event-management-frontend/

├── .angular/

├── node\_modules/

├── src/

│ ├── app/

│ │ ├── auth/

│ │ │ ├── login/

│ │ │ │ ├── login.component.ts

│ │ │ │ ├── login.component.html

│ │ │ │ └── login.component.css

│ │ │ └── register/

│ │ │ ├── register.component.ts <-- Your register.component.ts

│ │ │ ├── register.component.html

│ │ │ └── register.component.css

│ │ ├── events/

│ │ │ ├── event-details/

│ │ │ ├── event-form/

│ │ │ └── event-list/

│ │ ├── feedback/

│ │ │ └── feedback-list/

│ │ ├── guards/

│ │ │ └── auth.guard.ts <-- Your auth.guard.ts

│ │ ├── home/

│ │ │ └── home/

│ │ ├── interceptors/

│ │ │ └── auth.interceptor.ts

│ │ ├── models/

│ │ │ ├── event.model.ts

│ │ │ ├── feedback.model.ts

│ │ │ ├── ticket.model.ts

│ │ │ └── user.model.ts <-- Your user.model.ts

│ │ ├── services/

│ │ │ ├── auth.service.ts <-- Your auth.service.ts

│ │ │ ├── event.service.ts

│ │ │ ├── feedback.service.ts

│ │ │ └── ticket.service.ts

│ │ ├── shared/

│ │ │ └── navbar/

│ │ ├── tickets/

│ │ │ ├── my-tickets/

│ │ │ │ ├── my-tickets.component.ts

│ │ │ │ ├── my-tickets.component.html

│ │ │ │ └── my-tickets.component.css

│ │ │ └── ticket-booking/

│ │ │ ├── ticket-booking.component.ts <-- Your ticket-booking.component.ts

│ │ │ ├── ticket-booking.component.html

│ │ │ └── ticket-booking.component.css

│ │ ├── app.component.ts

│ │ ├── app.component.html

│ │ ├── app.component.css

│ │ ├── app.module.ts <-- Your app.module.ts

│ │ └── app-routing.module.ts <-- Your app-routing.module.ts

│ ├── assets/

│ ├── environments/

│ ├── favicon.ico

│ ├── index.html

│ ├── main.ts

│ ├── polyfills.ts

│ ├── styles.css

│ └── test.ts

├── angular.json

├── package.json

├── tsconfig.json

└── ...

// src/app/auth/login/login.component.ts

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

import { Router } from '@angular/router';

import { AuthService } from '../../services/auth.service';

import { UserLogin } from '../../models/user.model';

@Component({

  selector: 'app-login',

  templateUrl: './login.component.html',

  standalone:false,

  styleUrls: ['./login.component.css']

})

export class LoginComponent {

  user: UserLogin = { name: '', password: '' };

  errorMessage: string = '';

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

  onLogin(): void {

    this.errorMessage = ''; // Clear previous errors

    this.authService.login(this.user).subscribe({

      next: (token) => {

        console.log('Login successful:', token);

        // After login, navigate to a secured page, e.g., events list or home

        this.router.navigate(['/events']);

      },

      error: (err) => {

        console.error('Login failed:', err);

        this.errorMessage = 'Login failed. Please check your username and password.';

        // More detailed error handling could be added based on backend response

      }

    });

  }

}

<div class="container mt-5">

    <div class="row justify-content-center">

      <div class="col-md-6">

        <div class="card shadow-lg">

          <div class="card-header bg-primary text-white text-center">

            <h3>Login to Eventify</h3>

          </div>

          <div class="card-body p-4">

            <form (ngSubmit)="onLogin()">

              <div class="mb-3">

                <label for="username" class="form-label">Username</label>

                <input type="text" class="form-control" id="username" name="name" [(ngModel)]="user.name" required>

              </div>

              <div class="mb-3">

                <label for="password" class="form-label">Password</label>

                <input type="password" class="form-control" id="password" name="password" [(ngModel)]="user.password" required>

              </div>

              <div \*ngIf="errorMessage" class="alert alert-danger" role="alert">

                {{ errorMessage }}

              </div>

              <div class="d-grid gap-2">

                <button type="submit" class="btn btn-primary btn-lg">Login</button>

              </div>

            </form>

            <div class="text-center mt-3">

              <p>Don't have an account? <a routerLink="/register">Register here</a></p>

            </div>

          </div>

        </div>

      </div>

    </div>

  </div>

// src/app/auth/register/register.component.ts

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

import { Router } from '@angular/router';

import { AuthService } from '../../services/auth.service';

import { UserRegister } from '../../models/user.model';

@Component({

  selector: 'app-register',

  templateUrl: './register.component.html',

  standalone:false,

  styleUrls: ['./register.component.css']

})

export class RegisterComponent implements OnInit {

  // Initialize user registration data with empty strings

  user: UserRegister = {

    name: '', // Changed from firstName and lastName

    email: '',

    password: '',

    contactNumber: '',

    role: 'USER' // Default role for new registrations

  };

  confirmPassword = ''; // Field for password confirmation

  errorMessage: string = ''; // Message to display for errors

  successMessage: string = ''; // Message to display for success

  isLoading: boolean = false; // Flag to indicate if registration is in progress

  constructor(

    private authService: AuthService, // Inject the AuthService

    private router: Router // Inject the Router for navigation

  ) { }

  ngOnInit(): void {

    // Initialization logic if needed

  }

  // Handles the registration form submission

  onRegister(): void {

    this.errorMessage = '';    // Clear previous error messages

    this.successMessage = '';  // Clear previous success messages

    // Client-side validation: Check if passwords match

    if (this.user.password !== this.confirmPassword) {

      this.errorMessage = 'Passwords do not match.';

      return; // Stop execution if passwords don't match

    }

    this.isLoading = true; // Show loading spinner

    // Call the register method from AuthService

    this.authService.register(this.user).subscribe({

      next: (response) => {

        this.successMessage = 'Registration successful! You can now log in.';

        this.isLoading = false;

        // Redirect to login page after a short delay for user to see success message

        setTimeout(() => {

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

        }, 2000);

      },

      error: (error) => {

        // Log the detailed error for debugging

        console.error('Registration failed:', error);

        // Display the user-friendly error message from AuthService's handleError

        this.errorMessage = error.message || 'Registration failed. Please try again.';

        this.isLoading = false; // Hide loading spinner

      }

    });

  }

}

<div class="container mt-5">

    <div class="row justify-content-center">

      <div class="col-md-6">

        <div class="card shadow-lg">

          <div class="card-header bg-primary text-white text-center">

            <h3>Register</h3>

          </div>

          <div class="card-body p-4">

            <form (ngSubmit)="onRegister()">

              <div \*ngIf="errorMessage" class="alert alert-danger" role="alert">

                {{ errorMessage }}

              </div>

              <div \*ngIf="successMessage" class="alert alert-success" role="alert">

                {{ successMessage }}

              </div>

              <div class="mb-3">

                <label for="name" class="form-label">Name</label>

                <input type="text" class="form-control" id="name" name="name" [(ngModel)]="user.name" required>

