Login component

Code Sectie 1:

LoginComponent (login.component.ts)

```ts
import { Component } from '@angular/core';
import { AuthService } from '../services/security/auth.service';
import { NgForm } from '@angular/forms';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  username: string = '';
  password: string = '';
  scaleValue = 1;

  constructor(private authService: AuthService) {}

  changeScale() {
    this.scaleValue = 1.2;
  }

  changeScale2() {
    this.scaleValue = 1;
  }

  logIn() {
    this.authService.login(this.username, this.password);
  }
}
```

- @Component-decorator: Hier wordt de component geconfigureerd en aan de HTML en CSS gekoppeld.
- username en password: Variabelen die de gebruikersinvoer bijhouden.
- constructor: Hier wordt de AuthService-service geïnjecteerd om inlogfunctionaliteit mogelijk te maken.
- logIn(): Deze functie roept de login()-methode aan van de AuthService-service en geeft de gebruikersnaam en wachtwoord door om in te loggen.

Code Sectie 2:

HTML-sjabloon
(login.component.html)

```
<form (ngSubmit)="logIn()" #loginForm="ngForm">
  <div class="loginbox">
    <div class="text2">LOGIN</div>
    <div>
      <input
        type="text"
        placeholder="username"
        name="username"
        [(ngModel)]="username"
        required
      />
    </div>
    <div>
      <input
        type="password"
        placeholder="Password"
        name="password"
        [(ngModel)]="password"
        required
      />
    </div>
    <button type="submit" class="button0">LOG IN</button>
    <div class="text3">
      Don't you have an account? <br />Register below.
    </div>
    <button class="button0" routerLink="/registration">SIGN UP</button>
  </div>
</form>
```

- Hier wordt een formulier weergegeven met invoervelden voor gebruikersnaam en wachtwoord.
- Het (ngSubmit)-event wordt geactiveerd wanneer de gebruiker op de "LOG IN" knop klikt en roept de logIn()-functie aan uit de component.
- ((ngModel)) wordt gebruikt om de waarden van invoervelden bij te houden en te binden aan de username en password-variabelen in de component.

Registration component

Code Sectie 1:

RegistrationComponent
(registration.component.ts)

```
import { Component } from '@angular/core';
import { AuthService } from '../services/security/auth.service';

@Component({
  selector: 'app-registration',
  templateUrl: './registration.component.html',
  styleUrls: ['./registration.component.css']
})
export class RegistrationComponent {

  constructor(private authService: AuthService) {}

  register() {
    const username = (document.getElementById('username') as HTMLInputEleme
    const password = (document.getElementById('password') as HTMLInputEleme
    const email = (document.getElementById('email') as HTMLInputElement).va
    this.authService.register(username, password, email);
  }
}
```

- Dit TypeScript-bestand definieert een Angular-component voor registratie.
- @Component-decorator configureert de component en verwijst naar de bijbehorende HTML- en CSS-bestanden.
- In de constructor wordt de AuthService-service geïnjecteerd, wat suggereert dat deze component wordt gebruikt voor registratie met behulp van deze service.
- De register()-functie haalt waarden op uit invoervelden voor gebruikersnaam, wachtwoord en e-mail in het HTML-document en roept de register()-methode aan van de AuthService-service.

Code Sectie 2:

HTML-sjabloon
(registration.component.html)

```
<body>
  <div class="box">
    <div class="block">
      <div class="text1">THE READING</div>
      <div class="text5">of all good books is like</div>
      <div class="text6">CONVERSATION</div>
      <div class="text7">with the finest men of past centuries.</div>
      <div class="text0">(C) René Descartes</div>
      <form>
        <div class="loginbox">
          <div class="text2">REGISTRATION</div>
          <div>
            <input id="username" type="text" name="username" placeholder="u
          </div>
          <div>
            <input id="email" type="text" name="email" placeholder="email"
          </div>
          <div>
            <input id="password" type="password" name="password" placeholde
          </div>
          <button (click)="register()">SIGN UP</button>
          <div class="text3">Already have an account? <br>Login below</div>
          <button routerLink="/login">LOG IN</button>
        </div>
      </form>
      <div class="text4">Free Library</div>
      <ul>
        <li><a routerLink="/helppage">Help</a></li>
        <li><a routerLink="/about">About</a></li>
        <li><a routerLink="/privacy">Privacy</a></li>
        <li><a routerLink="/team">Team</a></li>
      </ul>
    </div>
  </div>
</body>
```

- Dit is de HTML-sjabloon voor de registratiepagina.
- Er zijn verschillende tekstuele elementen en een formulier voor registratie, inclusief invoervelden voor gebruikersnaam, wachtwoord en e-mail.
- Wanneer de gebruiker op de "SIGN UP" knop klikt, wordt de register()-functie van de component aangeroepen.

Code Sectie 3:

CSS-stijlen (registration.component.css)

Dit gedeelte van de code bevat CSS-stijlen die van toepassing zijn op elementen op de registratiepagina. Hier zijn enkele details:

- body: Dit stelt stijlen in voor de achtergrond van de hele pagina, zoals de achtergrondafbeelding en de grootte ervan.
- .button: Deze stijlen zijn van toepassing op knoppen op de pagina, inclusief achtergrondkleur, tekstkleur en knopgrootte.
- .block: Dit zijn de stijlen voor het blok dat de hoofdinhoud van de pagina bevat, zoals de achtergrondkleur en animatie-effecten.
- .text1, .text2, enzovoort: Dit zijn stijlen voor verschillende tekstuele elementen op de pagina, zoals lettertype, kleur en tekstuitlijning.
- .input: Stijlen voor invoervelden, inclusief breedte, hoogte en achtergrondkleur.
- .loginbox: Stijlen voor het registratieformulier, inclusief de achtergrondkleur en overgangseffecten wanneer je er met de muis overheen beweegt.
- @keyframes: Deze regels definiëren CSS-animaties. De ch-color-animatie wijzigt de tekstschaduw van tekstuele elementen, en de ani-animatie verandert de doorzichtigheid van elementen.

USER: Home component

Code Sectie 1:

TypeScript-code (home.component.ts)

- Vereiste modules:

```
import { Component } from '@angular/core';
import { ... } // Andere imports hier
```

- De component:

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
```

- Klasse en variabelen:

```
export class HomeComponent {
  private searchSubscription: Subscription | undefined;
  username: string = '';
  email: string = '';
  books: any[] = [];
  borrowedBooks: any[] = [];
  sortDirection: string = 'asc';
}
```

- Constructor met servicesinjecties:

