PROJEKT

1. Wymagania funkcjonalne i niefunkcjonalne:

**Wymagania funkcjonalne:**

1. **Rejestracja i logowanie użytkowników**:
   * Możliwość rejestracji nowych użytkowników.
   * Logowanie użytkowników.
   * Autoryzacja dostępu do funkcji aplikacji.
2. **Zarządzanie fakturami**:
   * Dodawanie nowych faktur.
   * Edytowanie istniejących faktur.
   * Usuwanie faktur.
   * Przeglądanie listy faktur.
3. **Zarządzanie kontrahentami**:
   * Dodawanie nowych kontrahentów.
   * Edytowanie danych kontrahentów.
   * Usuwanie kontrahentów.
   * Przeglądanie listy kontrahentów.
4. **Podsumowanie finansowe**:
   * Wyświetlanie podsumowania przychodów i wydatków.
   * Generowanie raportów finansowych.

**Wymagania niefunkcjonalne:**

1. **Wydajność**:
   * Aplikacja powinna obsługiwać jednocześnie co najmniej 100 użytkowników bez zauważalnego spadku wydajności.
2. **Bezpieczeństwo**:
   * Dane użytkowników muszą być szyfrowane.
   * Mechanizmy autoryzacji muszą być bezpieczne przed atakami typu CSRF i XSS.
3. **Niezawodność**:
   * Błędy krytyczne powinny być automatycznie zgłaszane do administratorów.
4. **Użyteczność**:
   * Interfejs użytkownika powinien być intuicyjny i łatwy w nawigacji.
   * Aplikacja powinna być responsywna i działać poprawnie na urządzeniach mobilnych.
5. Diagram klas

A close-up of a document

Description automatically generated

**Relacje pomiędzy tabelami:**

* Tabela **Faktury** posiada dwa klucze obce: **ID Wystawiający**, który odnosi się do tabeli **User**, oraz **ID Podmiot**, który odnosi się do tabeli **Podmioty**. Relacje te są oznaczone odpowiednimi strzałkami w diagramie.

**Opis sposobów i metod testowania**

**Testowanie jednostkowe**

* **Cel:** Sprawdzenie poprawności działania pojedynczych jednostek kodu (funkcji, metod).
* **Narzędzia:** Jest (JavaScript Testing Framework).
* **Przykłady:**
  + Testowanie funkcji logowania użytkownika.
  + Testowanie dodawania nowych faktur.

**Problemy:**

**Problem:** Funkcja dodawania faktur nie waliduje poprawności daty faktury.

* + **Opis:** Testy wykazały, że możliwe było dodanie faktury z datą w przeszłości lub przyszłości, która była niezgodna z przyjętymi zasadami biznesowymi.
  + **Rozwiązanie:** Dodanie walidacji daty w funkcji dodawania faktur i ponowne uruchomienie testów.

**Testowanie integracyjne**

* **Cel:** Weryfikacja współpracy między różnymi modułami aplikacji.
* **Narzędzia:** Jest, Supertest.
* **Przykłady:**
  + Testowanie interakcji między modułem autoryzacji a modułem zarządzania fakturami.

**Testowanie end-to-end (E2E)**

* **Cel:** Symulowanie rzeczywistych scenariuszy użytkowania aplikacji.
* **Narzędzia:** Cypress.
* **Przykłady:**
  + Testowanie całego procesu od logowania po dodanie nowej faktury i wyświetlenie podsumowania finansowego.

**Problemy:**

**Problem:** Niezgodność interfejsu użytkownika na różnych przeglądarkach.

* + **Opis:** Testy E2E wykazały, że interfejs użytkownika wyglądał inaczej na różnych przeglądarkach, co mogło wprowadzać użytkowników w błąd.
  + **Rozwiązanie:** Poprawienie stylów CSS oraz testowanie interfejsu na różnych przeglądarkach.

**IMPLEMENTACJA**

### Warstwa Logiki Biznesowej

#### Opis Projektu

Repozytorium zawiera aplikację webową do zarządzania fakturami, zobowiązaniami i użytkownikami. Warstwa logiki biznesowej jest odpowiedzialna za przetwarzanie danych i egzekwowanie reguł biznesowych pomiędzy warstwą prezentacji (frontend) a warstwą dostępu do danych (backend). Poniżej znajduje się szczegółowy opis tej warstwy.

#### Struktura Katalogów

* src/app/
  + shared/
    - auth.service.ts
    - dueDate.pipe.ts
    - guards.guard.ts
    - interfaces.ts
    - invoices.service.ts
    - vendor.service.ts
  + login-form/
    - login-form.component.ts
  + invoices-summary/
    - invoices-summary.component.ts
  + obligation-summary/
    - obligation-summary.component.ts

### Opis Plików

#### auth.service.ts

**Funkcja**: Zapewnia funkcje autoryzacji i uwierzytelniania dla użytkowników.

* **Metody**:
  + login(username: string, password: string): Loguje użytkownika.
  + logout(): Wylogowuje użytkownika.
  + isAuthenticated(): Sprawdza, czy użytkownik jest zalogowany.

