

Programowanie zaawansowane

II

Ćwiczenia

Igor Barcik 131780

June 23, 2024

Contents

1 Budowanie nowej aplikacji w Angular	2
1.1 Dodanie logowania	3
1.1.1 login.component.html	4
1.1.2 login.component.scss	7
1.1.3 login.component.ts	8
1.1.4 auth.service.ts	9
1.2 Testy jednostkowe	11
1.3 Dodawanie dashboardu	13
1.3.1 Utworzenie komponentu dashboard	13
1.3.2 Utworzenie routingu	14
1.3.3 Utworzenie komponentu menu	15
1.3.4 Utworzenie testów dla komponentu dashboard	18
2 Tworzenie testów automatycznych systemu w podejściu TDD	21
3 Testowania testów automatycznych systemu w podejściu BDD	23
4 Testowanie pojedynczego modułu, który jest zależny od kodu zewnętrznego.	25
5 Weryfikowanie czy moduły zależne współpracują między sobą w sposób poprawny.	28

1 Budowanie nowej aplikacji w Angular

Krok 1: Przygotowanie środowiska

- Upewnij się, że masz zainstalowanego *Node.js* na swoim komputerze.
- Zainstaluj narzędzie **Angular CLI** globalnie za pomocą polecenia `npm install -g @angular/cli`.

Krok 2: Utworzenie nowego projektu Angular

- Utwórz nowy projekt Angular, wykonując polecenie `ng new nazwa-projektu`. Postępuj zgodnie z instrukcjami w konsoli.

Krok 3: Rozwinięcie projektu

- Przejdź do katalogu projektu, wykonując `cd nazwa-projektu`.
- Twórz komponenty, usługi i edytuj szablony, aby dostosować projekt do swoich potrzeb.

Krok 4: Uruchomienie aplikacji

- Uruchom aplikację za pomocą polecenia `ng serve`.

Krok 5: Testowanie i rozwijanie

- Kontynuuj rozwijanie aplikacji, testowanie i dostosowywanie jej do swoich potrzeb.

A terminal window showing the following command and its output:

```
^A . ~/Doc/u/Sem5/Programowanie zaawansowane II/angular main:1 +2 13 ?24 > ll 66 node --version 66 bun --version
drwxr-xr-x - biggy biggy 2 Mar 13:46 node_modules
drwxr-xr-x - biggy biggy 26 May 09:28 src
drwxr-xr-x 2.8k biggy biggy 26 Mar 12:40 angular.json
drwxr-xr-x 381k biggy biggy 26 Mar 12:40 tsconfig.json
drwxr-xr-x 450k biggy biggy 2 Mar 12:41 package-lock.json
drwxr-xr-x 1,1k biggy biggy 3 Mar 21:51 package.json
drwxr-xr-x 1,1k biggy biggy 2 Mar 12:40 README.md
drwxr-xr-x 283 biggy biggy 2 Mar 12:40 tsconfig.app.json
drwxr-xr-x 900 biggy biggy 2 Mar 12:40 tsconfig.json
drwxr-xr-x 273 biggy biggy 2 Mar 12:40 tsconfig.spec.json
v22.2.0
1.1.10
```

The terminal window also shows the system status bar at the bottom indicating node v22.2.0 and bun v1.1.10.

Figure 1: Przygotowane środowisko, node v22.2.0, bun v1.1.10.

1.1 Dodanie logowania

Krok 1: Utworzenie ekranu logowania

- W terminalu przejdź do katalogu projektu za pomocą `cd nazwa-projektu`.
- Utwórz komponent logowania za pomocą Angular CLI: `ng generate component logowanie`.
- Edytuj plik `login.component.html`, aby utworzyć formularz logowania.
- W pliku `login.component.ts` zaimplementuj funkcję logowania.

Krok 2: Uruchomienie aplikacji

- Uruchom aplikację za pomocą polecenia `ng serve`.
- Twój aplikacji będzie dostępna pod adresem `http://localhost:4200/`.

Krok 3: Testowanie i rozwijanie

- Kontynuuj rozwijanie aplikacji, dodawaj nowe funkcjonalności i testuj je.

- Monitoruj jakość kodu i utrzymuj projekt w dobrej kondycji.

1.1.1 login.component.html

Zaprojektowanie prostego formularza logowania zawierającego pola *login* i *password*. Zawiera również bloki z komunikatami o błędach. Całość wystylizowana odrobiną SCSS'a.

```
File Edit Selection View Go Run Terminal Help <-- > angular
src/app/login/login.component.html > login.component.html > login.component.ts

1+ <div class="login-container">
2+   <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
3+     <div class="form-group">
4+       <label for="username">Üzüktownik:</label>
5+       <input type="text" id="username" name="username" [(ngModel)]="formData.username" required>
6+     </div>
7+
8+     <div class="form-group">
9+       <label for="password">Hasło:</label>
10+      <input type="password" id="password" name="password" [(ngModel)]="formData.password" required>
11+    </div>
12+
13+   </div>
14+   <div class="button-group">
15+     <button type="submit" [disabled]="!form.invalid">Zaloguj</button>
16+     <button type="reset" [disabled]="!loginForm.valid" (click)="onReset()> Reset</button>
17+   </div>
18+ </form>
19+ </div>
20+ </div>
21+ </div>
22+ </div>
```

```
src/app/login/login.component.ts > login.component.ts
```

```
1+ import { Component } from '@angular/core';
2+ import { FormBuilder, FormGroup, Validators } from '@angular/forms';
3+ import { Router } from '@angular/router';
4+
5+ @Component({
6+   selector: 'app-login',
7+   templateUrl: './login.component.html',
8+   styleUrls: ['./login.component.css']
9+ })
10+ export class LoginComponent {
11+   loginForm: FormGroup;
12+
13+   constructor(private fb: FormBuilder, private router: Router) {
14+     this.loginForm = this.fb.group({
15+       username: ['', Validators.required],
16+       password: ['', Validators.required]
17+     });
18+   }
19+
20+   onSubmit() {
21+     if (this.loginForm.valid) {
22+       // Perform login logic here
23+       this.router.navigate(['/home']);
24+     }
25+   }
26+
27+   onReset() {
28+     this.loginForm.reset();
29+   }
30+ }
```