```
constructor(private themeService: ThemeService,
            private authService:AuthService,
            private bookService:BookService,
            private searchService:SearchService,
            private borrowedBookService: BorrowedbookService,
            private cartService:CartService,
            private userService:UserService,
            public dialog: MatDialog) {}
```

Code Sectie 1:

TypeScript-code (home.component.ts)

Hier zijn details over de verschillende methoden in de Angular-component:

- sortDataByTitle(): Deze methode wordt aangeroepen wanneer de gebruiker op de kolomkop "Titel" klikt om de lijst met geleende boeken te sorteren. De sorteerfunctie werkt als volgt:

this.sortDirection houdt bij of de lijst momenteel in oplopende ('asc') of aflopende ('desc') volgorde wordt gesorteerd.

Als de lijst oplopend is gesorteerd ('asc'), wordt de lijst gesorteerd op basis van de titels van boeken in oplopende volgorde. Als deze al aflopend is gesorteerd ('desc'), wordt de lijst gesorteerd in oplopende volgorde.

Deze methode verandert de waarde van this.sortDirection zodat de knop met de pijl "omhoog" of "omlaag" kan worden bijgewerkt om de huidige sorteervolgorde weer te geven.

- sortDataByData(): Deze methode wordt op een vergelijkbare manier uitgevoerd als sortDataByTitle(), maar sorteert de lijst met geleende boeken op basis van de datum van retourneren (dataOfReturn) in oplopende of aflopende volgorde.

- sortDataByStatus(): Deze methode wordt opnieuw op een vergelijkbare manier uitgevoerd als de bovenstaande sorteerfuncties, maar sorteert de lijst met geleende boeken op basis van de status in oplopende of aflopende volgorde.

- openDescriptionDialog(book: any): Deze methode wordt aangeroepen wanneer een gebruiker op de "description" knop klikt naast een boek in de lijst met beschikbare boeken. Het opent een dialoogvenster waarin de beschrijving van het boek wordt weergegeven. De book parameter bevat de gegevens van het boek dat moet worden weergegeven in het dialoogvenster.

- toggleTheme(): Deze methode wordt aangeroepen wanneer een gebruiker op de knop klikt om tussen lichte en donkere thema's te schakelen. Het gebruikt de themeService om het thema van de applicatie te wijzigen tussen licht en donker.

Code Sectie 1:

TypeScript-code (home.component.ts)

Hier zijn details over de verschillende methoden in de Angular-component:

- addToCart(book: any): Deze methode wordt aangeroepen wanneer een gebruiker op de knop "ADD TO CART" klikt naast een boek in de lijst met beschikbare boeken. Het controleert of het boek al in het winkelwagentje is opgenomen en voegt het toe aan het winkelwagentje als dat niet het geval is. Het controleert ook of het winkelwagentje vol is en toont een waarschuwing als dit het geval is.

- cartItemCount: Dit is een getter die het aantal items in het winkelwagentje retourneert.

- cartItems: Dit is een getter die de array van items in het winkelwagentje retourneert.

- clearCart(): Deze methode wordt aangeroepen wanneer een gebruiker het winkelwagentje wil legen. Het verwijdert alle boeken uit het winkelwagentje.

- bookIsInCart(book: any): Deze methode controleert of een specifiek boek al in het winkelwagentje is opgenomen en retourneert true als dat het geval is, anders retourneert het false.

- getUserInfo(): Deze methode haalt gebruikersinformatie op en wordt uitgevoerd bij het initialiseren van de component. De gebruikersnaam en e-mail van de gebruiker worden ingesteld op basis van de gegevens die worden opgehaald via de userService.

- getAllBooks(): Deze methode haalt alle beschikbare boeken op en wordt uitgevoerd bij het initialiseren van de component.

- getBorrowedBooks(): Deze methode haalt alle geleende boeken op en wordt uitgevoerd bij het initialiseren van de component.

- ngOnDestroy(): Deze lifecycle-hook-methode wordt aangeroepen wanneer de component wordt vernietigd. Als er een zoekabonnement actief is, wordt het hiermee afgemeld om geheugenlekken te voorkomen.

Dit zijn de belangrijkste methoden in de Angular-component die verantwoordelijk zijn voor het sorteren, openen van dialoogvensters, beheren van winkelwagentjes, ophalen van gegevens en het verwerken van gebruikersinteracties op de pagina.

Code Sectie 2:

HTML-sjabloon:
(home.component.html)

1. Het HTML-sjabloon bevat de structuur van de webpagina. Het gebruikt *ngFor om boeken weer te geven en knoppen voor het winkelwagentje.
2. Er is een tabel om geleende boeken weer te geven en koppen die kunnen worden gebruikt om de lijst te sorteren.
3. De pagina bevat ook informatie over de gebruiker.

Code Sectie 3:

HTML-sjabloon:
(home.component.html)

1. CSS-stijlen om de pagina en elementen op te maken, waaronder achtergrondkleur, tekstkleur en tekstuitlijning.
2. Stijlen voor afbeeldingen en interactieve elementen.
3. Overgangsstijlen voor het in- en uitschakelen van het thema.
4. Mediaquery om de lay-out aan te passen op basis van de schermgrootte.

Dit is een complex Angular-component dat wordt gebruikt voor het weergeven van boekinformatie, het beheren van het winkelwagentje en het weergeven van geleende boeken. De HTML bevat structurele en interactieve elementen, terwijl de CSS verantwoordelijk is voor de visuele stijl en lay-out van de pagina. De TypeScript-code is verantwoordelijk voor de logica van het ophalen van gegevens en het beheren van interacties.

USER: Navbar component

Code Sectie 1:

TypeScript-code
(navbar.component.ts)

```typescript
export class NavbarComponent implements OnInit {
  // Array van navigatielinks
  navLinks: { label: string; route: string }[] = [];

  // Een vlag om de zichtbaarheid van inhoud te beheren
  isContentVisible = false;

  // FormControl voor zoekopdracht
  searchControl = new FormControl('');

  constructor(
    private themeService: ThemeService,
    private navService: NavService,
    private authService: AuthService,
    private searchService: SearchService
  ) {
    // Abonneer op wijzigingen in het zoekveld met debounceTime
    this.searchControl.valueChanges.pipe(debounceTime(300)).subscribe(quer
      if (query !== null) {
        this.searchService.updateSearchTerm(query);
      }
    });
  }

  ngOnInit(): void {
    // Haal navigatielinks op van de NavService
    this.navLinks = this.navService.getNavLinks();
  }

  // Methode om het thema te schakelen
  toggleTheme() {
    this.themeService.toggleTheme();
  }

  // Methode om inhoud zichtbaar te maken
  showContent() {
    this.isContentVisible = true;
  }

  // Methode om inhoud te verbergen
  hideContent() {
    this.isContentVisible = false;
  }

  // Methode om uit te loggen
  logOut(event: Event) {
    console.log('logOut methode aangeroepen');
    event.stopImmediatePropagation();
    event.stopPropagation();
    this.authService.logOut();
  }
}
```

Code Sectie 1:

TypeScript-code (navbar.component.ts)

toggleTheme(): Dit is een methode die wordt aangeroepen wanneer de "Toggle Theme" knop wordt ingedrukt. Het schakelt tussen lichte en donkere thema's. Deze methode roept de toggleTheme methode aan in de ThemeService om het thema te wijzigen.
Voorbeeld:

```
toggleTheme() {
  this.themeService.toggleTheme(); // Roep de themaschakelaar aan
}
```

showContent(): Deze methode wordt aangeroepen wanneer de showContent methode wordt getriggerd. Het stelt de isContentVisible eigenschap in op true.
Voorbeeld:

```
showContent() {
  this.isContentVisible = true; // Maak de inhoud zichtbaar
}
```

logOut(event: Event): Deze methode wordt aangeroepen wanneer de "LOG OFF" knop wordt ingedrukt. Het wordt gebruikt om de gebruiker uit te loggen door de logOut methode in de AuthService aan te roepen. Het ontvangt een Event als parameter en stopt de propagatie van dat event om te voorkomen dat het verder wordt verwerkt.
Voorbeeld:

