User Authentication and Authorization System Documentation

This document outlines the user authentication and authorization system for the ListKeeper application. It covers user registration, login, session management, and route protection.

1. Core Components and Services

The system is built around the following key files:

- `user.service.ts`: Handles all user-related API interactions, including login, logout, and registration. It also manages the current user's state.
- `auth.guard.ts`: A route guard that prevents unauthenticated users from accessing protected routes.
- `login.component.ts`: The component responsible for the user login form and logic.
- `user-status.component.ts`: Displays the user's login status and provides login/logout links.
- `user.model.ts`: The interface defining the `User` object.
- `app-routing.module.ts`: Configures the application's routes and applies the `AuthGuard`.

2. User Model

The `User` model is defined in `src/app/models/user.model.ts`.

```
export interface User {
    id: number;
    email: string;
    password?: string;
    role?: string;
    username?: string;
    firstname?: string;
    lastname?: string;
    phone?: string;
    createdAt?: Date;
    createdBy?: string;
    updatedAt?: Date;
    updatedBy?: string;
    deletedAt?: Date;
    deletedBy?: string;
    token?: string;
}
```

3. User Service (`user.service.ts`)

This service, located at `src/app/services/user.service.ts`, is the central point for managing user authentication.

- `currentUserSubject` **and** `currentUser`: A `BehaviorSubject` and `Observable` that store and expose the currently logged-in user. The user object is persisted in `localStorage` to maintain the session across browser refreshes.

- `login()`: Sends user credentials to the backend for authentication. Upon success, it stores the user object (including a JWT token) in `localStorage` and updates the `currentUserSubject`.
- `logout()`: Clears the user from `localStorage` and updates the `currentUserSubject` to `null`, effectively ending the session.
- `currentUserValue`: A getter that provides synchronous access to the current user object.

```
// src/app/services/user.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable, tap } from 'rxjs';
import { User } from '../models/user.model';
import { environment } from '../../environments/environment';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  private currentUserSubject: BehaviorSubject<User | null>;
  public currentUser: Observable<User | null>;
  private baseApiUrl = environment.baseApiUrl;

  constructor(private http: HttpClient) {
    const storedUser = localStorage.getItem('user');
    const parsedUser = storedUser ? JSON.parse(storedUser) : null;
    this.currentUserSubject = new BehaviorSubject<User | null>(parsedUser);
    this.currentUser = this.currentUserSubject.asObservable();
  }

  public get currentUserValue(): User | null {
    return this.currentUserSubject.value;
  }

  login(username: string, password: string) {
    return this.http.post<User>(`${this.baseApiUrl}/users/authenticate`, {
username, password })
      .pipe(tap(user => {
        localStorage.setItem('user', JSON.stringify(user));
        this.currentUserSubject.next(user);
        return user;
      }));
  }

  logout() {
    localStorage.removeItem('user');
    this.currentUserSubject.next(null);
  }
}
```

4. Route Protection (`auth.guard.ts`)

The `AuthGuard`, located at `src/app/guards/auth.guard.ts`, protects routes from unauthenticated access.

- **`canActivate()`**: This method is called by the Angular router before activating a route. It checks if a user is logged in by calling `this.userService.currentUserValue`.
    - If a user exists, it returns `true`, allowing access to the route.
    - If no user exists, it redirects the user to the home page (`''`) and returns `false`, preventing access.

```
// src/app/guards/auth.guard.ts
import { Injectable } from '@angular/core';
import { Router, CanActivate } from '@angular/router';
import { UserService } from '../services/user.service';

@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {
    constructor(
        private router: Router,
        private userService: UserService
    ) {}

    canActivate() {
        const currentUser = this.userService.currentUserValue;
        if (currentUser) {
            return true;
        }

        this.router.navigate(['']);
        return false;
    }
}
```

5. Routing Configuration (`app-routing.module.ts`)

The routing configuration in `src/app/app-routing.module.ts` defines which routes are protected.