Figure 2: Różnice między proponowanym rozwiązaniem a zaimplementowanym `login.component.html`

```

1 <div class="login-container">
2   <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
3     <div class="form-group">
4       <label for="username">Username:</label>
5       <input type="text" id="username" formControlName="username" required />
6       <div
7         class="error"
8         *ngIf="
9           loginForm.get('username')?.touched &&
10          loginForm.get('username')?.invalid
11        "
12      >
13        <div *ngIf="loginForm.get('username')?.hasError('required')">
14          Username is required
15        </div>
16      </div>
17    </div>
18
19    <div class="form-group">
20      <label for="password">Password:</label>
21      <input
22        type="password"
23        id="password"
24        formControlName="password"
25        required
26      />
27      <div
28        class="error"
29        *ngIf="
30          loginForm.get('password')?.touched &&
31          loginForm.get('password')?.invalid
32        "
33      >
34        <div *ngIf="loginForm.get('password')?.hasError('required')">
35          Password is required
36        </div>
37      </div>
38    </div>
39
40    <div class="button-group">
41      <button type="submit" [disabled]="!loginForm.valid">Login</button>
42      <button type="reset" [disabled]="!loginForm.valid" (click)="onReset()">
43        Reset
44      </button>
45    </div>
46    <div class="error" *ngIf="loginError">Invalid credentials</div>
47    <div class="success" *ngIf="loginSuccess">You are logged in</div>
48  </form>
49</div>

```

Figure 3: Zaimplementowany login.component.html

1.1.2 login.component.scss

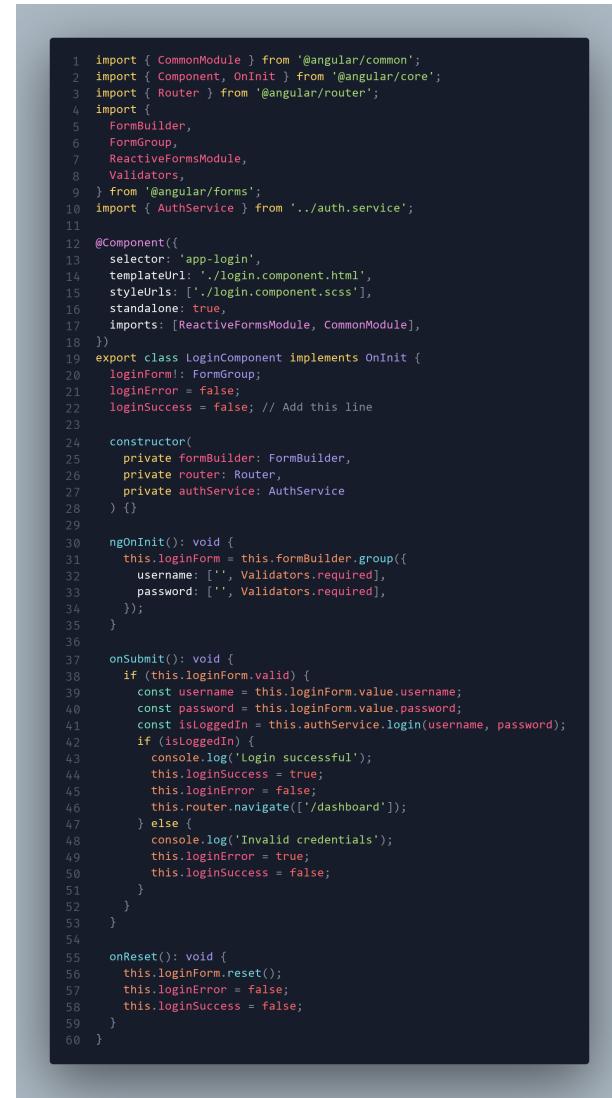
```
1 .login-container {
2   display: flex;
3   flex-direction: column;
4   align-items: center;
5   justify-content: center;
6   height: 100vh;
7   font-family: sans-serif;
8   .error {
9     color: red;
10    margin-top: 5px;
11    background-color: #f8d7da;
12    border-color: #f5c6cb;
13    padding: 10px;
14    border-radius: 4px;
15  }
16  .success {
17    color: green;
18    margin-top: 5px;
19    background-color: #d4edda;
20    border-color: #c3e6cb;
21    padding: 10px;
22    border-radius: 4px;
23  }
24  .form-group {
25    margin-bottom: 20px;
26  }
27
28  label {
29    display: block;
30    margin-bottom: 5px;
31  }
32
33  input {
34    width: 100%;
35    padding: 10px;
36    border: 1px solid #ccc;
37    border-radius: 4px;
38  }
39
40  button {
41    background-color: #007bff;
42    color: white;
43    padding: 10px 20px;
44    border: none;
45    border-radius: 4px;
46    cursor: pointer;
47
48    &:hover {
49      background-color: #0056b3;
50    }
51
52    &[disabled] {
53      background-color: #cccccc;
54      cursor: default;
55    }
56  }
57  .button-group {
58    display: flex;
59    gap: 10px;
60  }
61 }
```

Figure 4: Zaimplementowany login.component.scss

1.1.3 login.component.ts

Zawarcie prostej logiki komponentu logowania gdzie zwraca informacje o logowaniu do aplikacji wcześniej weryfikując dane wprowadzone przez użytkownika w komponencie serwisowym auth.service.ts.

Wzorowanie się na proponowanym rozwiążaniu załączonym przez prowadzącego.



```
1 import { CommonModule } from '@angular/common';
2 import { Component, OnInit } from '@angular/core';
3 import { Router } from '@angular/router';
4 import {
5   FormBuilder,
6   FormGroup,
7   ReactiveFormsModuleModule,
8   Validators,
9 } from '@angular/forms';
10 import { AuthService } from '../auth.service';
11
12 @Component({
13   selector: 'app-login',
14   templateUrl: './login.component.html',
15   styleUrls: ['./login.component.scss'],
16   standalone: true,
17   imports: [ReactiveFormsModuleModule, CommonModule],
18 })
19 export class LoginComponent implements OnInit {
20   loginForm!: FormGroup;
21   loginError = false;
22   loginSuccess = false; // Add this line
23
24   constructor(
25     private formBuilder: FormBuilder,
26     private router: Router,
27     private authService: AuthService
28   ) {}
29
30   ngOnInit(): void {
31     this.loginForm = this.formBuilder.group({
32       username: ['', Validators.required],
33       password: ['', Validators.required],
34     });
35   }
36
37   onSubmit(): void {
38     if (this.loginForm.valid) {
39       const username = this.loginForm.value.username;
40       const password = this.loginForm.value.password;
41       const isLoggedIn = this.authService.login(username, password);
42       if (isLoggedIn) {
43         console.log('Login successful');
44         this.loginSuccess = true;
45         this.loginError = false;
46         this.router.navigate(['/dashboard']);
47       } else {
48         console.log('Invalid credentials');
49         this.loginError = true;
50         this.loginSuccess = false;
51       }
52     }
53   }
54
55   onReset(): void {
56     this.loginForm.reset();
57     this.loginError = false;
58     this.loginSuccess = false;
59   }
60 }
```