```
logOut(event: Event) {
  console.log('logOut methode aangeroepen');
  event.stopImmediatePropagation(); // Stop de onmiddellijke propagatie
  event.stopPropagation(); // Stop de propagatie
  this.authService.logOut(); // Roep de uitlogmethode aan
}
```

Dit zijn enkele van de belangrijkste methoden in de NavbarComponent en hun functies. Ze worden gebruikt om interacties in de navigatiebalk te beheren, zoals het schakelen van thema's, tonen/verbergen van inhoud en uitloggen van de gebruiker.

Code Sectie 2:

HTML-sjabloon
(navbar.component.html)

```
<div class="nav">
  <div class="openmenu">MENU</div>
  <button *ngFor="let link of navLinks">
    <a [routerLink]="link.route">{{ link.label }}</a>
  </button>
  <button (click)="toggleTheme()">Toggle Theme</button>
  <button [routerLink]="['/cart']">CART</button>
  <form action="" method="get">
    <input [formControl]="searchControl" type="text" placeholder="Search..."
  </form>
  <button class="buttonExit" (click)="logOut($event)">LOG OFF</button>
</div>
```

Dit is het HTML-sjabloon voor de NavbarComponent. Hier volgt een samenvatting van de belangrijkste elementen:

- Een navigatiebalk ( ) bevat knoppen voor navigatielinks, een themaschakelaar, een zoekvak en een knop voor uitloggen.
- De *ngFor-directive wordt gebruikt om dynamisch navigatieknoppen te genereren op basis van de navLinks array.
- Het zoekvak is gekoppeld aan de searchControl FormControl voor live zoeken.
- De knoppen bevatten klikgebeurtenisluisteraars die methoden van de NavbarComponent aanroepen.

Dit is een overzicht van de code voor de NavbarComponent. Het component wordt gebruikt om de navigatiebalk van een Angular-toepassing te beheren en biedt functies zoals navigatie, themaschakeling en zoekfunctionaliteit.

USER: Cart component

Code Sectie 1:

TypeScript code
(cart.component.ts)

```typescript
import { Component } from '@angular/core';
import { CartService } from 'src/app/services/cartservice/cart.service';

@Component({
  selector: 'app-cart',
  templateUrl: './cart.component.html',
  styleUrls: ['./cart.component.css']
})
export class CartComponent {
  constructor(public cartService: CartService) { }

  getCartItems() {
    return this.cartService.items;
  }

  reserveBooks() {
    this.cartService.reserveBooks();
  }

  clearCart() {
    this.cartService.removeAllBooksFromCart();
  }

  removeBookFromCart(bookId: number) {
    this.cartService.removeBookFromCart(bookId);
  }
}
```

Dit is een Angular-component genaamd CartComponent, verantwoordelijk voor het beheren van de winkelwagenweergave.

Code Sectie 1:

TypeScript code
(cart.component.ts)

1. getCartItems(): Deze methode retourneert een lijst met items in de winkelwagen door toegang te krijgen tot de items eigenschap van de CartService.

2. reserveBooks(): Wanneer deze methode wordt aangeroepen, roept deze de reserveBooks() methode aan in de CartService. Dit suggereert dat het verantwoordelijk is voor het reserveren van boeken in de winkelwagen.

3. clearCart(): Deze methode roept de removeAllBooksFromCart() methode aan in de CartService, wat betekent dat het verantwoordelijk is voor het verwijderen van alle boeken uit de winkelwagen.

4. removeBookFromCart(bookId: number): Wanneer deze methode wordt aangeroepen met een bookId, roept deze de removeBookFromCart(bookId) methode aan in de CartService. Het verwijdert het specifieke boek met de opgegeven bookId uit de winkelwagen.

Code Sectie 2:

HTML sjabloon
(cart.component.html)

```
<body class="content">
  <app-navbarcart></app-navbarcart>
  <div class="container">
    <div style="display: flex">
      <button  (click)="reserveBooks()" class="buttonCart" style="position:
      <button (click)="clearCart()" class="buttonCart" style="position: fix
        CLEAR CART
      </button>
    </div>

    <div class="my-custom-div" style="margin-top: 3%">
      <div class="fixed-block"></div>
      <div class="scroll">
        <table>
          <!-- ... -->
        </table>
      </div>
    </div>
  </div>
</body>
```

Dit is het HTML-sjabloon voor de CartComponent. Het bevat een navigatiebalk (<app-navbarcart>) en knoppen voor acties.

USER: Account component

Code Sectie 1:

TypeScript code
(account.component.ts)

```typescript
import { Component } from '@angular/core';
import { BorrowedbookService } from 'src/app/services/borrowedbookservice/b
import { UserService } from 'src/app/services/userservice/user.service';

@Component({
  selector: 'app-account',
  templateUrl: './account.component.html',
  styleUrls: ['./account.component.css']
})
export class AccountComponent {
  // Attributen
  reservedBooks: any[] = [];
  sortDirection: string = 'asc';
  username: string = '';
  email: string = '';
  newUsername: string = '';
  newEmail: string = '';
  oldPassword: string = '';
  newPassword: string = '';
  confirmPassword: string = '';
  disableOtherFields: boolean = false;
  activeField: string = '';

  // Constructor
  constructor(private userService: UserService, private borrowedBookService

  // Lifecycle-hook
  ngOnInit(): void {
    this.getAllBorrwowedBooks();
    this.getUserInfo();
  }

  // Methoden
  sortDataByTitle() { /* ... */ }
  sortDataByData() { /* ... */ }
  onInputChange(field: string): void { /* ... */ }
  returnBook(bookId: number): void { /* ... */ }
  getAllBorrwowedBooks(): void { /* ... */ }
  changePassword(): void { /* ... */ }
  getUserInfo(): void { /* ... */ }
  updateUserInfo(): void { /* ... */ }
  deleteAccount(): void { /* ... */ }
}
```

Code Sectie 1:

TypeScript code
(account.component.ts)

getUserInfo(): Deze methode haalt de gebruikersinformatie op van de backend-service via de userService. Het gebruikt een asynchrone HTTP-aanroep om de gegevens op te halen en wijst deze toe aan de username en email attributen. Voorbeeld van gebruik in HTML-sjabloon:

```
<p class="userTextLogin">{{username}}</p>
<p class="userTextEmail">{{email}}</p>
```

updateUserInfo(): Deze methode wordt aangeroepen wanneer de gebruiker zijn/haar gebruikersnaam of e-mail wil bijwerken. Het maakt een PUT-verzoek naar de backend om de nieuwe gegevens op te slaan. Het nieuwe gebruikersnaam en e-mailadres worden uit de newUsername en newEmail attributen gehaald. Voorbeeld van gebruik in HTML-sjabloon:

```
<input
  type="text"
  placeholder="New username"
  [(ngModel)]="newUsername"
  (input)="onInputChange('username')"
  [disabled]="disableOtherFields && activeField !== 'username'"
/>
<button
  class="buttonEditUser"
  style="margin-top: 3%; width: 15%; margin-left: 42%"
  (click)="updateUserInfo()"
>
  CONFIRM
</button>
```

Code Sectie 1:

TypeScript code
(account.component.ts)

deleteAccount(): Deze methode wordt aangeroepen wanneer de gebruiker zijn/haar account wil verwijderen. Het maakt een DELETE-verzoek naar de backend om het account te verwijderen.
Voorbeeld van gebruik in HTML-sjabloon:

```
<button
  class="buttonEditUser"
  style="
    margin-top: 3%;
    width: 15%;
    margin-left: 42%;
    background-color: darkred;
  "
  (click)="deleteAccount()"
>
  DELETE ACCOUNT
</button>
```

Deze methoden stellen de gebruiker in staat om zijn/haar profielinformatie bij te werken en het account te verwijderen. De gegevens die worden verzonden naar of ontvangen van de backend, zijn afhankelijk van de backend-service en de API die wordt gebruikt in de Angular-toepassing.

ADMIN: Adminhome component

ADMIN: Users component

Book-description component

Code Sectie 1:

TypeScript code
(BookDescriptionDialogComponent)

```typescript
import { Component, Inject } from '@angular/core';
import { MatDialogRef, MAT_DIALOG_DATA } from '@angular/material/dialog';

@Component({
  selector: 'app-book-description-dialog',
  template: `
    <h2 mat-dialog-title>{{data.title}}</h2>
    <mat-dialog-content class="test">
      {{data.description}}
    </mat-dialog-content>
    <mat-dialog-actions>
      <button mat-button mat-dialog-close>Close</button>
    </mat-dialog-actions>
  `,
})
export class BookDescriptionDialogComponent {
  constructor(
    public dialogRef: MatDialogRef<BookDescriptionDialogComponent>,
    @Inject(MAT_DIALOG_DATA) public data: {title: string; description: stri
  ) {}
}
```

Code Sectie 1:

TypeScript code (BookDescriptionDialogComponent)

@Component Decorator: Dit markeert de klasse als een Angular-component en definieert enkele van de eigenschappen van de component, zoals de selector (waarmee je de component kunt identificeren in je HTML-sjabloon) en de template (de weergave van de component).

constructor: De constructor is verantwoordelijk voor het initialiseren van de BookDescriptionDialogComponent. Het accepteert twee parameters:

- dialogRef: Dit is een verwijzing naar het huidige dialoogvenster en wordt gebruikt om het dialoogvenster te sluiten of te beheren.
- data: Dit is een object dat wordt geïnjecteerd met behulp van MAT_DIALOG_DATA. Het bevat informatie over het boek, zoals de titel en de beschrijving.

Template: De template-sectie definieert de structuur van het dialoogvenster. Het gebruikt Angular Material-componenten om de titel, beschrijving en een sluitknop weer te geven. De waarden voor de titel en de beschrijving worden gehaald uit het data-object dat wordt geïnjecteerd.

mat-dialog-title, mat-dialog-content, en mat-dialog-actions zijn Angular Material-componenten die worden gebruikt om respectievelijk de titel, inhoud en acties van het dialoogvenster weer te geven.

De knop met het label "Close" in mat-dialog-actions wordt gebruikt om het dialoogvenster te sluiten. Dit wordt bereikt met mat-dialog-close, dat automatisch het dialoogvenster sluit wanneer erop wordt geklikt.

Dit component wordt gebruikt om een dialoogvenster te openen dat boekinformatie weergeeft, zoals de titel en beschrijving. De informatie wordt geleverd via het data-object dat wordt doorgegeven aan het dialoogvenster wanneer het wordt geopend. Wanneer de gebruiker op "Close" klikt, wordt het dialoogvenster gesloten.

SERVICES

ThemeService:

TypeScript Code (ThemeService):

@Injectable Decorator: Dit markeert de **ThemeService**-klasse als een injecteerbare service in Angular, waardoor deze beschikbaar is voor andere delen van de applicatie.

constructor: De constructor initialiseert de **ThemeService**. Het accepteert een **RendererFactory2**-instantie om een **Renderer2** te maken. De **Renderer2** wordt gebruikt om het thema in de applicatie bij te werken. Daarnaast worden de oorspronkelijke thema-instellingen (**isDark**, **isLight**) opgehaald via de **getInitialTheme**, **getInitialTheme2** methoden, en vervolgens wordt **updateTheme** opgeroepen om het thema in te stellen op basis van deze instellingen.

toggleTheme(): Deze methode wordt aangeroepen om te schakelen tussen thema's (bijvoorbeeld van licht naar donker of omgekeerd). Het keert de waarden van **isDark**, **isLight** om, slaat de themavoorkeur op in de lokale opslag (localStorage) en werkt het thema van de applicatie bij met behulp van **updateTheme**.

getInitialTheme(), **getInitialTheme2()**: Deze methoden worden gebruikt om de oorspronkelijke themavoorkeur op te halen uit de lokale opslag. Ze controleren of er een opgeslagen thema-instelling is en keren terug naar **true** als deze overeenkomt met het specifieke thema (bijv. 'dark', 'light', 'mid'), en anders wordt **false** teruggegeven. Dit bepaalt welk thema wordt weergegeven bij het opstarten van de applicatie.

saveThemePreference(): Deze methode slaat de huidige themavoorkeur op in de lokale opslag, zodat de themavoorkeur behouden blijft tussen sessies.

updateTheme(): Deze methode past het actieve thema van de applicatie aan door klassen toe te voegen en te verwijderen op het **document.body**-element. Als het donkere thema is geselecteerd, wordt de klasse 'dark-theme' toegevoegd en de klasse 'light-theme' verwijderd, en vice versa.

Deze service stelt de applicatie in staat om themawijzigingen te beheren en het thema dynamisch aan te passen op basis van gebruikersvoorkeuren.

BookService:

TypeScript Code
(BookService)

```typescript
import { Injectable } from '@angular/core';
import { UtilsService } from '../security/utils.service';
import axios from 'axios';

@Injectable({
  providedIn: 'root',
})
export class BookService {

  constructor(private util: UtilsService) { }

  async getAllBooks(): Promise<any> {
    const url = 'http://localhost:8080/api/book/getbooks';

    try {
      if (!this.util.getTokenFromLocalStorage()) {
        return Promise.reject('No token found');
      }

      // Maak een GET-verzoek naar de geparametriseerde URL
      return axios.get(url, this.util.getConfig()).then(response => respons

