```
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
└── ...
```

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

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  standalone:false,
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  user = { name: '', password: '' };
  errorMessage = '';

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

  onLogin(): void {
```

```
  this.errorMessage = '';
  this.authService.login(this.user).subscribe({
   next: (token) => {
    if (typeof window !== 'undefined') {
     localStorage.setItem('jwtToken', token);
    }
    this.router.navigate(['/home']);

   },
   error: () => {
    this.errorMessage = 'Login failed. Please check your username and password.';
   }
  });
 }
}
<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 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>

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

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  standalone:false,
  styleUrls: ['./register.component.css']
})
export class RegisterComponent {
  user = { name: '', email: '', password: '', contactNumber: '', role: 'USER' };
  confirmPassword = '';
  errorMessage = '';
  successMessage = '';
  isLoading = false;

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

  onRegister(): void {
    this.errorMessage = '';
    this.successMessage = '';
    if (this.user.password !== this.confirmPassword) {
      this.errorMessage = 'Passwords do not match.';
      return;
    }
```

```
    this.isLoading = true;
    this.authService.register(this.user).subscribe({
      next: () => {
        this.successMessage = 'Registration successful!';
        this.isLoading = false;
        setTimeout(() => this.router.navigate(['/login']), 2000);
      },
      error: () => {
        this.errorMessage = 'Registration failed. Please try again.';
        this.isLoading = false;
      }
    });
  }
}

<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';
import { TicketBookingRequest, Ticket } from '../../models/ticket.model'; // Ensure Ticket is imported

@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'));
  console.log('ngOnInit: Initial eventId from URL:', this.eventId); // Debugging line
  if (this.eventId) {
   this.loadEventDetails();
   this.loadFeedback();

   // Subscribe to get userId and log it
   this.authService.getCurrentUserId().subscribe(id => {
    this.userId = id;
    console.log('ngOnInit: userId from AuthService:', this.userId); // Debugging line
   });

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

   // Set userId for feedback if logged in (already handled above, but keeping for
completeness if separate logic needed)
   this.authService.getCurrentUserId().subscribe(id => {
    if (id) {
     this.newFeedback.userId = id;
    }
   });
```

```
  } else {
    console.error('ngOnInit: Event ID was not found in the URL. Check route configuration.'); // Debugging line
    this.router.navigate(['/events']); // Redirect if event ID is missing
  }
 }

 loadEventDetails(): void {
  if (this.eventId) {
   this.eventService.getEventById(this.eventId).subscribe({
    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({
    next: (data) => {
     this.feedbackList = data;
    },
```

```
    error: (err) => {
      console.error('Error fetching feedback:', err);
    }
   });
   this.feedbackService.getAverageRatingByEventId(this.eventId).subscribe({
    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({
```

```
    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 {
  console.log('onBookTicket called.'); // Debugging line
  console.log('Current eventId:', this.eventId); // Debugging line
  console.log('Current userId:', this.userId);   // Debugging line

  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

  const bookingRequest: TicketBookingRequest = {
   eventId: this.eventId,
   userId: this.userId,
   numberOfTickets: 1 // Assuming 1 ticket is booked by default from this page
  };

  this.ticketService.bookTickets(bookingRequest).subscribe({
   next: (ticket: Ticket) => { // Type 'ticket' as Ticket as returned by service
    this.bookingMessage = `Ticket booked successfully! Ticket ID: ${ticket.id}. Status: ${ticket.status}`;
    this.loadEventDetails(); // Refresh ticket count
   },
   error: (err: { error: any; }) => {
    console.error('Error booking ticket:', err);
    this.bookingMessage = `Failed to book ticket: ${err.error?.message || 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="event-form-container">
```

```
 <h2>{{ isEditMode ? 'Edit Event' : 'Add New Event' }}</h2> <div *ngIf="successMessage"
class="alert alert-success" role="alert">

  {{ successMessage }}

 </div>


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

  {{ errorMessage }}

 </div>


 <form [formGroup]="eventForm" (ngSubmit)="onSubmit()">

  <div class="mb-3">

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

   <input type="text" id="name" formControlName="name" class="form-control"

       [class.is-invalid]="eventForm.get('name')?.invalid &&
eventForm.get('name')?.touched">

   <div *ngIf="eventForm.get('name')?.invalid && eventForm.get('name')?.touched"
class="invalid-feedback">

    Event Name is required.