Figure 5: Zaimplementowany login.component.ts

1.1.4 auth.service.ts

Komponent usługi odpowiedzialny za logowanie użytkownika.

```
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class AuthService {
7   private _isLoggedIn = false;
8
9   get isLoggedIn(): boolean {
10     return this._isLoggedIn;
11   }
12
13   login(username: string, password: string): boolean {
14     if (username === 'admin' && password === 'admin') {
15       this._isLoggedIn = true;
16       return true;
17     }
18     return false;
19   }
20
21   logout(): void {
22     this._isLoggedIn = false;
23   }
24 }
```

Figure 6: Zaimplementowany auth.service.ts

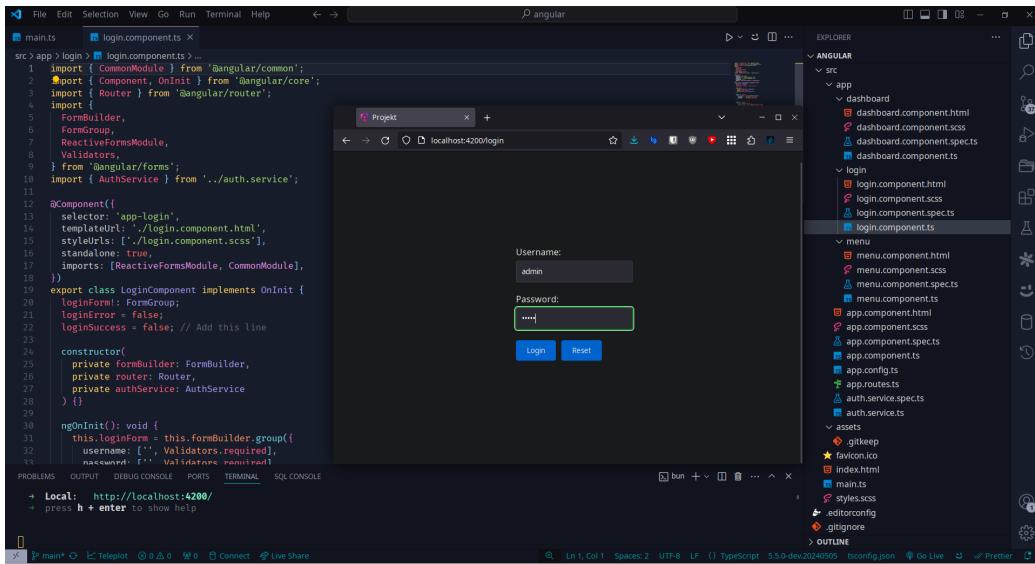


Figure 7: Gotowy komponent logowania

1.2 Testy jednostkowe

Napisanie testów dla komponentu logowanie, przy użyciu narzędzi takich jak Jasmine oraz Karma. Wzorowanie się na załączonym przez prowadzącego kodzie testu podczas pisania własnej implementacji.



```
1 import { ComponentFixture, TestBed } from '@angular/core/testing';
2 import { LoginComponent } from './login.component';
3 import { By } from '@angular/platform-browser';
4
5 describe('LoginComponent', () => {
6   let component: LoginComponent;
7   let fixture: ComponentFixture<LoginComponent>;
8   beforeEach(async () => {
9     await TestBed.configureTestingModule({
10       imports: [LoginComponent],
11     }).compileComponents();
12   });
13   beforeEach(() => {
14     fixture = TestBed.createComponent(LoginComponent);
15     component = fixture.componentInstance;
16     fixture.detectChanges();
17   });
18   it('should create the login', () => {
19     const fixture = TestBed.createComponent(LoginComponent);
20     const app = fixture.componentInstance;
21     expect(app).toBeTruthy();
22   });
23   it('should render a form', () => {
24     const fixture = TestBed.createComponent(LoginComponent);
25     fixture.detectChanges();
26     const compiled = fixture.nativeElement as HTMLElement;
27     expect(compiled.querySelector('form')).toBeTruthy();
28   });
29   it('should render a form with username and password fields', () => {
30     const fixture = TestBed.createComponent(LoginComponent);
31     fixture.detectChanges();
32     const compiled = fixture.nativeElement as HTMLElement;
33     expect(
34       compiled.querySelector('input[formControlName="username"]')
35     ).toBeTruthy();
36     expect(
37       compiled.querySelector('input[formControlName="password"]')
38     ).toBeTruthy();
39   });
40   it('should render a submit button', () => {
41     const fixture = TestBed.createComponent(LoginComponent);
42     fixture.detectChanges();
43     const compiled = fixture.nativeElement as HTMLElement;
44     expect(compiled.querySelector('button[type="submit"]')).toBeTruthy();
45   });
46   it('should render a reset button', () => {
47     const fixture = TestBed.createComponent(LoginComponent);
48     fixture.detectChanges();
49     const compiled = fixture.nativeElement as HTMLElement;
50     expect(compiled.querySelector('button[type="reset"]')).toBeTruthy();
51   });
52   it('should call the onSubmit method when the form is submitted', () => {
53     const fixture = TestBed.createComponent(LoginComponent);
54     const app = fixture.componentInstance;
55     spyOn(app, 'onSubmit');
56     fixture.detectChanges();
57     const compiled = fixture.nativeElement as HTMLElement;
58     compiled.querySelector('form').dispatchEvent(new Event('submit'));
59     expect(app.onSubmit).toHaveBeenCalled();
60   });
61   it('should display success message when login is successful', () => {
62     component.loginForm.setValue({
63       username: 'admin',
64       password: 'password',
65     });
66     component.onSubmit();
67     fixture.detectChanges(); // Ensure the DOM is updated
68     expect(component.loginSuccess).toBeTruthy(); // Add this line
69     const successMessage = fixture.debugElement.query(By.css('.success'));
70     expect(successMessage).toBeTruthy();
71     expect(successMessage.nativeElement.textContent).toContain(
72       'You are logged in'
73     );
74   });
75   it('should display error message when login fails', () => {
76     component.loginForm.setValue({
77       username: 'admin',
78       password: 'wrongpassword',
79     });
80     component.onSubmit();
81     fixture.detectChanges(); // Ensure the DOM is updated
82     const errorMessage = fixture.debugElement.query(By.css('.error'));
83     expect(errorMessage).toBeTruthy();
84     expect(errorMessage.nativeElement.textContent).toContain(
85       'Invalid credentials'
86     );
87   });
88 });