              </div>

              <div class="mb-3">

                <label for="email" class="form-label">Email address</label>

                <input type="email" class="form-control" id="email" name="email" [(ngModel)]="user.email" required>

              </div>

              <div class="mb-3">

                <label for="contactNumber" class="form-label">Contact Number</label>

                <input type="text" class="form-control" id="contactNumber" name="contactNumber" [(ngModel)]="user.contactNumber" required pattern="[0-9]{10}" title="Contact number must be 10 digits">

              </div>

              <div class="mb-3">

                <label for="password" class="form-label">Password</label>

                <input type="password" class="form-control" id="password" name="password" [(ngModel)]="user.password" required minlength="8">

                <div class="form-text">Password must be at least 8 characters long.</div>

              </div>

              <div class="mb-3">

                <label for="confirmPassword" class="form-label">Confirm Password</label>

                <input type="password" class="form-control" id="confirmPassword" name="confirmPassword" [(ngModel)]="confirmPassword" required>

              </div>

              <div class="mb-3">

                <label for="role" class="form-label">Role</label>

                <select class="form-select" id="role" name="role" [(ngModel)]="user.role" required>

                  <option value="USER">User</option>

                  <option value="ADMIN">Admin</option>

                </select>

              </div>

              <button type="submit" class="btn btn-primary w-100" [disabled]="isLoading">

                <span \*ngIf="isLoading" class="spinner-border spinner-border-sm me-2" role="status" aria-hidden="true"></span>

                Register

              </button>

            </form>

            <div class="text-center mt-3">

              Already have an account? <a routerLink="/login">Login here</a>

            </div>

          </div>

        </div>

      </div>

    </div>

  </div>

<div class="container mt-4">

    <div \*ngIf="event">

      <div class="card shadow-lg mb-4">

        <div class="card-header bg-primary text-white">

          <h1 class="mb-0">{{ event.name }}</h1>

        </div>

        <div class="card-body">

          <p class="card-text"><strong>Category:</strong> {{ event.category }}</p>

          <p class="card-text"><strong>Location:</strong> {{ event.location }}</p>

          <p class="card-text"><strong>Date:</strong> {{ event.date }}</p>

          <p class="card-text"><strong>Organizer ID:</strong> {{ event.organizerId }}</p>

          <p class="card-text">

            <strong>Tickets Available:</strong>

            <span [class.text-danger]="event.ticketCount <= 10" [class.text-warning]="event.ticketCount > 10 && event.ticketCount <= 50">

              {{ event.ticketCount }}

            </span>

          </p>

          <hr>

          <h4>Actions</h4>

          <div class="d-flex gap-2">

            <button class="btn btn-success" (click)="onBookTicket()" [disabled]="!userId || event.ticketCount <= 0">

              Book Ticket

            </button>

            <button class="btn btn-secondary" routerLink="/events">Back to Events</button>

          </div>

          <div \*ngIf="bookingMessage" class="alert alert-info mt-3" role="alert">

            {{ bookingMessage }}

          </div>

        </div>

      </div>

      <div class="card shadow-lg mb-4">

        <div class="card-header bg-info text-white">

          <h3 class="mb-0">Event Feedback</h3>

        </div>

        <div class="card-body">

          <div \*ngIf="averageRating !== null && averageRating > 0">

            <p class="lead"><strong>Average Rating:</strong> {{ averageRating | number:'1.1-1' }} / 5</p>

          </div>

          <div \*ngIf="feedbackList.length > 0">

            <h5>All Feedback:</h5>

            <ul class="list-group">

              <li class="list-group-item" \*ngFor="let feedback of feedbackList">

                <strong>Rating:</strong> {{ feedback.rating }} / 5 <br>

                <strong>Comments:</strong> {{ feedback.comments }} <br>

                <small class="text-muted">Submitted: {{ feedback.submittedTimestamp | date:'medium' }}</small>

              </li>

            </ul>

          </div>

          <div \*ngIf="feedbackList.length === 0 && averageRating === 0">

            <p>No feedback yet. Be the first to leave a review!</p>

          </div>

          <hr>

          <h5>Submit Your Feedback</h5>

          <form (ngSubmit)="onSubmitFeedback()" class="mt-3">

            <div class="mb-3">

              <label class="form-label">Rating:</label>

              <div>

                <ng-container \*ngFor="let star of [1, 2, 3, 4, 5]">

                  <i class="bi bi-star-fill"

                     [class.text-warning]="newFeedback.rating >= star"

                     [class.text-muted]="newFeedback.rating < star"

                     (click)="setRating(star)"

                     style="cursor: pointer; font-size: 1.5rem;"></i>

                </ng-container>

              </div>

            </div>

            <div class="mb-3">

              <label for="comments" class="form-label">Comments:</label>

              <textarea id="comments" name="comments" [(ngModel)]="newFeedback.comments" class="form-control" rows="3" required></textarea>

            </div>

            <button type="submit" class="btn btn-primary" [disabled]="!newFeedback.rating || !newFeedback.comments.trim() || !userId">Submit Feedback</button>

            <div \*ngIf="!userId" class="alert alert-warning mt-2">

              Please log in to submit feedback.

            </div>

          </form>

        </div>

      </div>

    </div>

    <div \*ngIf="!event" class="alert alert-warning" role="alert">

      Loading event details or event not found...

    </div>

  </div>

// src/app/events/event-details/event-details.component.ts

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

import { ActivatedRoute, Router } from '@angular/router';

import { EventService } from '../../services/event.service';

import { Event } from '../../models/event.model';

import { FeedbackService } from '../../services/feedback.service';

import { Feedback } from '../../models/feedback.model';

import { AuthService } from '../../services/auth.service';

import { TicketService } from '../../services/ticket.service';

@Component({

  selector: 'app-event-details',

  templateUrl: './event-details.component.html',

  standalone:false,

  styleUrls: ['./event-details.component.css']

})

export class EventDetailsComponent implements OnInit {

  event: Event | undefined;

  eventId: number | null = null;

  feedbackList: Feedback[] = [];

  averageRating: number | null = null;

  newFeedback: Feedback = { eventId: 0, userId: 0, rating: 0, comments: '' };

  userId: number | null = null;

  userRole: string | null = null;

  bookingMessage: string = '';

  constructor(

    private route: ActivatedRoute,

    private router: Router,

    private eventService: EventService,

    private feedbackService: FeedbackService,

    private authService: AuthService,

    private ticketService: TicketService

  ) { }

  ngOnInit(): void {

    this.eventId = Number(this.route.snapshot.paramMap.get('id'));

    if (this.eventId) {

      this.loadEventDetails();

      this.loadFeedback();

      this.authService.getCurrentUserId().subscribe(id => this.userId = id);

      this.authService.getUserRole().subscribe(role => this.userRole = role);

      this.newFeedback.eventId = this.eventId;

      // Set userId for feedback if logged in

      this.authService.getCurrentUserId().subscribe(id => {

        if (id) {

          this.newFeedback.userId = id;

        }

      });

    }

  }

  loadEventDetails(): void {

    if (this.eventId) {

      this.eventService.getEventById(this.eventId).subscribe({  //[cite: 11]

        next: (data) => {

          this.event = data;