   </div>

  </div>


  <div class="mb-3">

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

   <input type="text" id="category" formControlName="category" class="form-control"

       [class.is-invalid]="eventForm.get('category')?.invalid &&
eventForm.get('category')?.touched">

   <div *ngIf="eventForm.get('category')?.invalid &&
eventForm.get('category')?.touched" class="invalid-feedback">

    Category is required.
```

```
    </div>
  </div>

  <div class="mb-3">
    <label for="location" class="form-label">Location:</label>
    <input type="text" id="location" formControlName="location" class="form-control"
        [class.is-invalid]="eventForm.get('location')?.invalid &&
eventForm.get('location')?.touched">
    <div *ngIf="eventForm.get('location')?.invalid &&
eventForm.get('location')?.touched" class="invalid-feedback">
      Location is required.
    </div>
  </div>

  <div class="mb-3">
    <label for="date" class="form-label">Date and Time:</label>
    <input type="datetime-local" id="date" formControlName="date" class="form-
control"
        [class.is-invalid]="eventForm.get('date')?.invalid &&
eventForm.get('date')?.touched">
    <div *ngIf="eventForm.get('date')?.invalid && eventForm.get('date')?.touched"
class="invalid-feedback">
      Date and Time is required.
    </div>
  </div>

  <div class="mb-3">
    <label for="organizerId" class="form-label">Organizer ID:</label>
    <input type="number" id="organizerId" formControlName="organizerId"
class="form-control"
```

```
        [class.is-invalid]="eventForm.get('organizerId')?.invalid &&
eventForm.get('organizerId')?.touched">
    <div *ngIf="eventForm.get('organizerId')?.invalid &&
eventForm.get('organizerId')?.touched" class="invalid-feedback">
      Organizer ID is required and must be a positive number.
    </div>
  </div>

  <div class="mb-3">
    <label for="ticketCount" class="form-label">Ticket Count:</label>
    <input type="number" id="ticketCount" formControlName="ticketCount"
class="form-control"
        [class.is-invalid]="eventForm.get('ticketCount')?.invalid &&
eventForm.get('ticketCount')?.touched">
    <div *ngIf="eventForm.get('ticketCount')?.invalid &&
eventForm.get('ticketCount')?.touched" class="invalid-feedback">
      Ticket Count is required and must be a positive number.
    </div>
  </div>

  <button type="submit" class="btn btn-success" [disabled]="isSubmitting ||
eventForm.invalid">
    {{ isSubmitting ? (isEditMode ? 'Updating...' : 'Adding Event...') : (isEditMode ? 'Update
Event' : 'Add Event') }}
  </button>
  <button type="button" class="btn btn-secondary ms-2"
(click)="eventForm.reset()">Clear Form</button>
  <button type="button" class="btn btn-info ms-2" routerLink="/events">Back to
Events</button>
 </form>
</div>
```

```
// src/app/events/event-form/event-form.component.ts
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { EventService } from '../../services/event.service'; // Adjust path
import { Router, ActivatedRoute } from '@angular/router'; // Import ActivatedRoute
import { HttpErrorResponse } from '@angular/common/http';
import { Event } from '../../models/event.model'; // Import Event model

@Component({
  selector: 'app-event-form',
  templateUrl: './event-form.component.html',
  standalone:false,
  styleUrls: ['./event-form.component.css']
})
export class EventFormComponent implements OnInit {
  eventForm: FormGroup;
  isSubmitting = false;
  successMessage: string = '';
  errorMessage: string = '';
  isEditMode = false; // New flag to track mode
  eventId: number | null = null; // Stores event ID if in edit mode

  constructor(
    private fb: FormBuilder,
    private eventService: EventService,
    private router: Router,
    private route: ActivatedRoute // Inject ActivatedRoute
```

```
) {
  // Initialize the form with FormBuilder
  this.eventForm = this.fb.group({
    id: [null], // Include id field for edit mode, but it won't be displayed
    name: ['', Validators.required],
    category: ['', Validators.required],
    location: ['', Validators.required],
    date: ['', Validators.required],
    organizerId: [null, [Validators.required, Validators.min(1)]],
    ticketCount: [null, [Validators.required, Validators.min(1)]]
  });
}

ngOnInit(): void {
  // Check if we are in edit mode by looking for an 'id' in the route
  this.route.paramMap.subscribe(params => {
    const idParam = params.get('id');
    if (idParam) {
      this.isEditMode = true;
      this.eventId = +idParam; // Convert string to number
      this.loadEventForEditing(this.eventId);
    }
  });
}

loadEventForEditing(id: number): void {
  this.eventService.getEventById(id).subscribe({
    next: (event: Event) => {
```

```
      // Populate the form with event data
      // For date, convert to 'YYYY-MM-DDTHH:mm' format required by datetime-local input
      const formattedDate = event.date ? new Date(event.date).toISOString().slice(0, 16) : '';
      this.eventForm.patchValue({
        id: event.id,
        name: event.name,
        category: event.category,
        location: event.location,
        date: formattedDate,
        organizerId: event.organizerId,
        ticketCount: event.ticketCount
      });
      console.log('Event loaded for editing:', event);
    },
    error: (error: HttpErrorResponse) => {
      console.error('Error loading event for editing:', error);
      this.errorMessage = 'Failed to load event details. Please try again.';
      // Optional: Redirect if event not found or unauthorized
      // this.router.navigate(['/events']);
    }
  });
 }

 onSubmit(): void {
  this.isSubmitting = true;
  this.successMessage = '';
  this.errorMessage = '';
```

