Dokumentacja Projektu - Serwis Komputerowy

Skład grupy:
- Mateusz Mogielnicki
- Jakub Matyszek
- Przemysław Rutkowski

1. Architektura (drzewo) komponentów wraz z danymi przekazywanymi pomiędzy nimi.

Pliki html

1. AppComponent (app.component.html) - Jest to główny komponent aplikacji, który zawiera router-outlet i inne główne komponenty.

2. EditReportComponent (edit-report.component.html) - Komponent umożliwiający edycję raportów.

3. LoginComponent (login.component.html) - Komponent odpowiedzialny za logowanie użytkowników.

4. RegistrationComponent (registration.component.html) - Komponent umożliwiający rejestrację nowych użytkowników.

5. DisplayLogsComponent - Jest to komponent, w którym przesyłamy pobrane logi do: ShowLogsComponent - wykorzystujemy tutaj komunikację dwukierunkową pomiędzy komponentami.

6. ShowReportFormComponent - (show-report-form.component.html) Komponent do wyświetlania formularza raportu.

7. ShowReportsComponent - (show-reports.component.html) Komponent do wyświetlania raportów z różnymi opcjami sortowania i filtrowania. Warto zauważyć, że dane wykorzystywane w tym komponencie są pobierane za pomocą Angular services (np. logs.service.ts, reports.service.ts, user.service.ts).

Komponenty i Ich Zastosowanie

1. AppComponent (app.component.ts): Jest to główny komponent aplikacji, który pełni rolę kontenera dla innych komponentów oraz obsługuje główny layout i nawigację.

2. LoginComponent (login.component.ts): Odpowiada za logowanie użytkowników. Wykorzystuje user.service.ts do autentykacji i zarządzania sesją użytkownika.

3. RegistrationComponent (registration.component.ts): Umożliwia nowym użytkownikom rejestrację w systemie. Podobnie jak LoginComponent, korzysta z user.service.ts, który umożliwia zalogowanie się zarejestrowanym użytkownikom.

4. DisplayLogsComponent i ShowLogsComponent: Te komponenty mogą służyć do wyświetlania logów działań osób pracujących w serwisie komputerowym, korzystając z logs.service.ts.

5. EditReportComponent, ShowReportFormComponent, ShowReportsComponent: Związane z zarządzaniem raportami. Wykorzystują reports.service.ts do operacji na danych raportach.

Serwisy

1. UserService: Zarządza funkcjami użytkownika, takimi jak logowanie, wylogowanie, rejestracja.

2. ReportsService: Obsługuje operacje na raportach, takie jak pobieranie, edycja i usuwanie.

3. LogsService: Zarządza danymi logów aplikacji.

4. DiscountService : Serwis zarządzający ostateczną ceną usług wykonanych przez serwis.

Dyrektywy i Pipe'y

1. LogoDirective: Wykorzystujemy tą dyrektywę do logo naszego serwisu.

2. DatePriorityPipe: Używana do sortowania i filtracji raportów.

Modele Danych

1. Modele takie jak Log, Report, Client, User itp. służą jako definicje struktur danych wykorzystywanych w aplikacji. Są kluczowe dla zarządzania stanem i przepływu danych.

2. Routing Plik app-routing.module.ts definiuje ścieżki i przekierowania między różnymi komponentami. Jest kluczowy dla nawigacji i struktury URL w aplikacji.

2. Ścieżki i komponenty związane z routingiem.

1. Ścieżka główna ('/') Komponent: LoginComponent -> Główna ścieżka aplikacji, domyślnie ładuje komponent logowania.

2. Ścieżka '/registration' Komponent: RegistrationComponent -> Ścieżka do komponentu obsługującego rejestrację użytkowników.

3. Ścieżka '/reports' Komponent: ShowReportsComponent -> Ścieżka do komponentu wyświetlającego raporty.

4. Ścieżka '/reportform' Komponent: ShowReportFormComponent -> Ścieżka do formularza tworzenia nowego raportu.

5. Ścieżka '/edit-report/:id' Komponent: EditReportComponent -> Ścieżka do edycji raportu, zawiera parametr :id do identyfikacji konkretnego raportu.

6. Ścieżka '/showlogs' Komponent: ShowLogsComponent -> Ścieżka do komponentu służącego do wyświetlania logów aplikacji.

Podsumowanie:
Ścieżki i komponenty stanowią podstawę systemu nawigacji w serwisie komputerowym. Routing w Angularze pozwala na ładowanie różnych komponentów na podstawie adresu URL. Każda ścieżka odpowiada określonemu komponencie, który jest wyświetlany, gdy użytkownik nawiguje do danej ścieżki.

