small> |
| Email address <input type="text" value="a"/> <small>Niepoprawny email</small> | Telephone number <input type="text" value="a"/> <small>Niepoprawny nr telefonu</small> |
| Password <input type="password" value="a"/> <small>Niepoprawny hasło</small> | Repeat password <input type="password" value="b"/> <small>hasła różnią się</small> |
| <input type="button" value="Submit"/> Form Status: INVALID | |

Rysunek 13 - komunikaty o błędach w formularz rejestracji

Funkcje dodające nowego użytkownika w Firebase

```
48 export const registerUser = async (name: string, surname: string, email: string, telNumber: string, password: string) => {
49   const user = await createUserWithEmailAndPassword(auth, email, password);
50   writeUserData(user.user.uid, name, surname, email, telNumber);
51 }
52
53 1+ usages Jakub
53 function writeUserData(uid: string, name: string, surname: string, email: string, telNumber: string) {
54   const reference = ref(db, path: 'users/' + uid);
55   set(reference, value: {
56     name: name,
57     surname: surname,
58     email: email,
59     telNumber: telNumber
60   });
61 }
```

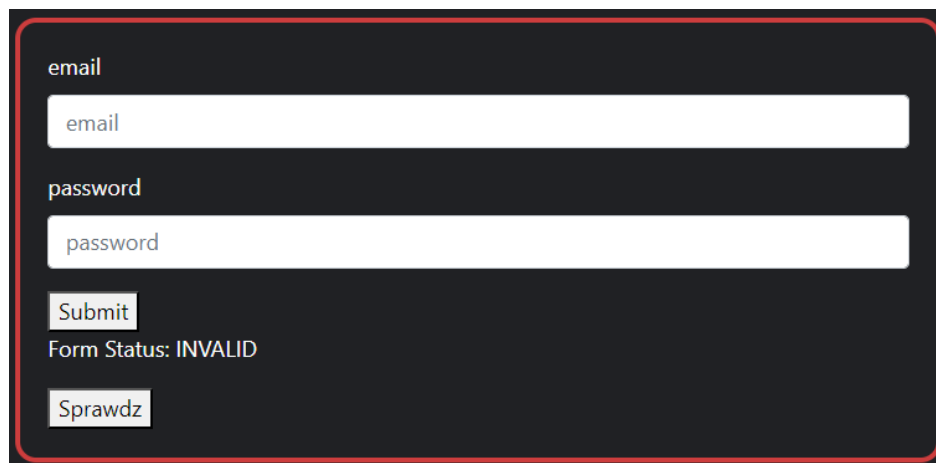
Rysunek 15 - funkcja dodająca nowego użytkownika i zapisująca dane

Firebase automatycznie hashuje dane oraz dodaje sól

Firebase Authentication uses an internally modified version of bcrypt to hash account passwords. Even when an account is uploaded with a password using a different algorithm, Firebase Auth will rehash the password the first time that account successfully logs in. Accounts downloaded from Firebase Authentication will only ever contain a password hash if one for this version of bcrypt is available, or contain an empty password hash otherwise.

Rysunek 16 - informacja ze strony Firebase

Logowanie



The image shows a login form on a dark background. It contains two input fields: one labeled 'email' and another labeled 'password'. Below these fields is a 'Submit' button. Under the button, the text 'Form Status: INVALID' is displayed. At the bottom of the form is a button labeled 'Sprawdz' (Check).

Rysunek 17 - formularz logowania

Obsługa pobierania danych z formularza i walidacji zrobiona podobnie jak w przypadku rejestracji

Jeżeli formularz poprawny zaloguj użytkownika

```
63   async validate(){
64     if(this.loginForm.valid){
65       if(await UserSession.loginUser(
66         email: this.loginForm.get('email')?.value ?? '',
67         password: this.loginForm.get('password')?.value ?? '')){
68         await this.router.navigate( commands: ['user/home']);
69       }
70     }
71   }
```

Rysunek 18 - funkcja sprawdzająca czy formularz poprawny

Funkcja setPersistence tworzy sesję użytkownika

```
80   static loginUser = async (email: string, password: string) => {
81     try {
82       await setPersistence(auth, browserSessionPersistence);
83       const user = await signInWithEmailAndPassword(auth, email, password);
84       //console.log(user.user.uid);
85       return true;
86     } catch (error: any) {
87       console.log(error);
88       return false;
89     }
90   }
```