```
if (this.eventForm.valid) {
  const eventToSave: Event = this.eventForm.value;

  if (this.isEditMode && this.eventId) {
    // Update existing event
    this.eventService.updateEvent(this.eventId, eventToSave).subscribe({
      next: (response) => {
        this.successMessage = 'Event updated successfully!';
        this.isSubmitting = false;
        console.log('Event updated:', response);
        // Optional: Navigate back to event list after a short delay
        setTimeout(() => {
          this.router.navigate(['/events']);
        }, 2000);
      },
      error: (error: HttpErrorResponse) => {
        this.isSubmitting = false;
        console.error('Error updating event:', error);
        this.handleError(error, 'update');
      }
    });
  } else {
    // Create new event
    this.eventService.createEvent(eventToSave).subscribe({
      next: (response) => {
        this.successMessage = 'Event added successfully!';
        this.isSubmitting = false;
```

```
          this.eventForm.reset(); // Clear the form for new entry
          console.log('Event added:', response);
          // Optional: Navigate back to event list after a short delay
          setTimeout(() => {
            this.router.navigate(['/events']);
          }, 2000);
        },
        error: (error: HttpErrorResponse) => {
          this.isSubmitting = false;
          console.error('Error adding event:', error);
          this.handleError(error, 'add');
        }
      });
    }
  } else {
    this.errorMessage = 'Please fill in all required fields correctly.';
    this.isSubmitting = false;
    this.eventForm.markAllAsTouched(); // Mark all fields as touched to display validation errors
  }
 }


 private handleError(error: HttpErrorResponse, operation: 'add' | 'update'): void {
  if (error.status === 400) {
    this.errorMessage = `Bad Request: ${error.error.message || 'Please check your input.'}`;
  } else if (error.status === 401 || error.status === 403) {
    this.errorMessage = `You are not authorized to ${operation} events. Please ensure you are logged in as an Admin.`;
```

```
  } else if (error.status === 404 && operation === 'update') {
    this.errorMessage = 'Event not found for updating.';
  } else {
    this.errorMessage = `An unexpected error occurred: ${error.message || 'Unknown error'}`;
  }
 }
}
<div class="event-list-container">
 <h2>Available Events</h2> <div class="mb-3">
  <button *ngIf="userRole === 'ADMIN'" routerLink="/events/new" class="btn btn-primary">
    <i class="fas fa-plus"></i> Add New Event
  </button>
 </div>

 <div *ngIf="errorMessage" class="alert alert-danger" role="alert">
  {{ errorMessage }}
 </div>

 <div *ngIf="events.length === 0 && !errorMessage" class="no-events-message">
  <p>No events available at the moment. Please check back later!</p>
 </div>

 <div *ngIf="events.length > 0" class="row">
  <div class="col-md-4 mb-4" *ngFor="let event of events">
   <div class="card event-card">
    <div class="card-body">
     <h5 class="card-title">{{ event.name }}</h5>
```

```
      <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 | date:'mediumDate' }}</p>
      <p class="card-text">
       <strong>Tickets Available:</strong>
       <span [ngClass]="{
        'text-danger': event.ticketCount !== undefined && event.ticketCount <= 10,
        'text-warning': event.ticketCount !== undefined && event.ticketCount > 10 && event.ticketCount <= 50,
        'text-success': event.ticketCount !== undefined && event.ticketCount > 50
       }">
        {{ event.ticketCount !== undefined ? event.ticketCount : 'N/A' }}
       </span>
      </p>

      <div class="mt-2 d-flex justify-content-between align-items-center">
       <ng-container >
        <button
         *ngIf="event.ticketCount !== undefined && event.ticketCount > 0"
         [routerLink]="['/tickets', event.id]"
         class="btn btn-sm btn-success flex-grow-1 me-2"
        >
         Book Tickets
        </button>
        <span *ngIf="event.ticketCount !== undefined && event.ticketCount <= 0" class="text-danger flex-grow-1 me-2">Sold Out!</span>
       </ng-container>
```

```
        <div *ngIf="userRole === 'ADMIN'" class="d-flex">
          <button
            [routerLink]="['/events/edit', event.id]"
            class="btn btn-sm btn-warning me-2"
          >
            Edit
          </button>
          <button
            (click)="deleteEvent(event.id)"
            class="btn btn-sm btn-danger"
          >
            Delete
          </button>
        </div>
      </div>

    </div> </div> </div> </div> </div>
```

```
import { Component, OnInit } from '@angular/core';
import { EventService } from '../../services/event.service'; // Adjust path if necessary
import { Event } from '../../models/event.model'; // Adjust path if necessary
import { HttpErrorResponse } from '@angular/common/http';
import { AuthService } from '../../services/auth.service'; // Adjust path if necessary