3. API serwera.

Ogólna Struktura Serwisów i API Serwisy Angulara: W Angularze serwisy są używane do zarządzania logiką biznesową i komunikacji z zewnętrznymi endpointami, takimi jak API serwera.

1. Typowe serwisy obejmują metody, które wykonują zapytania HTTP (GET, POST, PUT, DELETE) do określonych endpointów API.

2. HttpClient: Angular wykorzystuje HttpClient do wykonywania zapytań HTTP. Serwisy będą zawierać wstrzyknięty HttpClient i będą używać go do wysyłania zapytań do serwera.

3. Obsługa Zapytań HTTP: Przykładowo, serwis user.service.ts może zawierać metody do logowania, rejestracji i zarządzania danymi użytkownika, komunikując się z odpowiednimi endpointami API (np. /login, /register).

Modele Danych:

1. Modele danych, takie jak User.ts, Report.ts, itp., są używane do strukturyzowania danych wysyłanych do i odbieranych z API.

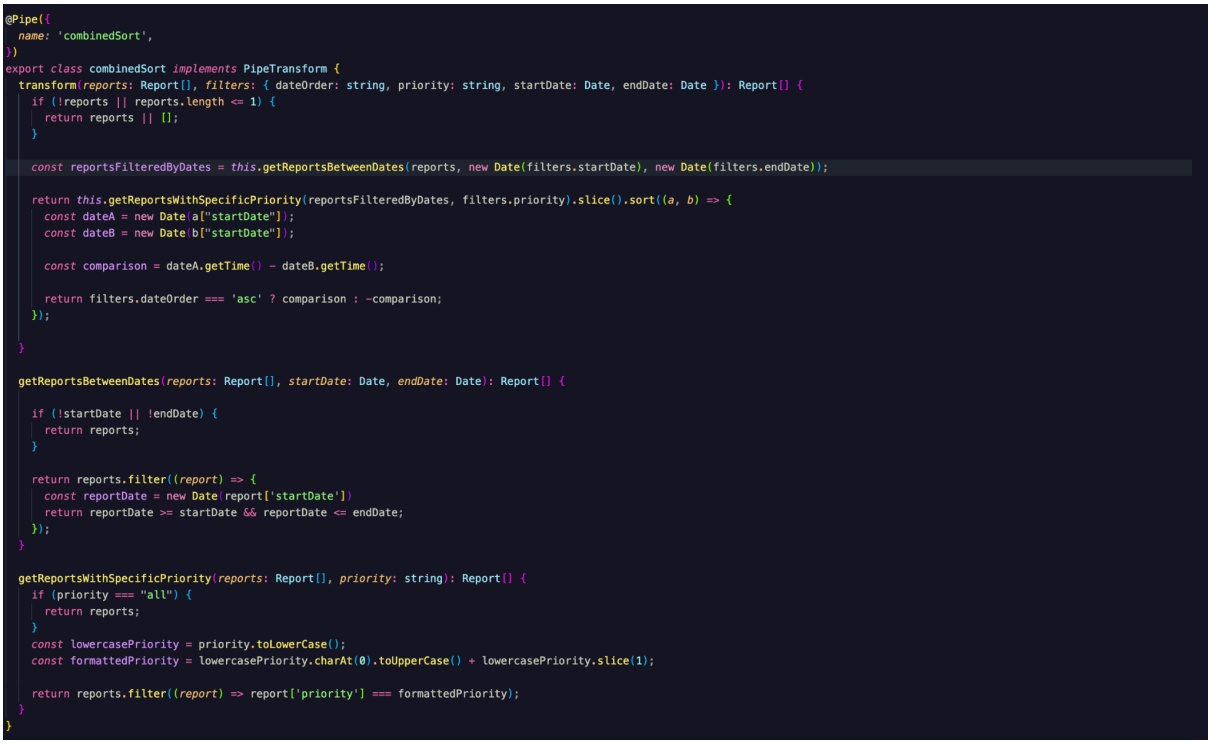
2. UserService: Metody: login(), register(), getUserDetails(). Endpointy: /api/login, /api/register, /api/user. Wykorzystuje HttpClient do wysyłania zapytań do API serwera.

3. ReportsService: Metody mogą obejmować: getReports(), createReport(), updateReport(), deleteReport(). Endpointy: /api/reports, /api/reports/{id}. Używa modelu Report.ts do strukturyzacji danych.

4. LogsService: Metody takie jak: getLogs(), getLogById(). Endpointy: /api/logs, /api/logs/{id}. Może zwracać dane w formacie określonym przez model Log.ts.