        },

        error: (err) => {

          console.error('Error fetching event details:', err);

          alert('Event not found or unauthorized access.');

          this.router.navigate(['/events']);

        }

      });

    }

  }

  loadFeedback(): void {

    if (this.eventId) {

      this.feedbackService.getFeedbackByEventId(this.eventId).subscribe({ //[cite: 19]

        next: (data) => {

          this.feedbackList = data;

        },

        error: (err) => {

          console.error('Error fetching feedback:', err);

        }

      });

      this.feedbackService.getAverageRatingByEventId(this.eventId).subscribe({  //[cite: 20]

        next: (avg) => {

          this.averageRating = avg;

        },

        error: (err) => {

          console.error('Error fetching average rating:', err);

        }

      });

    }

  }

  onSubmitFeedback(): void {

    if (this.newFeedback.rating === 0 || !this.newFeedback.comments.trim()) {

      alert('Please provide a rating and comments.');

      return;

    }

    if (!this.userId) {

      alert('You must be logged in to submit feedback.');

      return;

    }

    this.newFeedback.userId = this.userId; // Ensure userId is set before sending

    this.feedbackService.submitFeedback(this.newFeedback).subscribe({   //[cite: 18]

      next: (feedback) => {

        console.log('Feedback submitted:', feedback);

        this.newFeedback.comments = ''; // Clear comments

        this.newFeedback.rating = 0;   // Reset rating

        this.loadFeedback();           // Reload feedback list

        alert('Feedback submitted successfully!');

      },

      error: (err) => {

        console.error('Error submitting feedback:', err);

        alert('Failed to submit feedback. Please try again.');

      }

    });

  }

  onBookTicket(): void {

    if (!this.userId) {

      alert('You must be logged in to book a ticket.');

      return;

    }

    if (!this.eventId) {

      alert('Event ID is missing.');

      return;

    }

    this.bookingMessage = ''; // Clear previous messages

    this.ticketService.bookTicket(this.eventId, this.userId).subscribe({ //[cite: 35]

      next: (ticket) => {

        this.bookingMessage = `Ticket booked successfully! Ticket ID: ${ticket.id}. Status: ${ticket.status}`;

        this.loadEventDetails(); // Refresh ticket count

      },

      error: (err) => {

        console.error('Error booking ticket:', err);

        this.bookingMessage = `Failed to book ticket: ${err.error || 'Please try again.'}`;

        alert('Failed to book ticket. Check console for details (e.g., no tickets available).');

      }

    });

  }

  setRating(rating: number): void {

    this.newFeedback.rating = rating;

  }

}

<div class="container mt-4">

    <div \*ngIf="canManageEvents(userRole$ | async); else noPermission">

      <h2>{{ isEditMode ? 'Edit Event' : 'Create New Event' }}</h2>

      <div \*ngIf="errorMessage" class="alert alert-danger" role="alert">

        {{ errorMessage }}

      </div>

      <div \*ngIf="successMessage" class="alert alert-success" role="alert">

        {{ successMessage }}

      </div>

      <form (ngSubmit)="onSubmit()" class="needs-validation" novalidate>

        <div class="mb-3">

          <label for="name" class="form-label">Event Name:</label>

          <input type="text" id="name" name="name" [(ngModel)]="event.name" class="form-control" required>

        </div>

        <div class="mb-3">

          <label for="category" class="form-label">Category:</label>

          <input type="text" id="category" name="category" [(ngModel)]="event.category" class="form-control" required>

        </div>

        <div class="mb-3">

          <label for="location" class="form-label">Location:</label>

          <input type="text" id="location" name="location" [(ngModel)]="event.location" class="form-control" required>

        </div>

        <div class="mb-3">

          <label for="date" class="form-label">Date:</label>

          <input type="date" id="date" name="date" [(ngModel)]="event.date" class="form-control" required>

        </div>

        <div class="mb-3">

          <label for="organizerId" class="form-label">Organizer ID:</label>

          <input type="number" id="organizerId" name="organizerId" [(ngModel)]="event.organizerId" class="form-control" required>

          <div class="form-text">This should be the ID of the user acting as the organizer.</div>

        </div>

        <div class="mb-3">

          <label for="ticketCount" class="form-label">Ticket Count:</label>

          <input type="number" id="ticketCount" name="ticketCount" [(ngModel)]="event.ticketCount" class="form-control" required min="1">

        </div>

        <button type="submit" class="btn btn-primary me-2">{{ isEditMode ? 'Update Event' : 'Create Event' }}</button>

        <button type="button" class="btn btn-secondary" routerLink="/events">Cancel</button>

      </form>

    </div>

    <ng-template #noPermission>

      <div class="alert alert-danger" role="alert">

        You do not have permission to {{ isEditMode ? 'edit' : 'create' }} events. Only ADMINs and ORGANIZERs can manage events.

      </div>

      <button class="btn btn-secondary" routerLink="/events">Back to Events</button>

    </ng-template>

  </div>

// src/app/events/event-form/event-form.component.ts

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

import { EventService } from '../../services/event.service';

import { ActivatedRoute, Router } from '@angular/router';

import { Event } from '../../models/event.model';

import { AuthService } from '../../services/auth.service';

import { Observable } from 'rxjs';

@Component({

  selector: 'app-event-form',

  templateUrl: './event-form.component.html',

  standalone:false,

  styleUrls: ['./event-form.component.css']

})

export class EventFormComponent implements OnInit {

  event: Event = { name: '', category: '', location: '', date: '', organizerId: 0, ticketCount: 0 };

  isEditMode: boolean = false;

  eventId: number | null = null;

  errorMessage: string = '';

  successMessage: string = '';

  userRole$: Observable<string | null>;

  constructor(

    private eventService: EventService,

    private route: ActivatedRoute,

    private router: Router,

    private authService: AuthService

  ) {

    this.userRole$ = this.authService.getUserRole();

  }

  ngOnInit(): void {

    this.eventId = Number(this.route.snapshot.paramMap.get('id'));

    if (this.eventId) {

      this.isEditMode = true;

      this.eventService.getEventById(this.eventId).subscribe({  //[cite: 11]

        next: (data) => {

          this.event = data;

        },

        error: (err) => {

          console.error('Error fetching event for edit:', err);

          alert('Failed to load event for editing or unauthorized. Redirecting to events list.');

          this.router.navigate(['/events']);

        }

      });

    } else {

      // For new events, set organizerId to current user's ID if available

      this.authService.getCurrentUserId().subscribe(id => {

        if (id) {

          this.event.organizerId = id;

        } else {

          // If no user ID, prompt or handle appropriately

          console.warn('User ID not found for new event, organizerId might be invalid.');

        }

      });

    }

  }

  onSubmit(): void {

    this.errorMessage = '';

    this.successMessage = '';

    // Basic client-side validation, matching backend [cite: 159, 160, 161, 162, 163, 164]

    if (!this.event.name || !this.event.category || !this.event.location || !this.event.date || !this.event.organizerId || this.event.ticketCount <= 0) {

      this.errorMessage = 'All fields are required and Ticket Count must be positive.';

      return;