#### dueDate.pipe.ts

**Funkcja**: Dostarcza funkcjonalność do formatowania daty zgodnie z wymogami biznesowymi.

* **Metody**:
  + transform(value: Date): Formatuje datę na określony format.

#### guards.guard.ts

**Funkcja**: Zabezpiecza ścieżki routingu w aplikacji.

* **Metody**:
  + canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Sprawdza, czy użytkownik ma prawo do dostępu do określonej ścieżki.

#### interfaces.ts

**Funkcja**: Definiuje interfejsy używane w aplikacji.

* **Interfejsy**:
  + User: Definiuje strukturę obiektu użytkownika.
  + Invoice: Definiuje strukturę obiektu faktury.
  + Vendor: Definiuje strukturę obiektu dostawcy.

#### invoices.service.ts

**Funkcja**: Zarządza operacjami związanymi z fakturami.

* **Metody**:
  + getInvoices(): Pobiera listę faktur.
  + addInvoice(invoice: Invoice): Dodaje nową fakturę.
  + deleteInvoice(id: number): Usuwa fakturę o określonym ID.

#### vendor.service.ts

**Funkcja**: Zarządza operacjami związanymi z dostawcami.

* **Metody**:
  + getVendors(): Pobiera listę dostawców.
  + addVendor(vendor: Vendor): Dodaje nowego dostawcę.
  + deleteVendor(id: number): Usuwa dostawcę o określonym ID.

### Przykłady Użycia

#### Logowanie Użytkownika

**import { AuthService } from './shared/auth.service';**

**constructor(private authService: AuthService) {}**

**login() {**

**this.authService.login(this.username, this.password).subscribe(response => {**

**// Obsługa odpowiedzi**

**});**

**}**

Pobieranie Faktur

**import { InvoicesService } from './shared/invoices.service';**

**constructor(private invoicesService: InvoicesService) {}**

**ngOnInit() {**

**this.invoicesService.getInvoices().subscribe(invoices => {**

**this.invoices = invoices;**

**});**

**}**

Zabezpieczenie Routingu

**import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';**

**import { AuthService } from './shared/auth.service';**

**@Injectable({**

**providedIn: 'root'**

**})**

**export class AuthGuard implements CanActivate {**

**constructor(private authService: AuthService, private router: Router) {}**

**canActivate(**

**next: ActivatedRouteSnapshot,**

**state: RouterStateSnapshot): boolean {**

**if (this.authService.isAuthenticated()) {**

**return true;**

**} else {**

**this.router.navigate(['/login']);**

**return false;**

**}**

**}**

**}**

Implementacja w tym projekcie obejmuje usługi zarządzające fakturami, dostawcami oraz funkcjonalności uwierzytelniania i autoryzacji użytkowników.

### Warstwa Prezentacji

#### Opis Projektu

Warstwa prezentacji odpowiada za interakcję użytkownika z aplikacją oraz wyświetlanie danych. Korzysta z komponentów Angular do budowy interfejsu użytkownika. Poniżej znajduje się szczegółowy opis tej warstwy.

#### Struktura Katalogów

* src/app/
  + login-form/
    - login-form.component.ts
    - login-form.component.html
    - login-form.component.css
  + invoices-summary/
    - invoices-summary.component.ts
    - invoices-summary.component.html
    - invoices-summary.component.css
  + obligation-summary/
    - obligation-summary.component.ts
    - obligation-summary.component.html
    - obligation-summary.component.css

### Opis Plików

#### login-form/

**Funkcja**: Komponent logowania użytkownika. Odpowiada za interfejs do wprowadzania danych logowania i uwierzytelniania użytkownika.

**Pliki**:

* login-form.component.ts
  + **Opis**: Klasa komponentu logowania.
  + **Metody**:
    - onSubmit(): Obsługuje proces logowania po przesłaniu formularza.
* login-form.component.html
  + **Opis**: Szablon HTML dla formularza logowania.
  + **Struktura**:
    - Pole wprowadzania dla nazwy użytkownika.
    - Pole wprowadzania dla hasła.
    - Przycisk przesyłający formularz.
* login-form.component.css
  + **Opis**: Stylizacja CSS dla formularza logowania.
  + **Zawartość**:
    - Stylizacja pól wprowadzania.
    - Stylizacja przycisku logowania.