4. Wybrane przez autorów, szczególnie ciekawe fragmenty kodu.

1. DatePriorityPipe (datePriorityPipe.ts)
   Fragment Kodu:



```
@Pipe({
  name: 'combinedSort',
})
export class combinedSort implements PipeTransform {
  transform(reports: Report[], filters: { dateOrder: string, priority: string, startDate: Date, endDate: Date }): Report[] {
    if (!reports || reports.length <= 1) {
      return reports || [];
    }

    const reportsFilteredByDates = this.getReportsBetweenDates(reports, new Date(filters.startDate), new Date(filters.endDate));

    return this.getReportsWithSpecificPriority(reportsFilteredByDates, filters.priority).slice().sort((a, b) => {
      const dateA = new Date(a["startDate"]);
      const dateB = new Date(b["startDate"]);

      const comparison = dateA.getTime() - dateB.getTime();

      return filters.dateOrder === 'asc' ? comparison : -comparison;
    });

  }

  getReportsBetweenDates(reports: Report[], startDate: Date, endDate: Date): Report[] {

    if (!startDate || !endDate) {
      return reports;
    }

    return reports.filter((report) => {
      const reportDate = new Date(report['startDate'])
      return reportDate >= startDate && reportDate <= endDate;
    });
  }

  getReportsWithSpecificPriority(reports: Report[], priority: string): Report[] {
    if (priority === "all") {
      return reports;
    }
    const lowercasePriority = priority.toLowerCase();
    const formattedPriority = lowercasePriority.charAt(0).toUpperCase() + lowercasePriority.slice(1);

    return reports.filter((report) => report['priority'] === formattedPriority);
  }
}
```

Jest to definicja niestandardowego pipe'a w Angularze. Pipe'y są używane do transformacji danych wyświetlanych w szablonach komponentów. Ten konkretny pipe, DatePriorityPipe, służy do filtracji i sortowania raportów według dat lub priorytetów.

2. Dyrektywa Angulara
   Fragment Kodu:

```
@Directive({
  selector: '[LogoDirectiveDirective]',
})
export class LogoDirectiveDirective {
  private originalColor: string;
  private originalFontSize: string;
  private originalFontWeight: string;
  private originalFontFamily: string;
  private originalLetterSpacing: string;
  private isMouseOver: boolean = false;

  constructor(private el: ElementRef, private renderer: Renderer2) {
    this.originalColor = el.nativeElement.style.color || 'aliceblue';
    this.originalFontSize = el.nativeElement.style.fontSize || '22px';
    this.originalFontWeight = el.nativeElement.style.fontWeight || 'normal';
    this.originalLetterSpacing = el.nativeElement.style.letterSpacing || '2px';
    this.originalFontFamily = el.nativeElement.style.fontFamily || 'inherit';

    this.renderer.setStyle(el.nativeElement, 'transition', 'color 0.3s, font-size 0.3s, font-weight 0.3s, font-family 0.3s, letter-spacing 0.3s');
  }

  @HostListener('mouseenter') onMouseEnter() {
    if (!this.isMouseOver) {
      this.isMouseOver = true;
      this.changeStyles('aquamarine', '26px', 'bold', 'Poppins', '4px');
    }
  }

  @HostListener('mouseleave') onMouseLeave() {
    if (this.isMouseOver) {
      this.isMouseOver = false;
      this.changeStyles(this.originalColor, this.originalFontSize, this.originalFontWeight, this.originalFontFamily, this.originalLetterSpacing);
    }
  }

  private changeStyles(color: string, fontSize: string, fontWeight: string, fontFamily: string, letterSpacing: string): void {
    this.renderer.setStyle(this.el.nativeElement, 'color', color);
    this.renderer.setStyle(this.el.nativeElement, 'font-size', fontSize);
    this.renderer.setStyle(this.el.nativeElement, 'font-weight', fontWeight);
    this.renderer.setStyle(this.el.nativeElement, 'font-family', fontFamily);
    this.renderer.setStyle(this.el.nativeElement, 'letter-spacing', letterSpacing);
  }
}
```

Jes to fragment logo-directive. Dyrektywa jest używana do manipulowania elementami DOM. W tym przypadku, LogoDirective mogłaby być używana do powiększania i zmiany koloru loga naszego serwisu komputerowego.