    }

    if (this.isEditMode && this.eventId) {

      this.eventService.updateEvent(this.eventId, this.event).subscribe({ //[cite: 14]

        next: () => {

          this.successMessage = 'Event updated successfully!';

          setTimeout(() => this.router.navigate(['/events']), 1500);

        },

        error: (err) => {

          console.error('Error updating event:', err);

          this.errorMessage = 'Failed to update event: ' + (err.error || 'Check permissions or input.');

        }

      });

    } else {

      this.eventService.createEvent(this.event).subscribe({ //[cite: 13]

        next: () => {

          this.successMessage = 'Event created successfully!';

          this.event = { name: '', category: '', location: '', date: '', organizerId: this.authService.getCurrentUserId() as any, ticketCount: 0 }; // Reset form

          setTimeout(() => this.router.navigate(['/events']), 1500);

        },

        error: (err) => {

          console.error('Error creating event:', err);

          this.errorMessage = 'Failed to create event: ' + (err.error || 'Check permissions or input.');

        }

      });

    }

  }

  // Helper to check if current user is ADMIN or ORGANIZER

  canManageEvents(role: string | null): boolean {

    return role === 'ADMIN' || role === 'ORGANIZER'; //[cite: 96, 97]

  }

}

<div class="container mt-4">

    <div class="d-flex justify-content-between align-items-center mb-3">

      <h2>All Events</h2>

      <button \*ngIf="canManageEvents(userRole$ | async)" class="btn btn-primary" routerLink="/events/new">Add New Event</button>

    </div>

    <div \*ngIf="events.length === 0" class="alert alert-info" role="alert">

      No events found.

    </div>

    <div class="row row-cols-1 row-cols-md-2 row-cols-lg-3 g-4">

      <div class="col" \*ngFor="let event of events">

        <div class="card h-100 shadow-sm">

          <div class="card-body">

            <h5 class="card-title text-primary">{{ event.name }}</h5>

            <h6 class="card-subtitle mb-2 text-muted">{{ event.category }}</h6>

            <p class="card-text">

              <strong>Location:</strong> {{ event.location }}<br>

              <strong>Date:</strong> {{ event.date }}<br>

              <strong>Tickets Remaining:</strong> {{ event.ticketCount }}

            </p>

            <div class="d-flex justify-content-between align-items-center">

              <a routerLink="/events/{{ event.id }}" class="btn btn-info btn-sm">View Details</a>

              <div \*ngIf="canManageEvents(userRole$ | async)">

                <a routerLink="/events/edit/{{ event.id }}" class="btn btn-warning btn-sm me-2">Edit</a>

                <button \*ngIf="canDeleteEvent(userRole$ | async)" class="btn btn-danger btn-sm" (click)="onDeleteEvent(event.id)">Delete</button>

              </div>

            </div>

          </div>

        </div>

      </div>

    </div>

  </div>

// src/app/events/event-list/event-list.component.ts

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

import { EventService } from '../../services/event.service';

import { Event } from '../../models/event.model';

import { AuthService } from '../../services/auth.service';

import { Observable, combineLatest, map } from 'rxjs';

@Component({

  selector: 'app-event-list',

  templateUrl: './event-list.component.html',

  standalone:false,

  styleUrls: ['./event-list.component.css']

})

export class EventListComponent implements OnInit {

  events: Event[] = [];

  userRole$: Observable<string | null>;

  constructor(private eventService: EventService, private authService: AuthService) {

    this.userRole$ = this.authService.getUserRole();

  }

  ngOnInit(): void {

    this.loadEvents();

  }

  loadEvents(): void {

    this.eventService.getAllEvents().subscribe({ //[cite: 10]

      next: (data) => {

        this.events = data;

      },

      error: (err) => {

        console.error('Error fetching events:', err);

        // Handle unauthorized access or other errors

      }

    });

  }

  onDeleteEvent(id: number | undefined): void {

    if (id && confirm('Are you sure you want to delete this event?')) {

      this.eventService.deleteEvent(id).subscribe({ // [cite: 15]

        next: () => {

          console.log('Event deleted successfully');

          this.loadEvents(); // Refresh the list

        },

        error: (err) => {

          console.error('Error deleting event:', err);

          alert('Failed to delete event. You might not have the necessary permissions (ADMIN only).');

        }

      });

    }

  }

  // Helper to check if current user is ADMIN or ORGANIZER

  canManageEvents(role: string | null): boolean {

    return role === 'ADMIN' || role === 'ORGANIZER'; //[cite: 96, 97]

  }

  // Helper to check if current user is ADMIN

  canDeleteEvent(role: string | null): boolean {

    return role === 'ADMIN'; // [cite: 96]

  }

}

<div class="container mt-4">

    <div \*ngIf="canViewFeedback(userRole$ | async); else noPermission">

      <h2>Feedback List</h2>

      <div \*ngIf="errorMessage" class="alert alert-danger" role="alert">

        {{ errorMessage }}

      </div>

      <div class="mb-3">

        <label for="eventSelect" class="form-label">Select Event to view feedback:</label>

        <select id="eventSelect" class="form-select" [(ngModel)]="selectedEventId" (change)="onEventSelect()">

          <option [ngValue]="null">-- Select an Event --</option>

          <option \*ngFor="let event of events" [ngValue]="event.id">{{ event.name }}</option>

        </select>

      </div>

      <div \*ngIf="feedbackList.length > 0">

        <h4>Feedback for {{ getEventName(selectedEventId!) }}</h4>

        <table class="table table-striped table-bordered mt-3">

          <thead>

            <tr>

              <th>Rating</th>

              <th>Comments</th>

              <th>Submitted On</th>

              </tr>

          </thead>

          <tbody>

            <tr \*ngFor="let feedback of feedbackList">

              <td>{{ feedback.rating }} / 5</td>

              <td>{{ feedback.comments }}</td>

              <td>{{ feedback.submittedTimestamp | date:'medium' }}</td>

            </tr>

          </tbody>

        </table>

      </div>

      <div \*ngIf="selectedEventId && feedbackList.length === 0 && !errorMessage" class="alert alert-info" role="alert">

        No feedback available for this event yet.

      </div>

      <div \*ngIf="!selectedEventId && !errorMessage" class="alert alert-info" role="alert">

        Please select an event to view its feedback.

      </div>

    </div>

    <ng-template #noPermission>

      <div class="alert alert-danger" role="alert">

        You do not have permission to view feedback. Only ADMINs and ORGANIZERs can view feedback.