@Component({
  selector: 'app-event-list',
  templateUrl: './event-list.component.html',
```

```
  standalone:false,
  styleUrls: ['./event-list.component.css']
})
export class EventListComponent implements OnInit {
  events: Event[] = [];
  errorMessage: string = '';
  userRole: string | null = null;

  isAdminUser: boolean = false; // Flag to control admin specific UI elements

  constructor(
    private eventService: EventService,
    private authService: AuthService // Inject AuthService
  ) { }

  ngOnInit(): void {
    this.loadEvents();
    // Check if the logged-in user is an admin
    // this.isAdminUser = this.authService.isAdmin();
    this.authService.getUserRole().subscribe(role => this.userRole = role);
    // Optional: Subscribe to currentUser changes if roles can dynamically change without full page reload
    // this.authService.currentUser.subscribe(() => {
    //   this.isAdminUser = this.authService.isAdmin();
    // });
  }

  loadEvents(): void {
```

```
  this.eventService.getAllEvents().subscribe({
    next: (data: Event[]) => {
      this.events = data;
      this.errorMessage = ''; // Clear any previous error messages
      console.log('Events loaded successfully:', this.events); // For debugging
    },
    error: (error: HttpErrorResponse) => {
      console.error('Error fetching events:', error);
      if (error.status === 401 || error.status === 403) {
        this.errorMessage = 'You are not authorized to view events. Please log in.';
      } else if (error.status === 404) {
        this.errorMessage = 'The event list endpoint was not found.';
      } else if (error.error instanceof ErrorEvent) {
        this.errorMessage = `An error occurred: ${error.error.message}`;
      } else {
        this.errorMessage = `Error fetching events: ${error.status} - ${error.message ||
'Unknown error'}`;
      }
    }
  });
 }
 deleteEvent(id?: number): void {
  if (id === undefined) {
    console.error('Cannot delete event: ID is undefined.');
    this.errorMessage = 'Error: Event ID is missing for deletion.';
    return;
  }
```

```
  if (confirm('Are you sure you want to delete this event?')) {
    this.eventService.deleteEvent(id).subscribe({
      next: () => {
        console.log(` Event with ID ${id} deleted successfully.`);
        this.errorMessage = ''; // Clear any previous error
        // Remove the deleted event from the local array
        this.events = this.events.filter(event => event.id !== id);
      },
      error: (error: HttpErrorResponse) => {
        console.error(` Error deleting event with ID ${id}:` , error);
        if (error.status === 401 || error.status === 403) {
          this.errorMessage = 'You are not authorized to delete events. Please log in as an
Admin.';
        } else if (error.status === 404) {
          this.errorMessage = 'Event not found.';
        } else {
          this.errorMessage = ` Error deleting event: ${error.message || 'Unknown error'}` ;
        }
      }
    });
  }
 }
}


<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]
 }
}

import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';

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

 canActivate(): boolean {
  const token = typeof window !== 'undefined' ? localStorage.getItem('jwtToken') : null;
  if (!token) {
   this.router.navigate(['/login']);
```

```
    return false;
   }
   return true;
  }
}

import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, Router } from '@angular/router';
// import { AuthService } from './auth.service';
import { AuthService } from '../services/auth.service';
import { Observable, of } from 'rxjs';
import { map } from 'rxjs/operators';

@Injectable({ providedIn: 'root' })
export class RoleGuard implements CanActivate {

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

 canActivate(route: ActivatedRouteSnapshot): Observable<boolean> {
  const expectedRole = route.data['role'];

  return this.authService.getUserRole().pipe(
   map(role => {
    if (role === expectedRole || role === 'ADMIN') {
     return true;
    } else {
     this.router.navigate(['/unauthorized']);
     return false;
```

```
      }
    })
  );
 }
}
<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 {
    // This method is part of the OnInit interface.
    // It is called once after the component has been initialized.
    // You can add any initialization logic here if needed.
    // For now, an empty body is sufficient to resolve the "Method not implemented" error.

    // Example of potential logic if you needed to do something after roles are loaded:
    // this.userRole$.subscribe(role => {
    //   console.log('HomeComponent user role initialized:', role);
    // });
  }
}
```

```
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;
 }
// src/app/models/ticket.model.ts
export interface TicketBookingRequest {
 eventId: number;
 userId: number; // Assuming you get the user ID from your auth service/token
 numberOfTickets: number;
}

// Optional: If you want a response model for booked tickets
export interface Ticket {
 id?: number;
 eventId: number;
 userId: number;
 quantity: number;
 bookingDate?: string; // e.g., 'YYYY-MM-DDTHH:mm:ss'
 status?: 'CONFIRMED' | 'CANCELLED';
}

// 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;
}



  // import { Injectable, Inject, PLATFORM_ID } from '@angular/core';
// import { HttpClient } from '@angular/common/http';
// import { Observable, of } from 'rxjs';
// import { isPlatformBrowser } from '@angular/common';