#### invoices-summary/

**Funkcja**: Komponent podsumowania faktur. Wyświetla listę faktur oraz umożliwia ich dodawanie i usuwanie.

**Pliki**:

* invoices-summary.component.ts
  + **Opis**: Klasa komponentu podsumowania faktur.
  + **Metody**:
    - ngOnInit(): Pobiera listę faktur przy inicjalizacji komponentu.
    - deleteInvoice(id: number): Usuwa fakturę o określonym ID.
* invoices-summary.component.html
  + **Opis**: Szablon HTML dla podsumowania faktur.
  + **Struktura**:
    - Tabela wyświetlająca listę faktur.
    - Przycisk do dodawania nowych faktur.
    - Przycisk do usuwania faktur.
* invoices-summary.component.css
  + **Opis**: Stylizacja CSS dla podsumowania faktur.
  + **Zawartość**:
    - Stylizacja tabeli faktur.
    - Stylizacja przycisków.

#### obligation-summary/

**Funkcja**: Komponent podsumowania zobowiązań. Wyświetla listę zobowiązań oraz umożliwia ich dodawanie i usuwanie.

**Pliki**:

* obligation-summary.component.ts
  + **Opis**: Klasa komponentu podsumowania zobowiązań.
  + **Metody**:
    - ngOnInit(): Pobiera listę zobowiązań przy inicjalizacji komponentu.
    - deleteObligation(id: number): Usuwa zobowiązanie o określonym ID.
* obligation-summary.component.html
  + **Opis**: Szablon HTML dla podsumowania zobowiązań.
  + **Struktura**:
    - Tabela wyświetlająca listę zobowiązań.
    - Przycisk do dodawania nowych zobowiązań.
    - Przycisk do usuwania zobowiązań.
* obligation-summary.component.css
  + **Opis**: Stylizacja CSS dla podsumowania zobowiązań.
  + **Zawartość**:
    - Stylizacja tabeli zobowiązań.
    - Stylizacja przycisków.

### Przykłady Użycia

#### Formularz Logowania

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

**import { AuthService } from '../shared/auth.service';**

**@Component({**

**selector: 'app-login-form',**

**templateUrl: './login-form.component.html',**

**styleUrls: ['./login-form.component.css']**

**})**

**export class LoginFormComponent {**

**username: string = '';**

**password: string = '';**

**constructor(private authService: AuthService) {}**

**onSubmit() {**

**this.authService.login(this.username, this.password).subscribe(response => {**

**// Obsługa odpowiedzi**

**});**

**}**

**}**

Podsumowanie Faktur

**import { Component, OnInit } from '@angular/core';**

**import { InvoicesService } from '../shared/invoices.service';**

**import { Invoice } from '../shared/interfaces';**

**@Component({**

**selector: 'app-invoices-summary',**

**templateUrl: './invoices-summary.component.html',**

**styleUrls: ['./invoices-summary.component.css']**

**})**

**export class InvoicesSummaryComponent implements OnInit {**

**invoices: Invoice[] = [];**

**constructor(private invoicesService: InvoicesService) {}**

**ngOnInit() {**

**this.invoicesService.getInvoices().subscribe(invoices => {**

**this.invoices = invoices;**

**});**

**}**

**deleteInvoice(id: number) {**

**this.invoicesService.deleteInvoice(id).subscribe(() => {**

**this.invoices = this.invoices.filter(invoice => invoice.id !== id);**

**});**

**}**

**}**

Warstwa prezentacji odpowiada za wyświetlanie danych i interakcję użytkownika z aplikacją. Komponenty Angular, takie jak LoginFormComponent, InvoicesSummaryComponent, i ObligationSummaryComponent, są kluczowe do zapewnienia użytkownikowi możliwości logowania, zarządzania fakturami i zobowiązaniami. Stylizacja CSS oraz szablony HTML dopełniają funkcjonalność, zapewniając przyjazny interfejs użytkownika.

**Diagram sekwencji**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie**

**Opis metod:**

**Metody logowania, wylogowywania i automatycznego logowania użytkownika oraz pobierania danych użytkownika**

**logIn(login: string, password: string): void**

**Opis:**  
Ta metoda loguje użytkownika poprzez ustawienie danych użytkownika i zapisanie ich w pamięci lokalnej. Aktualizuje obserwowalnego użytkownika za pomocą wstępnie zdefiniowanego obiektu użytkownika.

**Parametry:**

* **login (string):** Nazwa użytkownika.
* **password (string):** Hasło użytkownika (obecnie nieużywane w logice funkcji).

**Zachowanie:**

* Aktualizuje obserwowalnego użytkownika za pomocą wstępnie zdefiniowanego obiektu użytkownika zawierającego właściwości ID, albumId, name, surname, nickname i access.
* Jeśli obserwowalny użytkownik nie jest nullem, zapisuje obiekt użytkownika w pamięci lokalnej.