- The `/notes` path is protected by the `AuthGuard`.
- The root path (`''`) displays the `HomeComponent` and is publicly accessible.
- The `/login` and `/signup` paths are also publicly accessible.

```
// src/app/app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './components/home/home.component';
import { NoteListComponent } from './components/notes/note-list/note-
list.component';
import { SignupComponent } from './components/users/signup/signup.component';
import { LoginComponent } from './components/users/login/login.component';
import { AuthGuard } from './guards/auth.guard';
```

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'notes', component: NoteListComponent, canActivate: [AuthGuard] },
  { path: 'signup', component: SignupComponent },
  { path: 'login', component: LoginComponent },
  { path: '**', redirectTo: '' }
];

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

6. Login Component (`login.component.ts`)

The `LoginComponent` at `src/app/components/users/login/login.component.ts` handles the user login process.

- It uses a reactive form (`loginForm`) to capture the username and password.
- On submission, it calls `userService.login()`.
- On successful login, it navigates the user to the `/notes` page.
- If login fails, it displays an error message.

```
// src/app/components/users/login/login.component.ts
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { UserService } from '../../../services/user.service';
import { first } from 'rxjs/operators';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
})
export class LoginComponent implements OnInit {
  loginForm!: FormGroup;
  error = '';

  constructor(
    private formBuilder: FormBuilder,
    private userService: UserService,
    private router: Router
  ) { }

  ngOnInit(): void {
    this.loginForm = this.formBuilder.group({
      username: ['', Validators.required],
      password: ['', Validators.required]
    });
  }
```

```
  get f() { return this.loginForm.controls; }

  onSubmit() {
    if (this.loginForm.invalid) {
      return;
    }

    this.userService.login(this.f['username'].value, this.f['password'].value)
      .pipe(first())
      .subscribe({
        next: () => {
          this.router.navigate(['/notes']);
        },
        error: () => {
          this.error = 'Login failed. Please check your credentials.';
        }
      });
  }
}
```

7. User Status Component (`user-status.component.ts`)

The `UserStatusComponent` at `src/app/components/users/user-status/user-status.component.ts` displays the current user's status.

- It subscribes to `userService.currentUser` to get the logged-in user's information.
- It displays a "Welcome" message and a "Logout" link if the user is logged in.
- If the user is not logged in, it displays a "Login" link, except on the home (`/`) and login (`/login`) pages.

```
// src/app/components/users/user-status/user-status.component.ts
import { Component, OnInit } from '@angular/core';
import { Router, NavigationEnd } from '@angular/router';
import { UserService } from '../../../services/user.service';
import { User } from '../../../models/user.model';
import { filter } from 'rxjs/operators';

@Component({
  selector: 'app-user-status',
  templateUrl: './user-status.component.html',
})
export class UserStatusComponent implements OnInit {
  user: User | null = null;
  currentRoute: string = '';

  constructor(
    private userService: UserService,
    private router: Router
  ) { }

  ngOnInit(): void {
```

```
    this.userService.currentUser.subscribe(user => {
      this.user = user;
    });

    this.router.events.pipe(
      filter(event => event instanceof NavigationEnd)
    ).subscribe((event: any) => {
      this.currentRoute = event.url;
    });
    this.currentRoute = this.router.url;
  }

  shouldShowLoginLink(): boolean {
    return this.currentRoute !== '/' && this.currentRoute !== '/login';
  }

  logout() {
    this.userService.logout();
    this.router.navigate(['']);
  }

  login() {
    this.router.navigate(['/login']);
  }
}
```

```
<!-- src/app/components/users/user-status/user-status.component.html -->
<div *ngIf="user; else loggedOut">
  Welcome {{ user.firstname }} {{ user.lastname }}! <a href="#"
(click)="logout()">Logout</a>
</div>
<ng-template #loggedOut>
  <a href="#" (click)="login()" *ngIf="shouldShowLoginLink()">Login</a>
</ng-template>
```