// @Injectable({ providedIn: 'root' })
// export class AuthService {
```

```
//   hasRole: any;
//   currentUser: any;
//   // isAdmin(): boolean {
//   //   return this.hasRole('ADMIN')
//   // }
//   private apiUrl = 'http://localhost:2061/auth';

//   constructor(
//     private http: HttpClient,
//     @Inject(PLATFORM_ID) private platformId: Object
//   ) {}

//   login(credentials: any): Observable<string> {
//     return this.http.post(`${this.apiUrl}/login`, credentials, { responseType: 'text' });

//   }

//   register(user: any): Observable<string> {
//     return this.http.post(`${this.apiUrl}/register`, user, { responseType: 'text' });
//   }

//   getToken(): string | null {
//     if (isPlatformBrowser(this.platformId)) {
//       return localStorage.getItem('jwtToken');
//     }
//     return null;
//   }
```

```
//   isLoggedIn(): Observable<boolean> {
//     return of(!!this.getToken());
//   }

//   logout(): void {
//     if (isPlatformBrowser(this.platformId)) {
//       localStorage.removeItem('jwtToken');
//     }
//   }

//   getUserRole(): Observable<string | null> {
//     const token = this.getToken();
//     if (!token) return of(null);

//     const payload = JSON.parse(atob(token.split('.')[1]));
//     const roles = payload.roles;

//     // Check structure: role might be an array of authorities
//     if (Array.isArray(roles)) {
//       // If first item is object: { authority: "ROLE_USER" }
//       if (typeof roles[0] === 'object' && roles[0] !== null) {
//         return of(roles[0].authority || null);
//       } else {
//         return of(roles[0]);
//       }
//     }

//     return of(null);
```

```
//   }

//   getCurrentUserId(): Observable<number | null> {
//     const token = this.getToken();
//     if (!token) return of(null);
//     const payload = JSON.parse(atob(token.split('.')[1]));
//     const sub = payload.sub;
//     return of(isNaN(Number(sub)) ? null : Number(sub));
//   }
// }
//=========================================================================
========================




// src/app/services/auth.service.ts
import { Injectable, Inject, PLATFORM_ID } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, of } from 'rxjs';
import { isPlatformBrowser } from '@angular/common';

@Injectable({ providedIn: 'root' })
export class AuthService {
  // Removed unused properties 'hasRole' and 'currentUser'
  private apiUrl = 'http://localhost:2061/auth';

  constructor(
    private http: HttpClient,
```

```
  @Inject(PLATFORM_ID) private platformId: Object
) {}

login(credentials: any): Observable<string> {
  return this.http.post(`${this.apiUrl}/login`, credentials, { responseType: 'text' });
}

register(user: any): Observable<string> {
  return this.http.post(`${this.apiUrl}/register`, user, { responseType: 'text' });
}

getToken(): string | null {
  if (isPlatformBrowser(this.platformId)) {
    return localStorage.getItem('jwtToken');
  }
  return null;
}

isLoggedIn(): Observable<boolean> {
  return of(!!this.getToken());
}

logout(): void {
  if (isPlatformBrowser(this.platformId)) {
    localStorage.removeItem('jwtToken');
  }
}
```

```
 getUserRole(): Observable<string | null> {
  const token = this.getToken();
  if (!token) return of(null);

  try {
   const payload = JSON.parse(atob(token.split('.')[1]));
   const roles = payload.roles;

   if (Array.isArray(roles) && roles.length > 0) {
    if (typeof roles[0] === 'object' && roles[0] !== null && 'authority' in roles[0]) {
     return of(roles[0].authority);
    } else {
     return of(roles[0]);
    }
   }
  } catch (e) {
   console.error('Error decoding or parsing JWT token for role:', e);
  }
  return of(null);
 }

 getCurrentUserId(): Observable<number | null> {
  const token = this.getToken();
  if (!token) return of(null);
  try {
   const payload = JSON.parse(atob(token.split('.')[1]));
   const userIdFromToken = payload.sub || payload.id; // Use 'sub' or 'id' based on your
backend JWT
```

```
    return of(isNaN(Number(userIdFromToken)) ? null : Number(userIdFromToken));
   } catch (e) {
    console.error('Error decoding or parsing JWT token for userId:', e);
   }
   return of(null);
 }
}

// 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 } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Ticket, TicketBookingRequest } from '../models/ticket.model'; // Import Ticket and TicketBookingRequest
import { AuthService } from './auth.service';

@Injectable({
  providedIn: 'root'
})
export class TicketService {
  private baseUrl = 'http://localhost:2061/tickets';

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

  private getAuthHeaders(): HttpHeaders {
    const token = this.authService.getToken();
    // Ensure token is not null before setting header
    return new HttpHeaders().set('Authorization', `Bearer ${token || ''}`);
  }