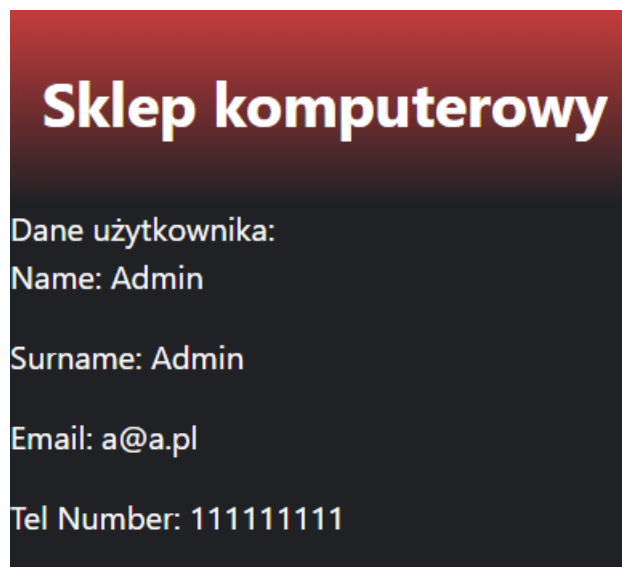
Rysunek 19 - funkcja logująca użytkownika i tworząca sesję

Funkcja sprawdzająca zmianę statusu zalogowania użytkownika w czasie rzeczywistym.

```
21 export class AppComponent {
22   public loginStatusChange: Subject<boolean | null> = new Subject<boolean | null>();
23   user1 = user;
24   title = 'sklep-projekt-JP';
25   Auth = new AuthenticationServiceComponent();
26   1+ usages new *
27   checkLogin(){
28     onAuthStateChanged(auth, nextOrObserver: (user : User | null ) => {
29       if(user){
30         this.user1 = user;
31         this.Auth.setLoggedIn(true)
32         console.log("zalogowano")
33       }else {
34         console.log("wylogowano")
35         this.Auth.setLoggedIn(false)
36         this.user1 = null;
37       }
38       this.loginStatusChange.next(this.Auth.getLoggedIn());
39       //AuthenticationService.setCurrentUser(user);
40     })
41   }
```

Rysunek 20 - sprawdzanie zmianę statusu zalogowania w czasie rzeczywistym

Po zalogowaniu się użytkownik zostaje przeniesiony na stronę /user/home



Rysunek 21 - wyświetlanie danych użytkownika

Funkcja pobierająca dane z Firebase

```
69   async readUserData(){
70     const reference = ref(db, path: 'users/' + auth.currentUser?.uid);
71     return new Promise( executor: (resolve) => {
72       onValue(reference, callback: (snapshot : DataSnapshot ) => {
73         const data = snapshot.val();
74         const user = new User(data.name, data.surname, data.email, data.telNumber);
75         resolve(user);
76       });
77     })
78   }
```

Rysunek 23 - funkcja pobierająca dane z firebase

```
ts user-home.component.ts  x  <> user-home.component.html  TS login.cc
8   @Component({
9     selector: 'app-user-home',
10    templateUrl: './user-home.component.html',
11    standalone: true,
12    styleUrls: ['./user-home.component.css']
13  })
14  export class UserHomeComponent implements OnInit{
15    name = "";
16    surname= "";
17    email= "";
18    telNumber= "";
19
20    no usages
21    constructor(private userSession: UserSession) {}
22
23    no usages
24    async ngOnInit(){
25      let user: User | undefined;
26      user = await this.userSession.readUserData() as User;
27      this.name = user.getName();
28      this.surname = user.getSurname();
29      this.email = user.getEmail();
30      this.telNumber = user.getTelNumber();
31    }
```

Rysunek 22 odbieranie danych z readData i zapisywanie do zmiennych

Podsumowanie

W projekcie udało się zrealizować podstawowe funkcję związane z obsługą użytkowników takie jak logowanie, rejestracja, pobieranie danych z bazy danych i zabezpieczenie tych danych.

Zaimplementowane mechanizmy obsługi formularzy, obsługi sesji, walidacji danych i hashowanie hasła z solą. Projekt jest wciąż rozwijany. Planowane jest dodanie przedmiotów sprzedawanych w sklepie pobieranych z Firebase Storage, możliwość kupienia przedmiotu przez użytkownika, dodanie systemu ról (Admin, Moderator), dodanie panelu administratora pozwalającego na dodanie nowych przedmiotów do sklepu i zarządzanie kontami użytkowników, dodanie systemu recenzji.