      </div>

      <button class="btn btn-secondary" routerLink="/events">Back to Events</button>

    </ng-template>

  </div>

// src/app/feedback/feedback-list/feedback-list.component.ts

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

import { FeedbackService } from '../../services/feedback.service';

import { Feedback } from '../../models/feedback.model';

import { EventService } from '../../services/event.service';

import { Event } from '../../models/event.model';

import { combineLatest, map, Observable, switchMap } from 'rxjs';

import { AuthService } from '../../services/auth.service';

@Component({

  selector: 'app-feedback-list',

  templateUrl: './feedback-list.component.html',

  standalone:false,

  styleUrls: ['./feedback-list.component.css']

})

export class FeedbackListComponent implements OnInit {

  feedbackList: Feedback[] = [];

  events: Event[] = [];

  selectedEventId: number | null = null;

  errorMessage: string = '';

  userRole$: Observable<string | null>;

  constructor(

    private feedbackService: FeedbackService,

    private eventService: EventService,

    private authService: AuthService

  ) {

    this.userRole$ = this.authService.getUserRole();

  }

  ngOnInit(): void {

    this.loadEventsForFilter();

    this.userRole$.subscribe(role => {

      if (role === 'ADMIN' || role === 'ORGANIZER') {

        // Initially load all feedback if no event is selected (not directly supported by API)

        // Or you might want to show a message to select an event

        // For now, we'll only load feedback when an event is selected

      } else {

        this.errorMessage = 'You do not have permission to view feedback list.';

      }

    });

  }

  loadEventsForFilter(): void {

    this.eventService.getAllEvents().subscribe({  //[cite: 10]

      next: (data) => {

        this.events = data;

      },

      error: (err) => {

        console.error('Error loading events for filter:', err);

        this.errorMessage = 'Could not load events for filtering feedback.';

      }

    });

  }

  onEventSelect(): void {

    this.feedbackList = []; // Clear previous list

    if (this.selectedEventId) {

      this.feedbackService.getFeedbackByEventId(this.selectedEventId).subscribe({   //[cite: 19]

        next: (data) => {

          this.feedbackList = data;

        },

        error: (err) => {

          console.error('Error fetching feedback for event:', err);

          this.errorMessage = 'Failed to load feedback for the selected event.';

        }

      });

    }

  }

  getEventName(eventId: number): string {

    const event = this.events.find(e => e.id === eventId);

    return event ? event.name : 'Unknown Event';

  }

  canViewFeedback(role: string | null): boolean {

    return role === 'ADMIN' || role === 'ORGANIZER'; //[cite: 96, 97]

  }

}

// src/app/guards/auth.guard.ts

import { Injectable } from '@angular/core';

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

import { Observable, take, map } from 'rxjs';

import { AuthService } from '../services/auth.service';

@Injectable({

  providedIn: 'root'

})

export class AuthGuard implements CanActivate {

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

  canActivate(

    route: ActivatedRouteSnapshot,

    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

    return this.authService.isLoggedIn().pipe(

      take(1), // Take the first value and complete

      map(isLoggedIn => {

        if (isLoggedIn) {

          // Optional: Implement role-based access here if needed

          // const expectedRole = route.data['role']; // Define data property in routes

          // const userRole = this.authService.getUserRole();

          // if (expectedRole && userRole !== expectedRole) {

          //   return this.router.createUrlTree(['/unauthorized']);

          // }

          return true;

        } else {

          // Not logged in, redirect to login page

          return this.router.createUrlTree(['/login']);

        }

      })

    );

  }

}

<div class="jumbotron jumbotron-fluid text-center bg-light p-5 my-4 rounded">

    <div class="container">

      <h1 class="display-4">Welcome to Eventify!</h1>

      <p class="lead">Your one-stop solution for managing and participating in events.</p>

      <hr class="my-4">

      <ng-container \*ngIf="!(isLoggedIn$ | async)">

        <p>Please log in or register to get started.</p>

        <a class="btn btn-primary btn-lg me-2" routerLink="/login" role="button">Login</a>

        <a class="btn btn-success btn-lg" routerLink="/register" role="button">Register</a>

      </ng-container>

      <ng-container \*ngIf="isLoggedIn$ | async">

        <p>You are logged in as a {{ userRole$ | async }}.</p>

        <p>Explore events or manage your activities.</p>

        <a class="btn btn-info btn-lg" routerLink="/events" role="button">View Events</a>

      </ng-container>

    </div>

  </div>

// src/app/home/home/home.component.ts

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

import { AuthService } from '../../services/auth.service';

import { Observable } from 'rxjs';

@Component({

  selector: 'app-home',

  templateUrl: './home.component.html',

  standalone:false,

  styleUrls: ['./home.component.css']

})

export class HomeComponent implements OnInit {

  isLoggedIn$: Observable<boolean>;

  userRole$: Observable<string | null>;

  constructor(private authService: AuthService) {

    this.isLoggedIn$ = this.authService.isLoggedIn();

    this.userRole$ = this.authService.getUserRole();

  }

  ngOnInit(): void { }

}

export interface Event {

    id?: number;

    name: string;

    category: string;

    location: string;

    date: string; // Using string for date (YYYY-MM-DD) to match Java's java.sql.Date

    organizerId: number;

    ticketCount: number;

  }

import { User } from "./user.model";

export interface Feedback {

    id?: number;

    eventId: number;

    userId: number;

    rating: number; // 1-5

    comments: string;

    submittedTimestamp?: string; // Using string for timestamp (ISO 8601)

    // Optionally, if you want to display event/user details in feedback list:

    event?: Event; // Nested event object if API returns it

    user?: User;   // Nested user object if API returns it

  }

export interface Notification {

    id?: number;

    userId: number;

    eventId: number;

    message: string;

    sentTimestamp?: string;

  }

import { User } from "./user.model";

export interface Ticket {

    id?: number;

    eventId: number;

    userId: number;

    bookingDate?: string; // Using string for date

    status: string; // e.g., "Confirmed", "Cancelled"

    // Optionally, if you want to display event/user details in ticket list:

    event?: Event;

    user?: User;

  }

// src/app/models/user.model.ts

// Represents the full User object received from the backend

export interface User {

  id: number;

  name: string; // Changed from firstName and lastName

  email: string;

  contactNumber: string;

  role: string;

}

// Represents the data structure for user registration requests

export interface UserRegister {

  name: string; // Changed from firstName and lastName

  email: string;

  password?: string;

  contactNumber: string;

  role: string;

}

// Represents the data structure for user login requests

export interface UserLogin {

  name: string;

  password: string;

}

// src/app/services/auth.service.ts

import { Injectable, PLATFORM\_ID, Inject } from '@angular/core';

import { isPlatformBrowser } from '@angular/common';

import { HttpClient, HttpErrorResponse } from '@angular/common/http'; // Import HttpErrorResponse

import { Observable, BehaviorSubject, tap, throwError } from 'rxjs'; // Import throwError

import { catchError } from 'rxjs/operators'; // Import catchError

import { User, UserLogin, UserRegister } from '../models/user.model';

import { Router } from '@angular/router';

@Injectable({

  providedIn: 'root'

})

export class AuthService {

  private baseUrl = 'http://localhost:2061/user'; // Base URL for User Controller

  private tokenKey = 'jwt\_token';

  private loggedIn = new BehaviorSubject<boolean>(false);

  private userRole = new BehaviorSubject<string | null>(null);

  private currentUserId = new BehaviorSubject<number | null>(null);

  private isBrowser: boolean;

  constructor(

    private http: HttpClient,

    private router: Router,

    @Inject(PLATFORM\_ID) private platformId: Object

  ) {

    this.isBrowser = isPlatformBrowser(this.platformId);

    if (this.isBrowser) {

      this.loggedIn.next(this.hasToken());

      this.userRole.next(localStorage.getItem('user\_role'));

      this.currentUserId.next(this.getUserIdFromToken());