  // Corrected signature: accepts ONLY the TicketBookingRequest object
  bookTickets(request: TicketBookingRequest): Observable<Ticket> {
    return this.http.post<Ticket>(`${this.baseUrl}/book`, request, { headers: this.getAuthHeaders() });
  }

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

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

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

 cancelTicket(ticketId: number): Observable<Ticket> {
  return this.http.put<Ticket>(`${this.baseUrl}/cancel/${ticketId}`, null, { headers: this.getAuthHeaders() });
 }
}
```

```
<!-- navbar.component.html --><nav class="navbar navbar-expand-lg navbar-light bg-light">
 <a class="navbar-brand" routerLink="/home">Events</a>
 <div class="collapse navbar-collapse">
  <ul class="navbar-nav me-auto">
   <li class="nav-item" *ngIf="userRole === 'USER'">
    <a class="nav-link" routerLink="/events">View Events</a>
   </li>
   <li class="nav-item" *ngIf="userRole === 'USER'">
    <a class="nav-link" routerLink="/tickets">Book Tickets</a>
   </li>
   <li class="nav-item" *ngIf="userRole === 'USER'">
    <a class="nav-link" routerLink="/feedback">Feedback</a>
```

```
    </li>
    <li class="nav-item" *ngIf="userRole === 'ADMIN'">
      <a class="nav-link" routerLink="/events">Manage Events</a>
    </li>

    <li class="nav-item" *ngIf="userRole === 'ADMIN'">
      <a class="nav-link" routerLink="/tickets">All Tickets</a>
    </li>
    <li class="nav-item" *ngIf="userRole === 'ADMIN'">
      <a class="nav-link" routerLink="/feedback">All Feedback</a>
    </li>
   </ul>
   <button class="btn btn-outline-danger" (click)="logout()">Logout</button>
  </div>
</nav>


import { Component, OnInit } from '@angular/core';
 import { Router } from '@angular/router';
 import { AuthService } from '../../services/auth.service';