```

Figure 8: Własna implementacja funkcji testujących komponent logowania

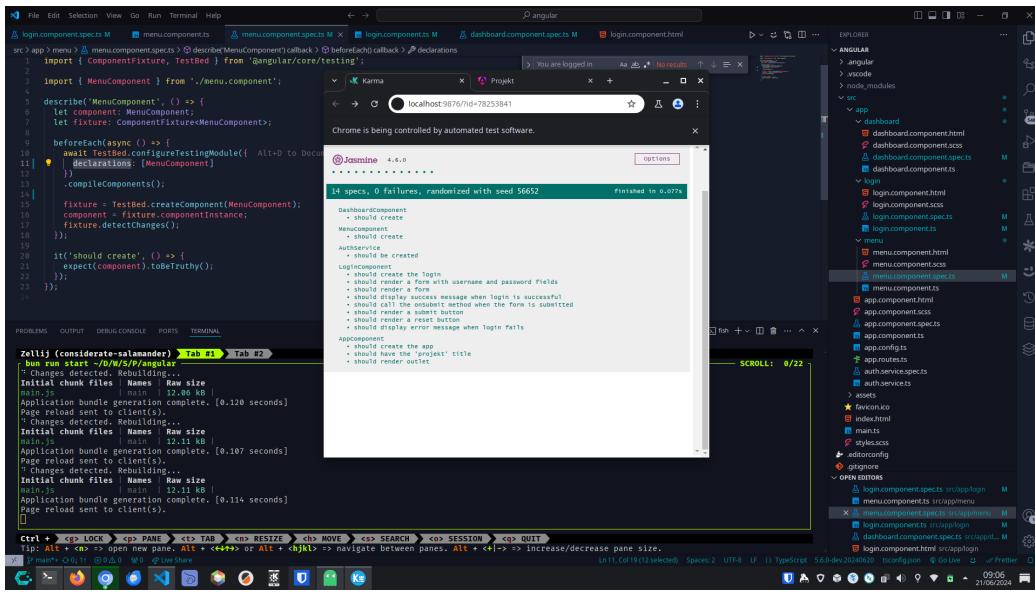


Figure 9: Wyniki testowania

1.3 Dodawanie dashboardu

- Utworzenie komponentu dashboard
- Utworzenie routingu
- Utworzenie komponentu menu
- Wykorzystanie wcześniej utworzonej usługi auth
- Utworzenie testów dla komponentu dashboard

1.3.1 Utworzenie komponentu dashboard



```
1 import { Component } from '@angular/core';
2 import { AuthService } from '../auth.service';
3 import { Router } from '@angular/router';
4 import { MenuComponent } from '../menu/menu.component';
5 import { CommonModule } from '@angular/common';
6
7 @Component({
8   selector: 'app-dashboard',
9   standalone: true,
10  templateUrl: './dashboard.component.html',
11  styleUrls: ['./dashboard.component.scss'],
12  imports: [MenuComponent, CommonModule],
13})
14 export class DashboardComponent {
15   login(arg0: string, arg1: string) {
16     throw new Error('Method not implemented.');
17   }
18   constructor(public authService: AuthService, private router: Router) {}
19
20   logout() {
21     this.authService.logout();
22     this.router.navigate(['/login']);
23   }
24 }
```

Figure 10: Implementacja dashboard.component.ts

```
1  <!-- dashboard.component.html -->
2  <div *ngIf="authService.isLoggedIn">
3      <app-menu></app-menu>
4      <p>dashboard content</p>
5  </div>
```

Figure 11: Implementacja dashboard.component.html

1.3.2 Utworzenie routingu

```
1 import { Routes } from '@angular/router';
2 import { LoginComponent } from './login/login.component';
3 import { DashboardComponent } from './dashboard/dashboard.component';
4
5 export const routes: Routes = [
6     { path: 'login', component: LoginComponent },
7     { path: 'dashboard', component: DashboardComponent },
8     { path: '', redirectTo: '/login', pathMatch: 'full' }
9 ];
```

Figure 12: Implementacja app.routes.ts

1.3.3 Utworzenie komponentu menu

Wykorzystanie do tego przygotowanej wcześniej usługi auth.

```
1 import { Component } from '@angular/core';
2 import { AuthService } from '../auth.service';
3 import { Router } from '@angular/router';
4
5 @Component({
6   selector: 'app-menu',
7   standalone: true,
8   templateUrl: './menu.component.html',
9   styleUrls: ['./menu.component.scss']
10 })
11 export class MenuComponent {
12   constructor(private authService: AuthService, private router: Router) {}
13
14   logout() {
15     this.authService.logout();
16     this.router.navigate(['/login']);
17   }
18 }
```

Figure 13: Implementacja menu.component.ts

```
1 <nav style="border-bottom: 1px solid black">
2   <h1>Menu</h1>
3   <ul style="list-style-type: none; display: flex">
4     <li style="margin-right: 10px">
5       <a
6         routerLink="/dashboard"
7         style="
8           text-decoration: none;
9           background-color: #f0f0f0;
10          padding: 5px 10px;
11          border-radius: 5px;
12          cursor: pointer;
13          "
14         >Dashboard</a>
15       >
16     </li>
17     <li>
18       <a
19         (click)="logout()"
20         style="
21           text-decoration: none;
22           background-color: #f0f0f0;
23           padding: 5px 10px;
24           border-radius: 5px;
25           cursor: pointer;
26           "
27         >Logout</a>
28       >
29     </li>
30   </ul>
31 </nav>
```