    } catch (error) {
      return Promise.reject(error);
    }
  }

  async searchBooks(query: string): Promise<any> {
    const url = 'http://localhost:8080/api/book';
    const parameterizedUrl = `${url}/search?query=${query}`;

    try {
      const response = await axios.get(parameterizedUrl, this.util.getConf
      return response.data;
    } catch (error) {
      console.error(error);
      return Promise.reject(error);
    }
  }
}
```

BookService:

TypeScript Code
(BookService)

1. **@Injectable** Decorator: Dit markeert de **BookService**-klasse als een injecteerbare service in Angular, waardoor deze beschikbaar is voor andere delen van de applicatie.
2. **constructor**: Hier wordt de constructor van de service gedefinieerd. Deze constructor ontvangt een afhankelijkheid, namelijk **UtilsService**, via dependency injection. De **UtilsService** wordt gebruikt om toegang te krijgen tot hulpmethoden voor het verkrijgen van een token en configuratie.
3. **getAllBooks()**: Dit is een asynchrone methode die wordt gebruikt om alle boeken op te halen vanuit een API. Het stelt eerst de URL in voor het GET-verzoek naar de boeken. Vervolgens controleert het of er een geldig token beschikbaar is met behulp van de **this.util.getTokenFromLocalStorage()** methode. Als er geen token beschikbaar is, wordt een fout geretourneerd. Anders wordt een GET-verzoek gedaan naar de URL met behulp van Axios en de configuratie opgehaald van **this.util.getConfig()**. De gegevens van het antwoord worden geretourneerd.
4. **searchBooks(query: string)**: Deze methode wordt gebruikt om boeken te zoeken op basis van een zoekopdracht (**query**). Het bouwt de URL voor het zoekverzoek op en maakt vervolgens een GET-verzoek naar die URL met behulp van Axios en de configuratie van **this.util.getConfig()**. Het resultaat van het zoekverzoek wordt geretourneerd.
5. **try-catch** Blokken: De code maakt gebruik van **try-catch** blokken om foutafhandeling uit te voeren. Als er een fout optreedt tijdens de HTTP-verzoeken of tokencontrole, wordt de fout afgehandeld en geretourneerd als een verworpen promise. De fout wordt ook gelogd naar de console voor foutopsporing.

De **BookService**-service is verantwoordelijk voor het afhandelen van HTTP-verzoeken voor het ophalen van boeken en het uitvoeren van zoekopdrachten. Het maakt gebruik van Axios, een populaire HTTP-client, om HTTP-verzoeken uit te voeren naar de gespecificeerde API-eindpunten. De **UtilsService** wordt gebruikt om toegang te krijgen tot hulpmethoden voor tokenverificatie en configuratie.

BookService:

TypeScript Code
(BookService)

```typescript
import { Injectable } from '@angular/core';
import { UtilsService } from '../security/utils.service';
import axios from 'axios';

@Injectable({
  providedIn: 'root',
})
export class BorrowedbookService {

  constructor(private util: UtilsService) { }

  async getActiveBorrowedBooks(): Promise<any> {
    const url = 'http://localhost:8080/api/borrow/getall';

    try {
      if (!this.util.getTokenFromLocalStorage()) {
        return Promise.reject('No token found');
      }

      // Haal de gebruikersnaam op uit het gedecodeerde token ("sub" in tok
      const id = this.util.getDecodedToken().id;

      console.log('username: ', id);

      // Maak een geparametriseerde URL
      const parameterizedUrl = url + '?id=' + id;

      console.log(this.util.getTokenFromLocalStorage()); // Alleen voor deb
      console.log(this.util.getConfig()); // Alleen voor debugdoeleinden

      // Voer een GET-verzoek uit naar de geparametriseerde URL
      return axios.get(parameterizedUrl, this.util.getConfig()).then(respons
    } catch (error) {
      return Promise.reject(error);
    }
  }

  returnBook(bookId: number): void {
    const bookIds: number[] = [];

    bookIds.push(bookId);

    console.log("Dit zijn de boek-ID's " + bookIds);

    // Voer een POST-verzoek uit naar het retourboekeindpunt
    axios.post('http://localhost:8080/api/borrow/returnbooks?userId=' + thi
  }
```

BorrowedbookService:

TypeScript Code
(BorrowedbookService)

1. **@Injectable** Decorator: Dit markeert de **BorrowedbookService**-klasse als een injecteerbare service in Angular.
2. **constructor**: De constructor van de service ontvangt de afhankelijkheid **UtilsService** via dependency injection. Deze service wordt gebruikt om toegang te krijgen tot hulpmethoden voor tokenverificatie en configuratie.
3. **getActiveBorrowedBooks()**: Dit is een asynchrone methode die wordt gebruikt om alle actieve geleende boeken op te halen voor de ingelogde gebruiker. Het stelt eerst de URL in voor het GET-verzoek naar geleende boeken. Vervolgens controleert het of er een geldig token beschikbaar is met behulp van **this.util.getTokenFromLocalStorage()**. Als er geen token beschikbaar is, wordt een fout geretourneerd. Anders haalt het de gebruikersnaam op uit het gedecodeerde token en bouwt het een geparametriseerde URL op. Vervolgens wordt een GET-verzoek gedaan naar de URL met Axios en de configuratie van **this.util.getConfig()**. Het resultaat van het GET-verzoek wordt geretourneerd.
4. **returnBook(bookId: number)**: Deze methode wordt gebruikt om een geleend boek terug te geven. Het verwacht een boek-ID als parameter. De boek-ID wordt in een array geplaatst, en vervolgens wordt een POST-verzoek gedaan naar het retourboekeindpunt met de boek-ID's en de gebruikers-ID van het gedecodeerde token. Dit wordt gedaan met behulp van Axios en de configuratie van **this.util.getConfig()**.
5. **try-catch** Blokken: De code maakt gebruik van **try-catch** blokken om foutafhandeling uit te voeren. Als er een fout optreedt tijdens de HTTP-verzoeken of tokencontrole, wordt de fout afgehandeld en geretourneerd als een verworpen promise. De fouten worden ook gelogd naar de console voor foutopsporing.

De **BorrowedbookService**-service is verantwoordelijk voor het afhandelen van HTTP-verzoeken met betrekking tot geleende boeken, waaronder het ophalen van actieve geleende boeken en het retourneren van boeken. Het maakt gebruik van Axios om HTTP-verzoeken uit te voeren naar de gespecificeerde API-eindpunten en de **UtilsService** voor tokenverificatie en configuratie.

CartService:

TypeScript Code (CartService)

@Injectable Decorator: Dit markeert de CartService-klasse als een injecteerbare service in Angular.

constructor: De constructor van de service ontvangt de afhankelijkheden BookService en UtilsService via dependency injection. Het initialiseert ook de cart-array door op te halen wat er in de lokale opslag is opgeslagen.

```
import { Injectable } from '@angular/core';
import { BookService } from '../bookservice/book.service';
import axios from 'axios';
import { UtilsService } from '../security/utils.service';

@Injectable({
  providedIn: 'root'
})
export class CartService {

  private cart: any[] = [];
  private readonly maxCartSize = 5;

  constructor(private bookService: BookService, private util: UtilsService)
    const storedCart = localStorage.getItem('cart');
    if (storedCart) {
      this.cart = JSON.parse(storedCart);
    } else {
      this.cart = [];
    }
  }
```

CartService:

TypeScript Code (CartService)

items Getter: Dit stelt een getter-methode in die de inhoud van de winkelwagen retourneert.

addToCart(item: any): boolean: Deze methode wordt gebruikt om een boekitem aan de winkelwagen toe te voegen. Als de winkelwagen niet de maximale grootte heeft bereikt, wordt het item aan de cart-array toegevoegd en wordt de winkelwagen naar de lokale opslag geschreven. De methode geeft true terug als het toevoegen is gelukt, anders geeft het false terug.

isBookInCart(book: any): boolean: Deze methode controleert of een bepaald boek al in de winkelwagen zit. Het gebruikt de cart-array om te zoeken naar een overeenkomst op basis van het boek-ID. Het retourneert true als het boek al in de winkelwagen zit, anders false.