@Component({
  selector: 'app-navbar',
   templateUrl: './navbar.component.html',
   standalone:false,
   styleUrls: ['./navbar.component.css'] })
    export class NavbarComponent implements OnInit {
     userRole: string | null = null;
```

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

ngOnInit(): void { this.authService.getUserRole().subscribe(role => this.userRole = role);
}

logout(): void { this.authService.logout();
 this.router.navigate(['/login']); } }
```

```
<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.');
      }
```

```
    });
  }
 }
}
```

```
<div class="ticket-booking-container">
 <h2 *ngIf="event">{{ event.name }} - Book Tickets</h2>
 <h2 *ngIf="!event">Book Tickets</h2>

 <div *ngIf="errorMessage" class="alert alert-danger" role="alert">
  {{ errorMessage }}
 </div>

 <div *ngIf="successMessage" class="alert alert-success" role="alert">
  {{ successMessage }}
 </div>

 <div *ngIf="event">
  <p><strong>Category:</strong> {{ event.category }}</p>
  <p><strong>Location:</strong> {{ event.location }}</p>
  <p><strong>Date:</strong> {{ event.date | date:'mediumDate' }}</p>
  <p><strong>Tickets Available:</strong> {{ event.ticketCount }}</p>

  <form [formGroup]="bookingForm" (ngSubmit)="onBookTickets()">
   <div class="mb-3">
    <label for="numberOfTickets" class="form-label">Number of Tickets:</label>
```

```
      <input type="number" id="numberOfTickets"
formControlName="numberOfTickets"
            class="form-control" min="1" [max]="maxTickets"
            [class.is-invalid]="bookingForm.get('numberOfTickets')?.invalid &&
bookingForm.get('numberOfTickets')?.touched">
      <div *ngIf="bookingForm.get('numberOfTickets')?.invalid &&
bookingForm.get('numberOfTickets')?.touched" class="invalid-feedback">
        <span *ngIf="bookingForm.get('numberOfTickets')?.errors?.['required']">Number
of tickets is required.</span>
        <span *ngIf="bookingForm.get('numberOfTickets')?.errors?.['min']">Must book at
least 1 ticket.</span>
        <span *ngIf="bookingForm.get('numberOfTickets')?.errors?.['max']">Cannot book
more than {{ maxTickets }} tickets.</span>
      </div>
    </div>

    <button type="submit" class="btn btn-success" [disabled]="isSubmitting ||
bookingForm.invalid || !event || maxTickets <= 0">
      {{ isSubmitting ? 'Booking...' : (maxTickets <= 0 ? 'Sold Out' : 'Confirm Booking') }}
    </button>
    <button type="button" class="btn btn-secondary ms-2"
routerLink="/events">Cancel</button>
   </form>
  </div>

  <div *ngIf="!event && !errorMessage">
   <p>Loading event details...</p>
  </div>
</div>
```

```
// src/app/tickets/ticket-booking/ticket-booking.component.ts
// (No changes needed in this file, as the call was already correct for the intended signature)

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { EventService } from '../../services/event.service';
import { TicketService } from '../../services/ticket.service';
import { AuthService } from '../../services/auth.service';
import { Event } from '../../models/event.model';
import { TicketBookingRequest } from '../../models/ticket.model';
import { HttpErrorResponse } from '@angular/common/http';

@Component({
  selector: 'app-ticket-booking',
  templateUrl: './ticket-booking.component.html',
  standalone:false,
  styleUrls: ['./ticket-booking.component.css']
})
export class TicketBookingComponent implements OnInit {
  eventId: number | null = null;
  event: Event | null = null;
  bookingForm: FormGroup;
  isSubmitting = false;
  successMessage: string = '';
  errorMessage: string = '';
  maxTickets: number = 0;
```

```
userId: number | null = null;
ticket: any;

constructor(
  private route: ActivatedRoute,
  private router: Router,
  private fb: FormBuilder,
  private eventService: EventService,
  private ticketService: TicketService,
  private authService: AuthService
) {
  this.bookingForm = this.fb.group({
    numberOfTickets: [1, [Validators.required, Validators.min(1)]]
  });
}

ngOnInit(): void {
  this.route.paramMap.subscribe(params => {
    const idParam = params.get('eventId');
    if (idParam) {
      this.eventId = +idParam;
      this.loadEventDetails(this.eventId);

      this.authService.getCurrentUserId().subscribe({
        next: (userId) => {
          if (userId !== null) {
            this.userId = userId;
            console.log('Logged-in userId from token:', this.userId);
```

```
        } else {
          this.errorMessage = 'User ID not found in token. Please log in.';
          console.error('User ID could not be retrieved from token. Is user logged in?');
        }
      },
      error: (err) => {
        console.error('Error getting current user ID:', err);
        this.errorMessage = 'Error retrieving user information. Please try logging in again.';
      }
    });

  } else {
    this.errorMessage = 'Event ID not provided. Cannot book tickets.';
    this.router.navigate(['/events']);
  }
 });
}

loadEventDetails(id: number): void {
 this.eventService.getEventById(id).subscribe({
  next: (event: Event) => {
    this.event = event;
    this.maxTickets = event.ticketCount !== undefined ? event.ticketCount : 0;
    this.bookingForm.get('numberOfTickets')?.setValidators([
      Validators.required,
      Validators.min(1),
      Validators.max(this.maxTickets)
    ]);
```

```
      this.bookingForm.get('numberOfTickets')?.updateValueAndValidity();
      console.log('Event details loaded:', this.event);
      console.log('Max tickets set to:', this.maxTickets);
    },
    error: (error: HttpErrorResponse) => {
      console.error('Error loading event details:', error);
      this.errorMessage = `Failed to load event details: ${error.message || 'Unknown error'}.`;
      this.router.navigate(['/events']);
    }
  });
 }

 onBookTickets(): void {
  this.isSubmitting = true;
  this.successMessage = '';
  this.errorMessage = '';

  console.log('Attempting to book tickets...');
  console.log('Form validity:', this.bookingForm.valid);
  console.log('Event ID:', this.eventId);
  console.log('User ID:', this.userId);
  console.log('Number of tickets requested:', this.bookingForm.get('numberOfTickets')?.value);
  console.log('Max tickets available:', this.maxTickets);
  console.log('Form errors:', this.bookingForm.errors);
  console.log('numberOfTickets control errors:', this.bookingForm.get('numberOfTickets')?.errors);
```

```
  if (this.bookingForm.valid && this.eventId && this.userId) {
    const numberOfTickets = this.bookingForm.get('numberOfTickets')?.value;

    if (numberOfTickets > this.maxTickets) {