Figure 14: Implementacja menu.component.html

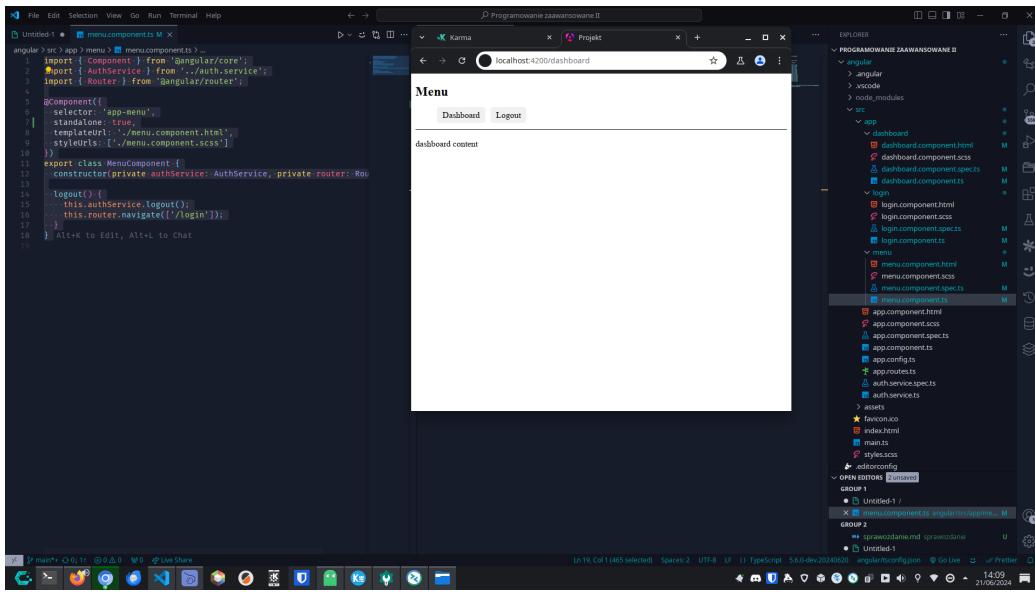


Figure 15: Prezentacja działających komponentów

1.3.4 Utworzenie testów dla komponentu dashboard

Wykorzystanie dostarczonych przez prowadzącego propozycji kodu w opracowaniu własnej implementacji.

```
1 import { ComponentFixture, TestBed } from '@angular/core/testing';
2 import { DashboardComponent } from './dashboard.component';
3 import { MenuComponent } from '../menu/menu.component';
4 import { AuthService } from '../auth.service';
5 import { Router } from '@angular/router';
6 describe('DashboardComponent', () => {
7   let component: DashboardComponent;
8   let fixture: ComponentFixture<DashboardComponent>;
9   let authService: AuthService;
10  let router: Router;
11
12  beforeEach(() => {
13    TestBed.configureTestingModule({
14      imports: [DashboardComponent, MenuComponent],
15      providers: [
16        AuthService,
17        {
18          provide: Router,
19          useValue: {
20            navigate: jasmine.createSpy('navigate'),
21          },
22        },
23      ],
24    });
25
26    fixture = TestBed.createComponent(DashboardComponent);
27    component = fixture.componentInstance;
28    authService = TestBed.inject(AuthService);
29    router = TestBed.inject(Router);
30  });
31  it('should create', () => {
32    expect(component).toBeTruthy();
33  });
34  it('should display the menu after logging in', () => {
35    authService.login('admin', 'admin');
36    fixture.detectChanges();
37
38    const menu = fixture.debugElement.nativeElement.querySelector('app-menu');
39    expect(menu).toBeTruthy();
40  });
41  it('should hide the menu and redirect to login after logging out', () => {
42    spyOn(authService, 'logout').and.callThrough();
43    fixture.detectChanges();
44
45    component.logout();
46    fixture.detectChanges();
47
48    const menu = fixture.debugElement.nativeElement.querySelector('app-menu');
49    expect(menu).toBeNull();
50    expect(router.navigate).toHaveBeenCalledWith(['/login']);
51  });
52});
```