```
get items() {
  return this cart;
}

addToCart(item: any): boolean {
  if (this.cart.length < this maxCartSize) {
    this.cart.push(item);
    localStorage.setItem('cart', JSON.stringify(this.cart));
    return true;
  }
  return false;  // Geeft aan dat het item niet kon worden toegevoegd van
}

isBookInCart(book: any): boolean {
  return this.cart.some(item => item.id === book.id);
}
```

CartService:

TypeScript Code (CartService)

removeAllBooksFromCart(): Deze methode verwijdert alle boeken uit de winkelwagen door de cart-array leeg te maken en de lokale opslag bij te werken.

removeBookFromCart(bookId: number): Deze methode verwijdert een specifiek boek uit de winkelwagen op basis van het boek-ID. Het gebruikt de filter-functie om alle boeken behalve degene die moeten worden verwijderd in de cart-array achter te laten.

reserveBooks(): Deze methode wordt gebruikt om de geselecteerde boeken in de winkelwagen te reserveren. Het haalt de gebruikersconfiguratie en token op en maakt een verzoek om elk boek afzonderlijk te reserveren. Als een boek is gereserveerd, wordt het uit de winkelwagen verwijderd.

```
removeAllBooksFromCart() {
  this.cart = [];
  localStorage.setItem('cart', JSON.stringify(this.cart));
}

removeBookFromCart(bookId: number): void {
  this.cart = this.cart.filter(item => item.id !== bookId);
  localStorage.setItem('cart', JSON.stringify(this.cart));
}

reserveBooks(): void {
  const config = this.util.getConfig();
  const token = this.util.getDecodedToken();
  const url = 'http://localhost:8080/api/borrow/reserve';

  const BorrowBookRequest = {
    userId: token.id,
    bookId: 0
  };

  const bookIds = this.cart.map(item => item.id);  // Maak een array van
```

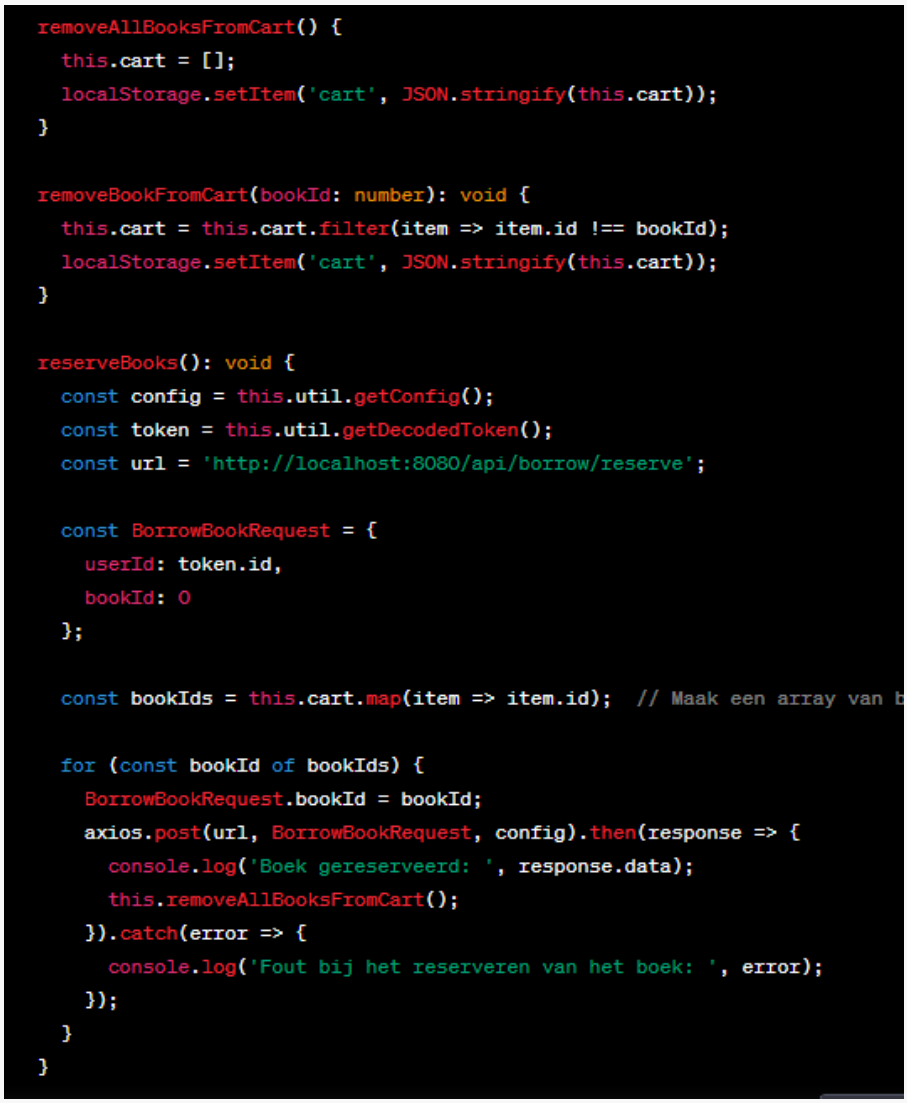
CartService:

TypeScript Code (CartService)

removeAllBooksFromCart(): Deze methode verwijdert alle boeken uit de winkelwagen door de cart-array leeg te maken en de lokale opslag bij te werken.

removeBookFromCart(bookId: number): Deze methode verwijdert een specifiek boek uit de winkelwagen op basis van het boek-ID. Het gebruikt de filter-functie om alle boeken behalve degene die moeten worden verwijderd in de cart-array achter te laten.

reserveBooks(): Deze methode wordt gebruikt om de geselecteerde boeken in de winkelwagen te reserveren. Het haalt de gebruikersconfiguratie en token op en maakt een verzoek om elk boek afzonderlijk te reserveren. Als een boek is gereserveerd, wordt het uit de winkelwagen verwijderd.



```
removeAllBooksFromCart() {
  this.cart = [];
  localStorage.setItem('cart', JSON.stringify(this.cart));
}

removeBookFromCart(bookId: number): void {
  this.cart = this.cart.filter(item => item.id !== bookId);
  localStorage.setItem('cart', JSON.stringify(this.cart));
}

reserveBooks(): void {
  const config = this.util.getConfig();
  const token = this.util.getDecodedToken();
  const url = 'http://localhost:8080/api/borrow/reserve';

  const BorrowBookRequest = {
    userId: token.id,
    bookId: 0
  };

  const bookIds = this.cart.map(item => item.id);  // Maak een array van b

  for (const bookId of bookIds) {
    BorrowBookRequest.bookId = bookId;
    axios.post(url, BorrowBookRequest, config).then(response => {
      console.log('Boek gereserveerd: ', response.data);
      this.removeAllBooksFromCart();
    }).catch(error => {
      console.log('Fout bij het reserveren van het boek: ', error);
    });
  }
}
```

CartService:

TypeScript Code
(CartService)

retrieveBooksFromStoredCart(): Deze methode haalt de inhoud van de winkelwagen op uit de lokale opslag als deze daar is opgeslagen. Het werkt de cart-array bij met de opgehaalde gegevens.