    }

  }

  private hasToken(): boolean {

    if (this.isBrowser) {

      return !!localStorage.getItem(this.tokenKey);

    }

    return false;

  }

  isLoggedIn(): Observable<boolean> {

    return this.loggedIn.asObservable();

  }

  getUserRole(): Observable<string | null> {

    return this.userRole.asObservable();

  }

  getCurrentUserId(): Observable<number | null> {

    return this.currentUserId.asObservable();

  }

  getToken(): string | null {

    if (this.isBrowser) {

      return localStorage.getItem(this.tokenKey);

    }

    return null;

  }

  private setToken(token: string): void {

    if (this.isBrowser) {

      localStorage.setItem(this.tokenKey, token);

      const decodedToken = this.decodeToken(token);

      const role = decodedToken?.role || null;

      const userId = decodedToken?.userId || null;

      localStorage.setItem('user\_role', role);

      localStorage.setItem('user\_id', userId);

      this.userRole.next(role);

      this.currentUserId.next(userId);

      this.loggedIn.next(true);

    }

  }

  private removeToken(): void {

    if (this.isBrowser) {

      localStorage.removeItem(this.tokenKey);

      localStorage.removeItem('user\_role');

      localStorage.removeItem('user\_id');

      this.userRole.next(null);

      this.currentUserId.next(null);

      this.loggedIn.next(false);

    }

  }

  register(userData: UserRegister): Observable<User> {

    return this.http.post<User>(`${this.baseUrl}/register`, userData).pipe(

      catchError(this.handleError) // Add error handling

    );

  }

  login(credentials: UserLogin): Observable<string> {

    return this.http.post(`${this.baseUrl}/login`, credentials, { responseType: 'text' }).pipe(

      tap((token: string) => {

        this.setToken(token);

        console.log('Login successful, token stored.');

      }),

      catchError(this.handleError) // Add error handling for login as well

    );

  }

  logout(): void {

    this.removeToken();

    this.router.navigate(['/login']);

    console.log('Logged out.');

  }

  private decodeToken(token: string): any {

    if (this.isBrowser && token) {

      try {

        const base64Url = token.split('.')[1];

        const base64 = base64Url.replace(/-/g, '+').replace(/\_/g, '/');

        const jsonPayload = decodeURIComponent(atob(base64).split('').map(function(c) {

          return '%' + ('00' + c.charCodeAt(0).toString(16)).slice(-2);

        }).join(''));

        return JSON.parse(jsonPayload);

      } catch (e) {

        console.error('Error decoding token:', e);

        return null;

      }

    }

    return null;

  }

  private getUserIdFromToken(): number | null {

    const token = this.getToken();

    if (token) {

      const decoded = this.decodeToken(token);

      return decoded?.userId || null;

    }

    return null;

  }

  // New error handling method

  private handleError(error: HttpErrorResponse) {

    let errorMessage = 'An unknown error occurred!';

    if (error.error instanceof ErrorEvent) {

      // Client-side or network error occurred. Handle it accordingly.

      errorMessage = `A client-side error occurred: ${error.error.message}`;

    } else {

      // The backend returned an unsuccessful response code.

      // The response body may contain clues as to what went wrong.

      console.error(

        `Backend returned code ${error.status}, ` +

        `body was: ${JSON.stringify(error.error)}`);

      if (error.error && typeof error.error === 'string') {

        // If the backend sends a plain string error message

        errorMessage = error.error;

      } else if (error.error && error.error.message) {

        // If the backend sends an error object with a 'message' field

        errorMessage = error.error.message;

      } else if (error.status === 400) {

        errorMessage = 'Invalid input provided. Please check your data.';

      } else if (error.status === 409) {

        errorMessage = 'User with this email already exists.';

      } else if (error.status === 500) {

        errorMessage = 'Server error. Please try again later.';

      }

    }

    // Return an observable with a user-facing error message.

    return throwError(() => new Error(errorMessage));

  }

}

// src/app/services/event.service.ts

import { Injectable } from '@angular/core';

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Event } from '../models/event.model';

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

@Injectable({

  providedIn: 'root'

})

export class EventService {

  private baseUrl = 'http://localhost:2061/events'; // Base URL for Event Controller [cite: 9]

  constructor(private http: HttpClient, private authService: AuthService) { }

  private getAuthHeaders(): HttpHeaders {

    const token = this.authService.getToken();

    return new HttpHeaders().set('Authorization', `Bearer ${token}`);

  }

  getAllEvents(): Observable<Event[]> {

    return this.http.get<Event[]>(`${this.baseUrl}/all`, { headers: this.getAuthHeaders() }); //[cite: 10]

  }

  getEventById(id: number): Observable<Event> {

    return this.http.get<Event>(`${this.baseUrl}/${id}`, { headers: this.getAuthHeaders() }); //[cite: 11]

  }

  getEventsByCategory(category: string): Observable<Event> { // Note: Backend returns single Event [cite: 12]

    return this.http.get<Event>(`${this.baseUrl}/category/${category}`, { headers: this.getAuthHeaders() });

  }

  createEvent(event: Event): Observable<Event> {

    return this.http.post<Event>(`${this.baseUrl}/save`, event, { headers: this.getAuthHeaders() });// [cite: 13]

  }

  updateEvent(id: number, event: Event): Observable<Event> {

    return this.http.put<Event>(`${this.baseUrl}/update/${id}`, event, { headers: this.getAuthHeaders() }); //[cite: 14]

  }

  deleteEvent(id: number): Observable<void> {

    return this.http.delete<void>(`${this.baseUrl}/${id}`, { headers: this.getAuthHeaders() }); //[cite: 15]

  }

}

// src/app/services/feedback.service.ts

import { Injectable } from '@angular/core';

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Feedback } from '../models/feedback.model';

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

@Injectable({

  providedIn: 'root'

})

export class FeedbackService {

  private baseUrl = 'http://localhost:2061/feedback'; // Base URL for Feedback Controller [cite: 17]

  constructor(private http: HttpClient, private authService: AuthService) { }

  private getAuthHeaders(): HttpHeaders {

    const token = this.authService.getToken();

    return new HttpHeaders().set('Authorization', `Bearer ${token}`);

  }

  submitFeedback(feedback: Feedback): Observable<Feedback> {

    return this.http.post<Feedback>(`${this.baseUrl}/submit`, feedback, { headers: this.getAuthHeaders() }); //[cite: 18]

  }

  getFeedbackByEventId(eventId: number): Observable<Feedback[]> {

    return this.http.get<Feedback[]>(`${this.baseUrl}/event/${eventId}`, { headers: this.getAuthHeaders() });// [cite: 19]

  }

  getAverageRatingByEventId(eventId: number): Observable<number> {

    return this.http.get<number>(`${this.baseUrl}/event/${eventId}/average-rating`, { headers: this.getAuthHeaders() }); //[cite: 20]