Figure 16: Implementacja dashboard.component.spec.ts

Poniższe zdjęcie przedstawia różnice pomiędzy proponowaną implementacją testu komponentu `dashboard.component.spec.ts` a ostateczną wersją. Problematyczny okazał się ostatni test, w którym sprawdzano, czy komponent nie jest renderowany. W proponowanym podejściu test ten nie przechodził.

```
1 import { ComponentFixture, Testbed } from '@angular/core/testing';
2 import { DashboardComponent } from './dashboard.component';
3 import { RouterTestingModule } from '@angular/router/testing';
4 import { AuthService } from '../auth.service';
5 import { Router } from '@angular/router';
6
7 declare const expect;
8 let component: DashboardComponent;
9 let fixture: ComponentFixture<DashboardComponent>;
10 let authService: AuthService;
11 let router: Router;
12
13 beforeAll(() => {
14   Testbed.configureTestingModule({
15     imports: [DashboardComponent, MenuComponent],
16     providers: [
17       AuthService,
18       { provide: Router, useValue: jasmine.createSpy('navigate') }
19     ],
20     declarations: [DashboardComponent, MenuComponent]
21   });
22 });
23
24 it('should create', () => {
25   fixture = Testbed.createComponent(DashboardComponent);
26   component = fixture.componentInstance;
27   authService = Testbed.inject(AuthService);
28   router = Testbed.inject(Router);
29 });
30
31 it('should create', () => {
32   component.detectChanges();
33 });
34
35 it('should display the menu after logging in', () => {
36   authService.login('admin', 'admin');
37   fixture.detectChanges();
38 });
39
40 const menu = fixture.debugElement.nativeElement.querySelector('app-menu');
41 expect(menu).toBeTruthy();
42
43 it('should hide the menu and redirect to login after logging out', () => {
44   authService.logout();
45   fixture.detectChanges();
46 });
47
48 const menu = fixture.debugElement.nativeElement.querySelector('app-menu');
49 expect(menu).toBeFalsy();
50
51 it('should detect changes', () => {
52   component.logout();
53   fixture.detectChanges();
54 });
55
56 const menu = fixture.debugElement.nativeElement.querySelector('app-menu');
57 expect(menu).toBeFalsy();
58
59 it('should navigate to login', () => {
60   expect(router.navigate).toHaveBeenCalledWith(['/login']);
61 });
62 });
63
64
65 import { ComponentFixture, Testbed } from '@angular/core/testing';
66 import { DashboardComponent } from './dashboard.component';
67 import { RouterTestingModule } from '@angular/router/testing';
68 import { AuthService } from '../auth.service';
69 import { RouterTestingModule } from '@angular/router/testing';
70
71 declare const expect;
72 let component: DashboardComponent;
73 let fixture: ComponentFixture<DashboardComponent>;
74 let authService: AuthService;
75 let router: Router;
76
77 beforeAll(() => {
78   Testbed.configureTestingModule({
79     declarations: [DashboardComponent, MenuComponent],
80     providers: [AuthService],
81     imports: [RouterTestingModule]
82   });
83 });
84
85 it('should create', () => {
86   fixture = Testbed.createComponent(DashboardComponent);
87   component = fixture.componentInstance;
88   authService = Testbed.inject(AuthService);
89   router = Testbed.inject(Router);
90 });
91
92 it('should display the menu after logging in', () => {
93   authService.login('admin', 'admin');
94   fixture.detectChanges();
95 });
96
97 const menu = fixture.debugElement.nativeElement.querySelector('app-menu');
98 expect(menu).toBeTruthy();
99
100 it('should hide the menu and redirect to login after logging out', () => {
101   authService.logout();
102   fixture.detectChanges();
103 });
104
105 const menu = fixture.debugElement.nativeElement.querySelector('app-menu');
106 expect(menu).toBeFalsy();
107
108 it('should detect changes', () => {
109   component.logout();
110   fixture.detectChanges();
111 });
112
113 const menu = fixture.debugElement.nativeElement.querySelector('app-menu');
114 expect(menu).toBeFalsy();
115
116 it('should navigate to login', () => {
117   expect(router.navigate).toHaveBeenCalledWith(['/login']);
118 });
119 });
120 
```

Figure 17: Porównanie konfiguracji testów komponentu dashboard.component.spec.ts

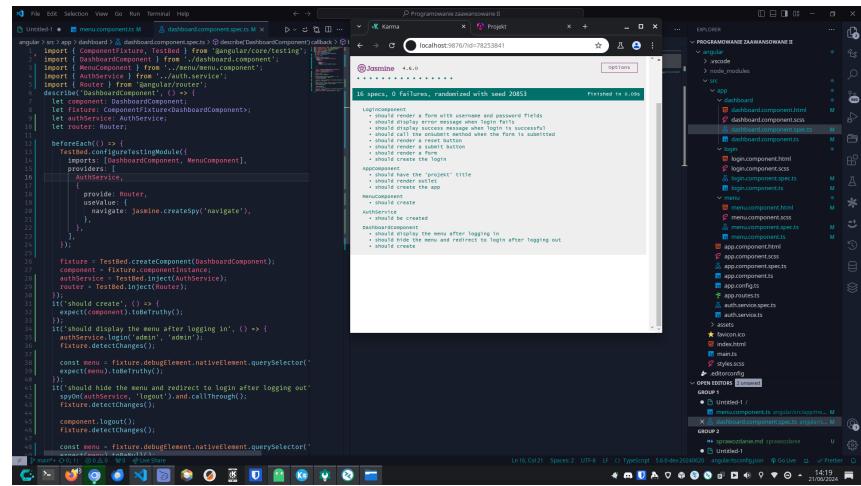
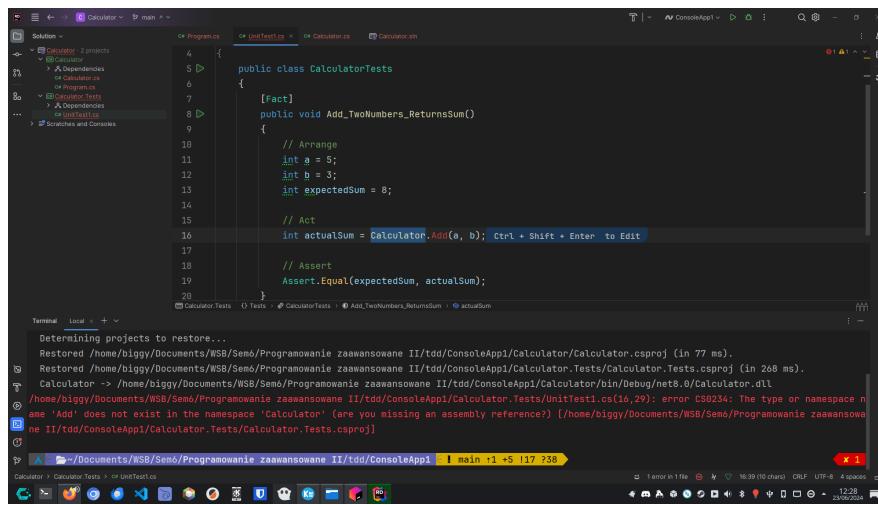