**Przykład użycia:**

logIn('exampleLogin', 'examplePassword');

**logOut(): void**

**Opis:**  
Ta metoda wylogowuje obecnego użytkownika, usuwając dane użytkownika zarówno z obserwowalnego, jak i z pamięci lokalnej.

**Zachowanie:**

* Ustawia obserwowalnego użytkownika na null.
* Usuwa obiekt użytkownika z pamięci lokalnej.

**Przykład użycia:**

logOut();

**autoLogOn(): void**

**Opis:**  
Ta metoda automatycznie loguje użytkownika, jeśli dane użytkownika są zapisane w pamięci lokalnej. Aktualizuje obserwowalnego użytkownika za pomocą przechowywanych danych użytkownika.

**Zachowanie:**

* Pobiera obiekt użytkownika z pamięci lokalnej.
* Jeśli brak danych użytkownika, ustawia obserwowalnego użytkownika na null.
* Jeśli dane użytkownika zostaną znalezione, analizuje je i aktualizuje obserwowalnego użytkownika.
* Loguje bieżącą wartość obserwowalnego użytkownika do konsoli.

**Przykład użycia:**

autoLogOn();

**getUser(): User**

**Opis:**  
Ta metoda zwraca bieżącą wartość obserwowalnego użytkownika.

**Zwraca:**

* Bieżący obiekt użytkownika, jeśli użytkownik jest zalogowany.
* null, jeśli żaden użytkownik nie jest zalogowany.

**Przykład użycia:**

const currentUser = getUser();

console.log(currentUser);

**Struktura obiektu użytkownika**

Gdy obiekt użytkownika jest ustawiany lub pobierany, ma następującą strukturę:

{

"ID": number,

"albumId": number,

"name": string,

"surname": string,

"nickname": string,

"access": number

}

**Właściwości:**

* **ID (number):** ID użytkownika.
* **albumId (number):** ID albumu przypisanego do użytkownika.
* **name (string):** Imię użytkownika.
* **surname (string):** Nazwisko użytkownika.
* **nickname (string):** Nick użytkownika.
* **access (number):** Poziom dostępu użytkownika.

**Przykładowe użycie**

const authService = new AuthService();

// Logowanie użytkownika

authService.logIn('userLogin', 'userPassword');

// Pobranie bieżącego użytkownika

const user = authService.getUser();

console.log(user);

// Wylogowanie bieżącego użytkownika

authService.logOut();

// Automatyczne logowanie użytkownika, jeśli dane istnieją w pamięci lokalnej

authService.autoLogOn();

**Angular Guard: stażnik trasy**

**guardsGuard**

* + guardsGuard jest funkcją pełniącą rolę strażnika trasy (route guard). Jej zadaniem jest kontrola dostępu do określonych ścieżek w aplikacji Angular. Funkcja ta jest definiowana jako stała (const) i implementuje interfejs CanActivateFn.

**Wstrzykiwanie zależności**

* + const authService = inject(AuthService);: Wstrzykuje instancję AuthService.
  + const router = inject(Router);: Wstrzykuje instancję Router.

**Pobranie aktualnego użytkownika**

* + const currentUser = authService.getUser();: Pobiera aktualnie zalogowanego użytkownika za pomocą metody getUser() z serwisu AuthService.

1. **Logika kontroli dostępu**
   * Funkcja sprawdza, czy currentUser nie jest null.
     + Jeśli currentUser jest różny od null (czyli użytkownik jest zalogowany), funkcja zwraca true, co oznacza, że dostęp do trasy jest dozwolony.
     + Jeśli currentUser jest null (czyli użytkownik nie jest zalogowany), funkcja wyświetla alert z komunikatem "Log in required!" i przekierowuje użytkownika na stronę logowania za pomocą router.navigate(['login']). Następnie funkcja zwraca false, co oznacza, że dostęp do trasy jest zablokowany.

**Przykład użycia**

Wstrzyknięcie serwisu AuthService

const authService = inject(AuthService);

 Wstrzykuje instancję serwisu AuthService, który jest odpowiedzialny za zarządzanie stanem logowania użytkownika.

 **Wstrzyknięcie Router**

const router = inject(Router);

 Wstrzykuje instancję Router, która umożliwia nawigację między różnymi stronami w aplikacji.

 **Sprawdzenie stanu logowania użytkownika**

const currentUser = authService.getUser();

 Pobiera aktualnie zalogowanego użytkownika z serwisu AuthService.

 **Logika kontroli dostępu**

if (currentUser !== null) {

return true;

} else {

alert('Log in required!');

router.navigate(['login']);

return false;

}

Funkcja sprawdza, czy użytkownik jest zalogowany. Jeśli tak, pozwala na dostęp do trasy. Jeśli nie, wyświetla alert i przekierowuje na stronę logowania.