```
  retrieveBooksFromStoredCart(): void {
    const storedCart = localStorage.getItem('cart');
    if (storedCart) {
      this.cart = JSON.parse(storedCart);
    } else {
      this.cart = [];
    }
  }
}
```

De CartService-service biedt functies voor het beheren van een winkelwagen, waaronder het toevoegen, verwijderen, controleren van de beschikbaarheid van boeken en het reserveren van boeken.

SearchService:

TypeScript Code (SearchService)

```typescript
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class SearchService {

  constructor() { }

  private searchSubject = new BehaviorSubject<string>('');

  searchObservable = this.searchSubject.asObservable();

  updateSearchTerm(term: string) {
    this.searchSubject.next(term);
  }
}
```

@Injectable Decorator: Dit markeert de SearchService-klasse als een injecteerbare service in Angular. Hierdoor kan deze service worden geïnjecteerd in andere onderdelen van de Angular-applicatie.

constructor: De constructor van de service is leeg. Het bevat geen parameters of logica.

searchSubject: Dit is een privégedeelte van de service en is een instantie van BehaviorSubject. Een BehaviorSubject is een speciaal type observable dat zowel waarden uitgeeft als bijhoudt wat de laatst uitgegeven waarde was. In dit geval is het een BehaviorSubject die strings uitgeeft.

searchObservable: Dit gedeelte maakt een openbare eigenschap searchObservable die de searchSubject blootstelt als een Observable. Dit maakt het mogelijk voor andere onderdelen van de applicatie om zich te abonneren op wijzigingen in de zoekterm.

updateSearchTerm(term: string): Dit is een openbare methode waarmee andere delen van de applicatie een nieuwe zoekterm aan de service kunnen doorgeven. Wanneer deze methode wordt aangeroepen, wordt de nieuwe zoekterm als waarde naar de searchSubject verzonden met behulp van next(). Dit leidt tot het uitgeven van de nieuwe zoekterm aan iedereen die is geabonneerd op searchObservable.

UserService:

TypeScript Code (UserService)

@Injectable Decorator: Markeert de UserService als een injecteerbare service in Angular, zodat deze in andere delen van de applicatie kan worden geïnjecteerd.

constructor: De constructor accepteert twee services, UtilsService en AuthService, via dependency injection.

getUserInfo(): Een methode waarmee de gebruikersinformatie kan worden opgehaald. Deze functie maakt een GET-verzoek naar een API-endpoint om de gebruikersinformatie op te halen op basis van de gebruikers-id die wordt ontleed uit het JWT-token. De methode retourneert een Promise met de gebruikersinformatie.

```typescript
import { Injectable } from '@angular/core';
import { UtilsService } from '../security/utils.service';
import axios from 'axios';
import { AuthService } from '../security/auth.service';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  constructor(private util: UtilsService, private authService: AuthService)

  getUserInfo(): Promise<any> {
    return axios.get('http://localhost:8080/api/users/get/id?id=' + this.ut
      .then(response => {
        if (response.data) {
          const user = {
            id: response.data.id,
            username: response.data.username,
            email: response.data.email
          };
          return user;
        } else {
          throw new Error('No data returned');
        }
      })
      .catch(error => {
        console.error('Error getting user info:', error);
        throw error;
      });
```

UserService:

TypeScript Code (UserService)

checkPassword(password: string): Deze asynchrone methode controleert het wachtwoord van de gebruiker door een GET-verzoek te doen naar een API-endpoint met het opgegeven wachtwoord en gebruikers-id. Het retourneert true als het wachtwoord overeenkomt en false als dit niet het geval is.

updateUserInfo(user: any): Deze methode maakt een PUT-verzoek om gebruikersinformatie bij te werken en de gebruiker wordt uitgelogd na een succesvolle update.

deleteUser(): Deze methode verwijdert het gebruikersaccount na bevestiging en logt de gebruiker uit. Het maakt een DELETE-verzoek naar een API-endpoint om het account te verwijderen.

Dit zijn de belangrijkste functionaliteiten van de UserService, die verantwoordelijk is voor het beheren van gebruikersgerelateerde taken zoals het ophalen van informatie, het controleren van wachtwoorden, bijwerken van informatie en het verwijderen van gebruikersaccounts.

```
  async checkPassword(password: string): Promise<boolean> {
    const userId = this.util.getDecodedToken().id;
    const response = await axios.get('http://localhost:8080/api/users/chec
    return response;
  }

  updateUserInfo(user: any): void {
    const userId = this.util.getDecodedToken().id;
    user.id = userId;
    axios.post('http://localhost:8080/api/users/update', user, this.util.g
    alert("User info updated successfully, you will be redirected to the l
    this.authService.logOut();
  }

  deleteUser(): void {
    const confirmDeletion = confirm("Are you sure you want to delete your
    if (confirmDeletion) {
      axios.delete('http://localhost:8080/api/users/delete?id=' + this.util
      alert("We are sad to see you leave :(, you will be redirected to the
      this.authService.logOut();
    }
  }
}
```

SECURITY

UtilsService:

TypeScript Code (UtilsService)

@Injectable Decorator: Markeert de UtilsService als een injecteerbare service in Angular, zodat deze in andere delen van de applicatie kan worden geïnjecteerd.

constructor: De constructor van de service is leeg, aangezien deze geen externe afhankelijkheden heeft.

getTokenFromLocalStorage(): Deze methode haalt het JWT-token op uit de lokale opslag van de browser, dat meestal wordt opgeslagen na een succesvolle inlogpoging.

```
import { Injectable } from '@angular/core';
import jwt_decode from 'jwt-decode';

@Injectable({
  providedIn: 'root'
})
export class UtilsService {
  constructor() { }

  getTokenFromLocalStorage(): string {
    return localStorage.getItem('authToken') || '';
  }
```

deCodeToken(token: string): Deze methode decodeert een JWT-token en retourneert de gedecodeerde informatie. Het gebruikt de externe bibliotheek jwt-decode om dit te doen.

```
deCodeToken(token: string): any {
  return jwt_decode(token);
}
```

getDecodedToken(): Deze methode haalt het JWT-token uit de lokale opslag en decodeert het om de gedecodeerde tokeninformatie op te halen. De gedecodeerde tokeninformatie bevat vaak claims zoals gebruikers-id, naam, rol, enzovoort.

```
getDecodedToken(): any {
  return this.deCodeToken(this.getTokenFromLocalStorage());
}
```

UtilsService:

TypeScript Code (UtilsService)

getConfig(): Deze methode retourneert een configuratie-object dat vaak wordt gebruikt voor het maken van HTTP-verzoeken. Het omvat een Authorization header met het JWT-token, wat handig is voor authenticatie bij beveiligde API-verzoeken.

```
  getConfig(): any {
    return {
      headers: {
        Authorization: `Bearer ${this.getTokenFromLocalStorage()}`
      }
    };
  }
```

getUserRole(): Deze methode haalt de rol van de huidige gebruiker op uit de lokale opslag. De rollen worden meestal opgeslagen als een array van strings, en de methode retourneert de eerste rol in de array. Dit kan handig zijn om bepaalde acties of toegangscontroles te beheren op basis van de rol van de gebruiker.