Figure 18: Pokrycie testów Karma

2 Tworzenie testów automatycznych systemu w podejściu TDD



The screenshot shows the Visual Studio Code interface with a C# project named 'Calculator'. The 'CalculatorTests' file contains a single test method:

```
public class CalculatorTests
{
    [Fact]
    public void Add_TwoNumbers_ReturnsSum()
    {
        // Arrange
        int a = 5;
        int b = 3;
        int expectedSum = 8;

        // Act
        int actualSum = Calculator.Add(a, b); Ctrl + Shift + Enter to Edit

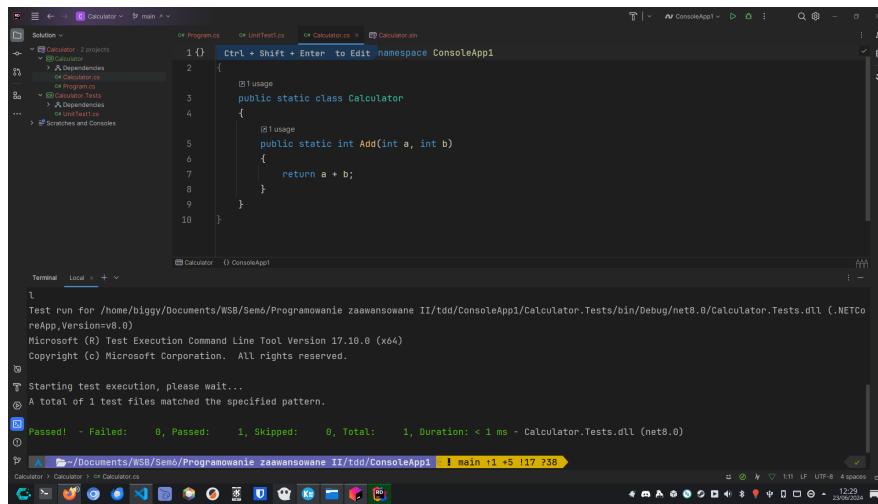
        // Assert
        Assert.Equal(expectedSum, actualSum);
    }
}
```

The terminal window shows the output of the test run:

```
Determining projects to restore...
Restored /home/biggy/Documents/WSB/Sem6/Programowanie zaawansowane II/tdd/ConsoleApp1/Calculator/Calculator.csproj (in 77 ms).
Restored /home/biggy/Documents/WSB/Sem6/Programowanie zaawansowane II/tdd/ConsoleApp1/Calculator.Tests.csproj (in 268 ms).
Calculator -> /home/biggy/Documents/WSB/Sem6/Programowanie zaawansowane II/tdd/ConsoleApp1/Calculator/bin/Debug/net8.0/calculator.dll
/home/biggy/Documents/WSB/Sem6/Programowanie zaawansowane II/tdd/ConsoleApp1/calculator.Tests/UnitTests.cs(16,29): error CS0234: The type or namespace name 'Add' does not exist in the namespace 'Calculator' (are you missing an assembly reference?) (/home/biggy/Documents/WSB/Sem6/Programowanie zaawansowane II/tdd/ConsoleApp1/calculator.Tests/calculator.Tests.csproj)
Calculator Tests />/Documents/WSB/Sem6/Programowanie zaawansowane II/tdd/ConsoleApp1 | main :1 +5 !17 ?38
```

A red box highlights the error message in the terminal: "error CS0234: The type or namespace name 'Add' does not exist in the namespace 'Calculator' (are you missing an assembly reference?)".

Figure 19: Fail testu, brak funkcjonalności w testowanej klasie



The screenshot shows the Visual Studio Code interface with the same 'Calculator' project. The 'CalculatorTests' file now contains the 'Add' method definition:

```
public static class Calculator
{
    public static int Add(int a, int b)
    {
        return a + b;
    }
}
```

The terminal window shows the output of the test run:

```
Test run for /home/biggy/Documents/WSB/Sem6/Programowanie zaawansowane II/tdd/ConsoleApp1/Calculator.Tests/bin/Debug/net8.0/Calculator.Tests.dll (.NET6.0-relApp, Version=v8.0)
Microsoft (R) Test Execution Command Line Tool Version 17.10.0 (x64)
Copyright (c) Microsoft Corporation. All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed! - Failed: 0, Passed: 1, Skipped: 0, Total: 1, Duration: < 1 ms - Calculator.Tests.dll (net8.0)

Calculator Tests />/Documents/WSB/Sem6/Programowanie zaawansowane II/tdd/ConsoleApp1 | main :1 +5 !17 ?38
```

A green box highlights the 'Passed!' message in the terminal.

Figure 20: Dodanie oczekiwanej metody do tesowanej klasy

Przedstawiony kod został napisany na tyle prosto i przejrzystie że nie wymaga refaktoryzacji.

```
using Xunit;

namespace Calculator.Tests {
    public class CalculatorTests {
        [Fact]
        public void Add_TwoNumbers_ReturnsSum() {
            // Arrange
            int a = 5;
            int b = 3;
            int expectedSum = 8;
            // Act
            int actualSum = Calculator.Add(a, b);
            // Assert
            Assert.Equal(expectedSum, actualSum);
        }
    }
}

namespace Calculator {
    public static class Calculator {
        public static int Add(int a, int b) {
            return a + b;
        }
    }
}
```

3 Testowania testów automatycznych systemu w podejściu BDD

Ze względu na brak kompatybilności SpecFlow z Visual Studio 2022, zastosowano alternatywne narzędzie Reqrnroll. Proces użytkowania Reqrnroll okazał się bardzo zbliżony do SpecFlow. Mimo że problem ten znaczowo utrudnił realizację zadania, ostatecznie udało się je pomyślnie ukończyć. *Na potrzeby zadania użyłem środowiska Windows'owego*