  }

  getFeedbackById(id: number): Observable<Feedback> {

    return this.http.get<Feedback>(`${this.baseUrl}/${id}`, { headers: this.getAuthHeaders() }); //[cite: 21]

  }

}

// src/app/services/notification.service.ts

import { Injectable } from '@angular/core';

import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';

import { Observable } from 'rxjs';

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

@Injectable({

  providedIn: 'root'

})

export class NotificationService {

  private baseUrl = 'http://localhost:2061/notifications'; // Base URL for Notification Controller [cite: 23]

  constructor(private http: HttpClient, private authService: AuthService) { }

  private getAuthHeaders(): HttpHeaders {

    const token = this.authService.getToken();

    return new HttpHeaders().set('Authorization', `Bearer ${token}`);

  }

  sendDefaultNotification(userId: number, eventId: number): Observable<string> {

    let params = new HttpParams()

      .set('userId', userId.toString())

      .set('eventId', eventId.toString());

    return this.http.post(`${this.baseUrl}/send`, null, { params, headers: this.getAuthHeaders(), responseType: 'text' }); //[cite: 27]

  }

  sendCustomNotification(userId: number, subject: string, message: string): Observable<string> {

    let params = new HttpParams()

      .set('userId', userId.toString())

      .set('subject', subject)

      .set('message', message);

    return this.http.post(`${this.baseUrl}/send-custom`, null, { params, headers: this.getAuthHeaders(), responseType: 'text' }); //[cite: 30]

  }

}

// src/app/services/ticket.service.ts

import { Injectable } from '@angular/core';

import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Ticket } from '../models/ticket.model';

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

@Injectable({

  providedIn: 'root'

})

export class TicketService {

  private baseUrl = 'http://localhost:2061/tickets'; // Base URL for Ticket Controller [cite: 32]

  constructor(private http: HttpClient, private authService: AuthService) { }

  private getAuthHeaders(): HttpHeaders {

    const token = this.authService.getToken();

    return new HttpHeaders().set('Authorization', `Bearer ${token}`);

  }

  bookTicket(eventId: number, userId: number): Observable<Ticket> {

    let params = new HttpParams()

      .set('eventId', eventId.toString())

      .set('userId', userId.toString());

    return this.http.post<Ticket>(`${this.baseUrl}/book`, null, { params, headers: this.getAuthHeaders() });// [cite: 35]

  }

  getTicketsByUserId(userId: number): Observable<Ticket[]> {

    return this.http.get<Ticket[]>(`${this.baseUrl}/user/${userId}`, { headers: this.getAuthHeaders() });// [cite: 36]

  }

  getTicketsByEventId(eventId: number): Observable<Ticket[]> {

    return this.http.get<Ticket[]>(`${this.baseUrl}/event/${eventId}`, { headers: this.getAuthHeaders() }); //[cite: 37]

  }

  cancelTicket(ticketId: number): Observable<Ticket> {

    return this.http.put<Ticket>(`${this.baseUrl}/cancel/${ticketId}`, null, { headers: this.getAuthHeaders() }); //[cite: 38]

  }

}

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

    <div class="container-fluid">

      <a class="navbar-brand" routerLink="/">Eventify</a>

      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

        <span class="navbar-toggler-icon"></span>

      </button>

      <div class="collapse navbar-collapse" id="navbarNav">

        <ul class="navbar-nav me-auto mb-2 mb-lg-0">

          <ng-container \*ngIf="isLoggedIn$ | async">

            <li class="nav-item">

              <a class="nav-link" routerLink="/events" routerLinkActive="active">Events</a>

            </li>

            <li class="nav-item">

              <a class="nav-link" routerLink="/my-tickets" routerLinkActive="active">My Tickets</a>

            </li>

            <ng-container \*ngIf="(userRole$ | async) === 'ADMIN' || (userRole$ | async) === 'ORGANIZER'">

              <li class="nav-item">

                <a class="nav-link" routerLink="/events/new" routerLinkActive="active">Add Event</a>

              </li>

              <li class="nav-item">

                <a class="nav-link" routerLink="/feedback-list" routerLinkActive="active">Feedback</a>

              </li>

            </ng-container>

          </ng-container>

        </ul>

        <ul class="navbar-nav">

          <ng-container \*ngIf="!(isLoggedIn$ | async)">

            <li class="nav-item">

              <a class="nav-link" routerLink="/login" routerLinkActive="active">Login</a>

            </li>

            <li class="nav-item">

              <a class="nav-link" routerLink="/register" routerLinkActive="active">Register</a>

            </li>

          </ng-container>

          <ng-container \*ngIf="isLoggedIn$ | async">

            <li class="nav-item">

              <span class="nav-link text-white">Welcome, {{ (userRole$ | async) }}!</span>

            </li>

            <li class="nav-item">

              <button class="btn btn-outline-light" (click)="onLogout()">Logout</button>

            </li>

          </ng-container>

        </ul>

      </div>

    </div>

  </nav>

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

import { AuthService } from '../../services/auth.service';

import { Router } from '@angular/router';

import { Observable } from 'rxjs';

@Component({

  selector: 'app-navbar',

  standalone: false,

  templateUrl: './navbar.component.html',

  styleUrl: './navbar.component.css'

})

export class NavbarComponent implements OnInit {

  isLoggedIn$: Observable<boolean>;

  userRole$: Observable<string | null>;

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

    this.isLoggedIn$ = this.authService.isLoggedIn();

    this.userRole$ = this.authService.getUserRole();

  }

  ngOnInit(): void {

  }

  onLogout(): void {

    this.authService.logout();

  }

}

<div class="container mt-4">

    <h2>My Booked Tickets</h2>

    <div \*ngIf="errorMessage" class="alert alert-danger" role="alert">

      {{ errorMessage }}

    </div>

    <div \*ngIf="!userId && !errorMessage" class="alert alert-info" role="alert">

      Please log in to view your booked tickets.

    </div>

    <div \*ngIf="userId && myTickets.length === 0 && !errorMessage" class="alert alert-info" role="alert">

      You have not booked any tickets yet.