3. Logika Routingu (app-routing.module.ts)
   Fragment Kodu:

```
const routes: Routes = [
  { path: '', component: LoginComponent },
  { path: 'registration', component: RegistrationComponent },
  { path: 'dashboard', component: TestComponent },
  { path: 'reports', component: ShowReportsComponent },
  { path: 'reportform', component: ShowReportFormComponent },
  { path: 'edit-report/:id', component: EditReportComponent },
  { path: 'showlogs', component: ShowLogsComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes), HttpClientModule],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Ten fragment kodu z app-routing.module.ts definiuje ścieżki w aplikacji. Routing w Angularze pozwala na ładowanie różnych komponentów w zależności od URL, co jest kluczowe dla Single Page Applications (SPA). Definiuje on na przykład, że ścieżka /register prowadzi do Registration Component, co jest przykładem dynamicznego ładowania komponentów.

5. Wypunktowane elementy techniczne, które zostały zrealizowane w projekcie wraz z krótkim komentarzem odnośnie realizacji: jak zrealizowano i w którym pliku.

1. Dodawanie Zgłoszeń Serwisowych
    - Realizacja: Funkcjonalność dodawania zgłoszeń serwisowych została realizowana za pomocą formularza w komponencie (show-report-form.component.html).
    - Plik: show-report-form.component.html i show-report-form.component.ts dla logiki formularza.
    - Komentarz: Użytkownik wypełnia formularz, który jest przetwarzany przez komponent, a następnie dane są wysyłane do serwera przez odpowiedni serwis (reports.service.ts).

2. Przeglądanie Zgłoszeń
    - Realizacja: Przeglądanie i filtrowanie zgłoszeń jest implementowane w komponencie show-reports.component.
    - Plik: show-reports.component.html dla UI i show-reports.component.ts dla logiki filtrowania.
    - Komentarz: Użytkownicy serwisu mogą używać różnych filtrów do przeglądania zgłoszeń, a dane są pobierane z serwera za pomocą reports.service.ts.

3. Przydzielanie Zadań
    - Realizacja: Przydzielanie zadań jest częścią logiki w komponencie show-reports.component.
    - Plik: show-reports.component.ts.
    - Komentarz: Funkcjonalność ta pozwala na przypisanie zgłoszenia do pracownika serwisu, przez zmianę atrybutu zgłoszenia w bazie danych.

4. Rozwiązywanie Problemów
    - Realizacja: Aktualizacja statusu zgłoszenia w komponencie edit-report.component.
    - Plik: edit-report.component.ts.
    - Komentarz: Pracownik może aktualizować status zgłoszenia, co jest odzwierciedlone w bazie danych.

5. Historia Zgłoszeń
   - Realizacja: Historia zgłoszeń jest zaimplementowana jako część komponentu show-reports.component.
   - Plik: show-reports.component.html i show-reports.component.ts.
   - Komentarz: Użytkownicy mogą przeglądać historię zgłoszeń.

6. Powiadomienia
   - Realizacja: Możliwość przeglądania historii zgłoszeń w display-logs.component.
   - Plik: display-logs.component.html i display-logs.component.ts.
   - Komentarz: Klienci mogą zobaczyć postęp swoich zgłoszeń w historii.

7. Klasy Modeli Danych
   - Realizacja: Klasy modeli danych takie jak Report.ts, Client.ts, User.ts.
   - Plik: Report.ts, Client.ts, User.ts.
   - Komentarz: Modele danych służą do strukturyzowania i przechowywania informacji o zgłoszeniach, klientach, pracownikach itp.

6. Podział pracy w zespole oraz podsumowanie

Podsumowanie projektu "Zarządzanie serwisem komputerowym" skupia się na obsłudze zgłoszeń serwisowych, zarządzaniu nimi i śledzeniu ich postępów. Został zrealizowany za pomocą różnych komponentów Angulara, serwisów oraz modeli danych, które razem tworzą kompleksowy system do zarządzania serwisem komputerowym. Każdy z tych elementów odgrywa istotną rolę w funkcjonowaniu aplikacji i jej interakcji z użytkownikiem oraz serwerem.

Funkcjonalności podzielone na osoby

- Przemysław Rutkowski
    - komponenty: Login oraz Registration
    - Routing
    - Modele: Client.ts oraz Serviceman.ts
    - Serwisy: User.service.ts, discount.service.ts

- Mateusz Mogielnicki
    - komponenty: EditReport oraz ShowReports
    - Modele: Status.ts, Priority.ts oraz przykładowa baza danych db.json
    - filtr: datePriorityPipe.ts
    - Serwis: logs.service.ts
    - ogólny wygląd wszystkich stron

- Jakub Matyszek
    - komponenty: displayLogs, showLogs, showReportForm
    - Modele: Log.ts oraz Report.ts
    - Serwis: reports.service.ts

Wszystkie inne nie wymienione aspekty serwisu komputerowego opracowaliśmy razem ze względu na złożoność problemu.