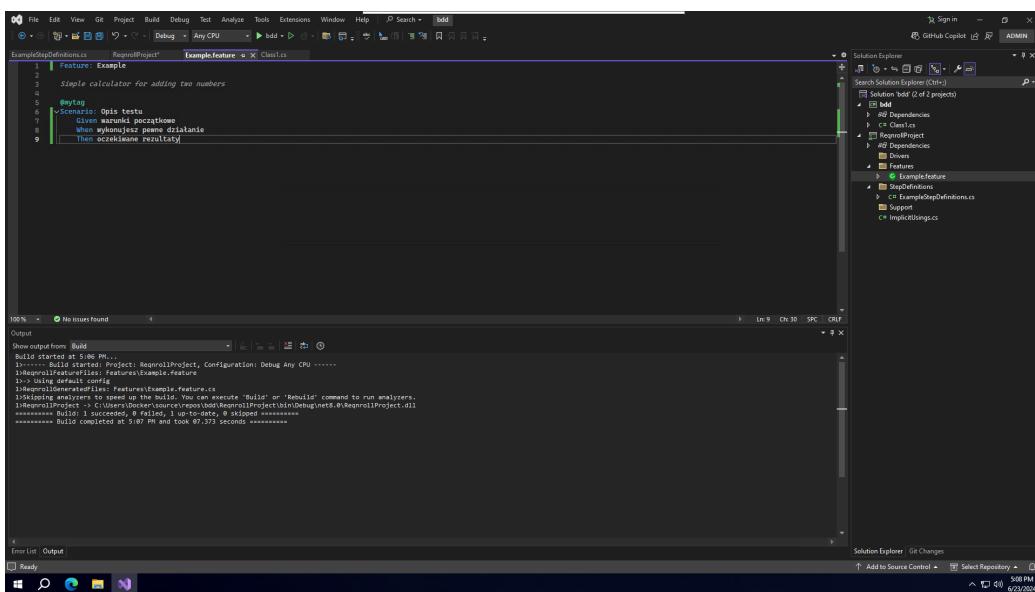


Figure 21: Utworzenie przykładowego feature'a

```

using System;
using Reprull;

namespace RegrullProject.StepDefinitions
{
    [Binding]
    public class ExampleStepDefinitions
    {
        [Given("marunki poczatekowe")]
        public void GivenMarunkiPoczatekowe()
        {
            throw new PendingStepException();
        }

        [When("jakiś kojarzy pełne działanie")]
        public void WhenJakiśKojarzyPełneDziałanie()
        {
            throw new PendingStepException();
        }

        [Then("oczekiwane rezultaty")]
        public void ThenOczekiwaneRezultaty()
        {
            throw new PendingStepException();
        }
    }
}

```

Figure 22: Wygenerowanie szkieletu kodu testowego

Figure 23: Rezultat uruchomienia testów po skonfigurowaniu środowiska

4 Testowanie pojedynczego modułu, który jest zależny od kodu zewnętrznego.

Zastosowanie techniki symulowania zależności za pomocą biblioteki Moq w języku C# umożliwia testowanie kodu korzystającego z zewnętrznych komponentów, takich jak bazy danych czy usługi sieciowe.

Implementacja klas Calculator.cs oraz IWebService.cs

```
namespace CalculatorTests;
public interface IWebService {
    void SendData(string data);
}

namespace CalculatorTests;
public class Calculator {
    private readonly IWebService _webService;
    public Calculator(IWebService webService) {
        _webService = webService;
    }
    public int Add(int a, int b) {
        int result = a + b;
        _webService.SendData($"Add operation: {a} + {b} = {result}");
        return result;
    }
}
```

Implementacja kodu testów

```
using Xunit;
using Moq;

namespace CalculatorTests {
    public class CalculatorTests {
        [Fact]
        public void Add_ShouldSendDataToWebService() {
            // Arrange
            var webServiceMock = new Mock<IWebService>();
            var calculator = new Calculator(webServiceMock.Object);
            // Act
            int result = calculator.Add(3, 5);
            // Assert
            Assert.Equal(8, result);
            webServiceMock.Verify(ws => ws.SendData("Add operation: 3 + 5 = 8"),
                Times.Once);
        }
    }
}
```

The screenshot shows a Visual Studio interface with the following details:

- Solution Explorer:** Shows the project structure with files like UnitTest1.cs, WebService.cs, and Calculator.cs.
- Code Editor:** Displays the code for the unit test class `CalculatorTests`. The current method is `Add_ShouldSendDataToWebService`, which uses `Xunit` and `Moq` to verify that the `calculator.Add(3, 5)` call results in 8 and triggers a service call to `ws.SendData("Add operation: 3 + 5 = 8")`.
- Unit Test Explorer:** Shows the test results for the `Add_ShouldSendDataToWebService` test, indicating it was successful.
- Status Bar:** Shows the date and time as 19.09.2024 11:23.

Figure 24: Wynik uruchomienia testów

5 Weryfikowanie czy moduły zależne współpracują między sobą w sposób poprawny.

Wykorzystanie biblioteki Moq w celu sprawdzenie, czy OrderProcessor poprawnie korzysta z IOrderService podczas przetwarzania zamówienia, czyli testowaniu interakcji między dwoma modułami.

To podejście pozwala na dokładne przetestowanie współpracy między modułami, zapewniając jednocześnie izolację i powtarzalność testów jednostkowych.

Wykorzystanie projektu z poprzedniego zadania

Implementacja wymaganych klas OrderProcessor oraz IOrderService

```
namespace CalculatorTests;
public interface IOrderService {
    void PlaceOrder(string product, int quantity);
}

namespace CalculatorTests;
public class OrderProcessor {
    private readonly IOrderService _orderService;
    public OrderProcessor(IOrderService orderService) {
        _orderService = orderService;
    }
    public void ProcessOrder(string product, int quantity) {
        _orderService.PlaceOrder(product, quantity);
    }
}
```

Implementacja kodu testów

```
public class OrderProcessorTests {
    [Fact]
    public void ProcessOrder_ShouldCallPlaceOrderWithCorrectParameters() {
        // Arrange
        var orderServiceMock = new Mock<IOrderService>();
        var orderProcessor = new OrderProcessor(orderServiceMock.Object);
        // Act
        orderProcessor.ProcessOrder("ProductABC", 3);
        // Assert
        orderServiceMock.Verify(os => os.PlaceOrder("ProductABC", 3), Times.Once);
    }
    [Fact]
    public void ProcessOrder_ShouldCallPlaceOrderMultipleTimes() {
        // Arrange
        var orderServiceMock = new Mock<IOrderService>();
        var orderProcessor = new OrderProcessor(orderServiceMock.Object);
        // Act
        orderProcessor.ProcessOrder("Product1", 2);
        orderProcessor.ProcessOrder("Product2", 5);
        // Assert
        orderServiceMock.Verify(os => os.PlaceOrder("Product1", 2), Times.Once);
        orderServiceMock.Verify(os => os.PlaceOrder("Product2", 5), Times.Once);
    }
}
```

The screenshot shows the Visual Studio IDE interface. The top part displays the code for the `OrderProcessorTests` class, specifically the `ProcessOrder_ShouldCallPlaceOrderWithCorrectParameters` test. The bottom part shows the Test Explorer window with the following results:

- `CalculatorTests / [test] Success`
- `CalculatorTests / [test] Success`
- `CalculatorTests / [test] Success`
- `Add_ShouldSendDataToWebService Success` (highlighted with a red border)
- `OrderProcessorTests / [test] Success`
 - `ProcessOrder_ShouldCallPlaceOrderMultipleTimes Success`
 - `ProcessOrder_ShouldCallPlaceOrderWithCorrectParameters Success`

Figure 25: Wynik uruchomienia testów