```
  getUserRole(): string {
    const roles = JSON.parse(localStorage.getItem('roles') || '[]');
    return roles[0];
  }
}
```

De UtilsService bevat hulpprogrammafuncties die vaak nodig zijn bij het werken met JWT-tokens, zoals het decoderen van tokens, het ophalen van gebruikersinformatie, het ophalen van rollen en het samenstellen van HTTP-verzoekconfiguraties voor authenticatie. Het is een nuttige service voor verschillende delen van de applicatie, met name voor het beheren van gebruikersgerelateerde taken.

AuthService:

TypeScript Code (AuthService)

@Injectable Decorator: Markeert de AuthService als een injecteerbare service in Angular. Hiermee kan de service in andere delen van de applicatie worden geïnjecteerd.

constructor: De constructor van de service ontvangt twee afhankelijkheden, Router en UtilsService. De Router wordt gebruikt om de navigatie binnen de applicatie te beheren, en UtilsService wordt gebruikt voor verschillende hulpprogrammafuncties, zoals het werken met JWT-tokens.

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import axios from 'axios';
import { UtilsService } from './utils.service';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  constructor(private router: Router, private utils: UtilsService) { }
```

login(inputusername, inputpassword): Deze methode wordt gebruikt om gebruikers aan te melden bij de applicatie. Het maakt een POST-verzoek naar de inlog-eindpunt en stuurt de inloggegevens.

```
login(inputusername: string, inputpassword: string) {
  // Create a request object with the username and password
  const loginRequest = {
    username: inputusername,
    password: inputpassword
  };
```

laxios.post(...): Hier wordt een POST-verzoek gedaan naar het inlog-eindpunt met behulp van Axios, een populaire HTTP-clientbibliotheek. Het wacht op een reactie van de server.

```
axios.post('http://localhost:8080/api/auth/signin', loginRequest, {
})
  .then(response => {
    // Handle the response from the server
    console.log('Login successful:', response);
```

AuthService:

TypeScript Code (AuthService)

Serverrespons verwerken: De methode verwerkt de respons van de server na een succesvolle aanmelding. Het decodeert het JWT-token en de rollen uit de respons, slaat ze op in de lokale opslag en leidt de gebruiker door naar de juiste bestemming op basis van hun rol.

```
        const accessToken = response.data.accessToken;
        const roles = response.data.roles;

        if (accessToken && roles) {
          // Save the accessToken and roles to local storage
          localStorage.setItem('authToken', accessToken);
          localStorage.setItem('roles', JSON.stringify(roles));

          // Redirect the user based on their role
          switch (roles[0]) {
            case 'ROLE_USER':
              this.router.navigate(['']);
              break;
            case 'ROLE_ADMIN':
              this.router.navigate(['/admin/users']);
              break;
            default:
              console.error('Invalid role:', roles[0]);
          }
        } else {
          console.error('No accessToken received');
        }
      })
      .catch(error => {
        console.error('Login error:', error);
        alert('Login failed, check the console for details');
      });
  }
```

AuthService:

TypeScript Code (AuthService)

```
register(inputusername: string, inputpassword: string, inputemail: string
  // Create a request object with username, password, and email
  const signUpRequest = {
    username: inputusername,
    password: inputpassword,
    email: inputemail
  };
```

register(inputusername, inputpassword, inputemail): Deze methode wordt gebruikt om een nieuwe gebruiker te registreren. Het maakt een POST-verzoek naar het registratie-eindpunt en stuurt de registratiegegevens.

```
  axios.post('http://localhost:8080/api/auth/signup', signUpRequest, {
    headers: {
      'Content-Type': 'application/json'
    }
  }).then(response => {
    console.log('Registration successful:', response);

    alert('Registration successful, please login');

    this.router.navigate(['/login']);
  }).catch(error => {
    console.error('Registration error:', error);
    alert('Registration failed, check the console for details');
  });
}
```

axios.post(...): Hier wordt een POST-verzoek gedaan naar het registratie-eindpunt. Als de registratie succesvol is, wordt een bericht weergegeven en wordt de gebruiker doorgestuurd naar de inlogpagina.

```
  logOut(): void {
    localStorage.clear();
    this.router.navigate(['/login']);
  }
}
```

logOut(): Deze methode wordt gebruikt om de huidige gebruiker af te melden. Het verwijdert alle opgeslagen gegevens uit de lokale opslag en leidt de gebruiker terug naar de inlogpagina.

De AuthService beheert de gebruikersauthenticatie en -registratie in de applicatie. Het maakt gebruik van Axios om HTTP-verzoeken te doen naar de backend-eindpunten en slaat gebruikersgegevens op in de lokale opslag voor toekomstig gebruik. Het is verantwoordelijk voor het verwerken van inlog- en registratieverzoeken, evenals het afmelden van gebruikers.

AuthGuard:

TypeScript Code (AuthGuard)

```
import { inject } from '@angular/core';
import { CanActivateFn, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTre
import { UtilsService } from './utils.service';

export const authGuard: CanActivateFn = (route: ActivatedRouteSnapshot, sta
  const router = inject(Router);
  const util = inject(UtilsService);

  const allowedRoles = route.data['allowedRoles'] as Array<string>;
```

authGuard: Dit is een aangepaste routebewaker (guard) die wordt gebruikt om te bepalen of een gebruiker toegang heeft tot een bepaalde route. Het is een pure functie die wordt uitgevoerd wanneer een route wordt geactiveerd. De routebewaker krijgt informatie over de huidige route en routerstaat.

inject(Router) en inject(UtilsService): Hier worden de Angular Router en UtilsService geïnjecteerd om toegang te krijgen tot router- en hulpprogrammafunctionaliteit binnen de bewaker.

allowedRoles: Dit is een array van rollen die zijn toegestaan om toegang te krijgen tot de route. De lijst van toegestane rollen wordt uit de routeconfiguratie opgehaald.

```
  // Special handling for login page
  if (route.routeConfig && (route.routeConfig.path === 'registration' ||
    const userRole = util.getUserRole();
    if (userRole) {
      // User has a role, redirect to home page
      if (userRole === 'ROLE_ADMIN') {
        router.navigate(['/admin/users']);
        return false;
      } else
      router.navigate(['']);
      return false;
    }
    // User has no role, allow access to the login page
    return true;
  }
```

Speciale behandeling voor de inlogpagina: Deze bewaker bevat speciale logica voor de inlogpagina ('login' en 'registration'). Als de gebruiker al is ingelogd (heeft een rol), wordt hij automatisch doorgestuurd naar de homepagina ('/') of de admingebruikerspagina ('/admin/users') op basis van zijn rol. Als de gebruiker nog geen rol heeft (niet is ingelogd), wordt toegang verleend tot de inlogpagina.

AuthGuard:

TypeScript Code (AuthGuard)

```
  // Handling for other pages
  const userRole = util.getUserRole();
  if (allowedRoles && allowedRoles.includes(userRole)) {
    return true;
  } else {
    // Navigate to a default route if the user doesn't have the necessary
    router.navigate(['/login']);
    return false;
  }
};
```

Behandeling voor andere pagina's: Voor andere pagina's wordt gecontroleerd of de gebruiker de vereiste rol heeft om toegang te krijgen tot de route. Als de gebruiker de juiste rol heeft, wordt toegang verleend (de bewaker geeft true terug). Als de gebruiker niet over de vereiste rol beschikt, wordt hij automatisch doorgestuurd naar de inlogpagina ('/login').

Deze authGuard is een krachtig hulpmiddel om toegangscontrole in je Angular-applicatie te beheren. Het behandelt speciale gevallen, zoals de inlogpagina, en zorgt ervoor dat gebruikers alleen toegang hebben tot pagina's die overeenkomen met hun rollen.