    </div>

    <div \*ngIf="myTickets.length > 0">

      <table class="table table-striped table-bordered mt-3">

        <thead>

          <tr>

            <th>Ticket ID</th>

            <th>Event Name</th>

            <th>Booking Date</th>

            <th>Status</th>

            <th>Actions</th>

          </tr>

        </thead>

        <tbody>

          <tr \*ngFor="let ticket of myTickets">

            <td>{{ ticket.id }}</td>

            <td>{{ getEventName(ticket.eventId) }}</td>

            <td>{{ ticket.bookingDate | date:'mediumDate' }}</td>

            <td>

              <span class="badge" [class.bg-success]="ticket.status === 'Confirmed'" [class.bg-danger]="ticket.status === 'Cancelled'">

                {{ ticket.status }}

              </span>

            </td>

            <td>

              <button class="btn btn-warning btn-sm" (click)="onCancelTicket(ticket.id)" [disabled]="ticket.status === 'Cancelled'">Cancel Ticket</button>

            </td>

          </tr>

        </tbody>

      </table>

    </div>

  </div>

// src/app/tickets/my-tickets/my-tickets.component.ts

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

import { TicketService } from '../../services/ticket.service';

import { Ticket } from '../../models/ticket.model';

import { AuthService } from '../../services/auth.service';

import { EventService } from '../../services/event.service';

import { Event } from '../../models/event.model';

import { combineLatest, switchMap, map } from 'rxjs';

@Component({

  selector: 'app-my-tickets',

  templateUrl: './my-tickets.component.html',

  standalone:false,

  styleUrls: ['./my-tickets.component.css']

})

export class MyTicketsComponent implements OnInit {

  myTickets: Ticket[] = [];

  userId: number | null = null;

  errorMessage: string = '';

  eventsMap: { [key: number]: Event } = {}; // To store event details for display

  constructor(

    private ticketService: TicketService,

    private authService: AuthService,

    private eventService: EventService

  ) { }

  ngOnInit(): void {

    this.authService.getCurrentUserId().pipe(

      switchMap(id => {

        this.userId = id;

        if (id) {

          return this.ticketService.getTicketsByUserId(id); //[cite: 36]

        } else {

          this.errorMessage = 'You must be logged in to view your tickets.';

          return []; // Return empty array if not logged in

        }

      }),

      switchMap(tickets => {

        this.myTickets = tickets;

        const eventIds = new Set(tickets.map(ticket => ticket.eventId));

        const eventObservables = Array.from(eventIds).map(eventId => this.eventService.getEventById(eventId)); //[cite: 11]

        return combineLatest(eventObservables).pipe(

          map(events => {

            events.forEach(event => {

              if (event.id) {

                this.eventsMap[event.id] = event;

              }

            });

            return tickets;

          })

        );

      })

    ).subscribe({

      next: () => {}, // Data already set in switchMap

      error: (err) => {

        console.error('Error fetching tickets or events:', err);

        this.errorMessage = 'Failed to load your tickets. ' + (err.error || 'Please try again.');

      }

    });

  }

  getEventName(eventId: number): string {

    return this.eventsMap[eventId]?.name || 'Loading...';

  }

  onCancelTicket(ticketId: number | undefined): void {

    if (ticketId && confirm('Are you sure you want to cancel this ticket?')) {

      this.ticketService.cancelTicket(ticketId).subscribe({   //[cite: 38]

        next: (updatedTicket) => {

          console.log('Ticket cancelled:', updatedTicket);

          alert(`Ticket ${ticketId} has been ${updatedTicket.status}.`);

          // Update the status locally or reload tickets

          const index = this.myTickets.findIndex(t => t.id === ticketId);

          if (index !== -1) {

            this.myTickets[index].status = updatedTicket.status;

          }

        },

        error: (err) => {

          console.error('Error cancelling ticket:', err);

          this.errorMessage = 'Failed to cancel ticket: ' + (err.error || 'Please try again.');

        }

      });

    }

  }

}

// src/app/interceptors/auth.interceptor.ts

import { Injectable } from '@angular/core';

import {

  HttpRequest,

  HttpHandler,

  HttpEvent,

  HttpInterceptor

} from '@angular/common/http';

import { Observable } from 'rxjs';

// import { AuthService } from '../services/auth.service';

import { AuthService } from './services/auth.service';

@Injectable()

export class AuthInterceptor implements HttpInterceptor {

  constructor(private authService: AuthService) {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {

    const token = this.authService.getToken();

    if (token) {

      const cloned = request.clone({

        headers: request.headers.set('Authorization', `Bearer ${token}`)

      });

      return next.handle(cloned);

    } else {

      return next.handle(request);

    }

  }

}

// import { NgModule } from '@angular/core';

// import { BrowserModule, provideClientHydration, withEventReplay } from '@angular/platform-browser';

// import { AppRoutingModule } from './app-routing.module';

// import { AppComponent } from './app.component';

// import { LoginComponent } from './auth/login/login.component';

// import { RegisterComponent } from './auth/register/register.component';

// import { EventListComponent } from './events/event-list/event-list.component';

// import { EventDetailsComponent } from './events/event-details/event-details.component';

// import { EventFormComponent } from './events/event-form/event-form.component';

// import { FeedbackFormComponent } from './feedback/feedback-form/feedback-form.component';

// import { FeedbackListComponent } from './feedback/feedback-list/feedback-list.component';

// import { TicketBookingComponent } from './tickets/ticket-booking/ticket-booking.component';

// import { MyTicketsComponent } from './tickets/my-tickets/my-tickets.component';

// import { NavbarComponent } from './shared/navbar/navbar.component';

// import { HomeComponent } from './home/home/home.component';

// @NgModule({

//   declarations: [

//     AppComponent,

//     LoginComponent,

//     RegisterComponent,

//     EventListComponent,

//     EventDetailsComponent,

//     EventFormComponent,

//     FeedbackFormComponent,

//     FeedbackListComponent,

//     TicketBookingComponent,

//     MyTicketsComponent,

//     NavbarComponent,

//     HomeComponent

//   ],

//   imports: [

//     BrowserModule,

//     AppRoutingModule

//   ],

//   providers: [

//     provideClientHydration(withEventReplay())

//   ],

//   bootstrap: [AppComponent]

// })

// export class AppModule { }

// src/app/app.module.ts

// src/app/app.module.ts

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { FormsModule } from '@angular/forms';

import { HTTP\_INTERCEPTORS, HttpClientModule } from '@angular/common/http';

import { CommonModule } from '@angular/common'; // <--- Import CommonModule here

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { LoginComponent } from './auth/login/login.component';

import { RegisterComponent } from './auth/register/register.component';

import { EventListComponent } from './events/event-list/event-list.component';

import { EventDetailsComponent } from './events/event-details/event-details.component';

import { EventFormComponent } from './events/event-form/event-form.component';

import { FeedbackListComponent } from './feedback/feedback-list/feedback-list.component';

import { MyTicketsComponent } from './tickets/my-tickets/my-tickets.component';

import { TicketBookingComponent } from './tickets/ticket-booking/ticket-booking.component';

import { NavbarComponent } from './shared/navbar/navbar.component';

import { HomeComponent } from './home/home/home.component';

import { AuthInterceptor } from './auth.interceptor';

// Optional: If you decided to use AuthInterceptor

// import { HTTP\_INTERCEPTORS } from '@angular/common/http';

// import { AuthInterceptor } from './interceptors/auth.interceptor';

@NgModule({

  declarations: [

    AppComponent,

    LoginComponent,

    RegisterComponent,

    EventListComponent,

    EventDetailsComponent,

    EventFormComponent,

    FeedbackListComponent,

    MyTicketsComponent,

    TicketBookingComponent,

    NavbarComponent,

    HomeComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule,

    HttpClientModule,

    CommonModule // <--- Add CommonModule to the imports array

  ],

  providers: [

    // Optional: If you decided to use AuthInterceptor

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: AuthInterceptor,

      multi: true

    }

  ],

  bootstrap: [AppComponent]

})

export class AppModule { }

s