      this.errorMessage = `You can only book up to ${this.maxTickets} tickets.`;
      this.isSubmitting = false;
      return;
    }

    const bookingRequest: TicketBookingRequest = {
      eventId: this.eventId,
      userId: this.userId,
      numberOfTickets: numberOfTickets
    };

    // The call was already correct, it just needed the TicketService signature to match
    this.ticketService.bookTickets(bookingRequest).subscribe({
      next: (response: any) => {
        this.successMessage = `Successfully booked ${numberOfTickets} ticket(s) for ${this.event?.name}!`;
        this.isSubmitting = false;
        if (this.event && this.event.ticketCount !== undefined) {
          this.event.ticketCount -= numberOfTickets;
          this.maxTickets = this.event.ticketCount;
        }
        this.bookingForm.reset({ numberOfTickets: 1 });
        console.log('Booking response:', response);
      },
```

```
      error: (error: HttpErrorResponse) => {
        this.isSubmitting = false;
        console.error('Error booking tickets:', error);
        if (error.status === 400) {
          this.errorMessage = `Booking failed: ${error.error.message || 'Not enough tickets available or invalid request.'}`;
        } else if (error.status === 401 || error.status === 403) {
          this.errorMessage = 'You are not authorized to book tickets. Please log in.';
        } else {
          this.errorMessage = `An unexpected error occurred: ${error.message || 'Unknown error'}`;
        }
      }
    });
  } else {
    if (!this.bookingForm.valid) {
      this.errorMessage = 'Please enter a valid number of tickets.';
    } else if (this.eventId === null || this.userId === null) {
      this.errorMessage = 'Cannot book tickets without event or user ID. Please ensure you are logged in and event details are loaded.';
    }
    this.isSubmitting = false;
    this.bookingForm.markAllAsTouched();
  }
 }
}


<div class="unauthorized-container">
  <h2>Access Denied!</h2>
```

```
    <p>You do not have the necessary permissions to view this page.</p>
    <p>Please log in with an authorized account or contact support.</p>
    <button routerLink="/login" class="btn btn-primary">Go to Login</button>
    <button routerLink="/" class="btn btn-secondary">Go to Home</button>
  </div>
import { Component } from '@angular/core';

@Component({
  selector: 'app-unauthorized',
  standalone: false,
  templateUrl: './unauthorized.component.html',
  styleUrl: './unauthorized.component.css'
})
export class UnauthorizedComponent {

}

import { Injectable } from '@angular/core';
import {
  HttpEvent,
  HttpHandler,
  HttpInterceptor,
  HttpRequest
} from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {
```

```
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const token = typeof window !== 'undefined' ? localStorage.getItem('jwtToken') : null;
    if (token) {
      req = req.clone({
        setHeaders: { Authorization: `Bearer ${token}` }
      });
    }
    return next.handle(req);
  }
}
// import { NgModule } from '@angular/core';
// import { RouterModule, Routes } from '@angular/router';

// const routes: Routes = [];

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

// src/app/app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
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 { HomeComponent } from './home/home/home.component';

import { AuthGuard } from './guards/auth.guard'; // Import your AuthGuard


// import { RoleGuard } from './services/role.guard';

import { RoleGuard } from './guards/role.gaurd';

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

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

import { UnauthorizedComponent } from './unauthorized/unauthorized.component';


// const routes: Routes = [

//   { path: '', component: HomeComponent },

//    { path: 'login', component: LoginComponent },

//    { path: 'register', component: RegisterComponent },


//   { path: 'home', component: HomeComponent, canActivate: [AuthGuard] },


//   { path: 'events', component: EventListComponent, canActivate: [AuthGuard] },

//   { path: 'events/:id', component: EventDetailsComponent, canActivate: [AuthGuard] },

//   { path: 'events/new', component: EventFormComponent, canActivate: [RoleGuard], data: { role: 'ROLE_ADMIN' } },

//   { path: 'events/edit/:id', component: EventFormComponent, canActivate: [RoleGuard], data: { role: 'ROLE_ADMIN' } },
```

```
//   { path: 'tickets', component: TicketBookingComponent, canActivate: [RoleGuard], data: { role: 'ROLE_USER' } },
//   { path: 'feedback', component: FeedbackFormComponent, canActivate: [RoleGuard], data: { role: 'ROLE_USER' } },
//   { path: 'unauthorized', component: UnauthorizedComponent }

// ];

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


const routes: Routes = [
 { path: '', component: HomeComponent },
 { path: 'login', component: LoginComponent },
 { path: 'register', component: RegisterComponent },

 // Protected routes
 { path: 'events', component: EventListComponent, canActivate: [AuthGuard] },
 { path: 'events/new', component: EventFormComponent, canActivate: [AuthGuard] },
 // { path: 'events/edit/:id', component: EventFormComponent, canActivate: [AuthGuard] },
 { path: 'tickets/:eventId', component: TicketBookingComponent, canActivate: [RoleGuard], data: { role: 'USER' } }, // <--- MODIFIED

 { path: 'events/:id', component: EventDetailsComponent, canActivate: [AuthGuard] },
```

```
  { path: 'feedback-list', component: FeedbackListComponent, canActivate: [AuthGuard] },
  { path: 'my-tickets', component: MyTicketsComponent, canActivate: [AuthGuard] },
  { path: 'event-details', component: EventDetailsComponent, canActivate: [AuthGuard] },
  { path: 'events/edit/:id', component: EventFormComponent, canActivate: [RoleGuard], data: { role: 'ADMIN' } }, // <--- ADD THIS LINE



  // Wildcard route for any other invalid path
  { path: '**', redirectTo: '' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
// 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, ReactiveFormsModule } 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 { RouterModule } from '@angular/router';

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';
import { UnauthorizedComponent } from './unauthorized/unauthorized.component';

// 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,
```

```
    UnauthorizedComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule,
    CommonModule,
    ReactiveFormsModule// <--- 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 { }
```