

Angular & .NET Cheatsheet (Code Only)

2025-12-02

Table of Contents

1. Frontend	1
1.1. Authentication	1
1.1.1. Interceptor	1
1.1.2. Auth Guard	1
1.2. Directives	2
1.2.1. Filter Directives	2
1.2.2. Async Directives	2
1.3. Routing	3
1.3.1. Routes	3
1.3.2. Navigation	3
1.3.3. Path Parameter	3
1.3.4. Query Parameter	4
1.4. Forms	4
1.4.1. Reactive Forms	4
1.4.2. Template Driven Forms	5
1.5. Signals	5
1.5.1. Models	5
1.5.2. Input	6
1.5.3. Output	7
1.6. HTTP Client	8
1.6.1. GET	8
1.6.2. POST	9
1.6.3. PUT	9
1.6.4. PATCH	9
1.6.5. DELETE	9
1.7. Template Syntax	10
1.7.1. @for	10
1.7.2. @if	10
1.8. Templates	10
1.8.1. Table	10
1.8.2. Input Form	11
1.8.3. Details View	16
2. Backend	22
2.1. CDI (Dependency Injection)	22
2.2. Swagger	22
2.2.1. Swagger Builder	22
2.2.2. Cors	23
2.3. Minimal API	23

2.3.1. GET	23
2.3.2. POST	24
2.3.3. PUT	24
2.3.4. PATCH	24
2.3.5. DELETE	25
2.4. Authentication	25
2.5. Services	25
2.5.1. Service Interface	26
2.5.2. Service Implementation	26
2.5.3. Service Registration	26
2.6. Unit Tests	27
2.6.1. XUnit	27
2.6.2. NUnit	28
2.6.3. FluentAssertions	28
2.6.4. Moq	29
2.6.5. Testen von Minimal APIs	31

1. Frontend

1.1. Authentication

1.1.1. Interceptor

```
// Automatically adds JWT token from sessionStorage to all HTTP requests
export const authInterceptor: HttpInterceptorFn = (request, next) => {
  const jwt = sessionStorage.getItem('jwt');
  const isLoggedIn = jwt;
  const isApiUrl = request.url.startsWith(environment.apiUrl);

  // Clone request and add Authorization header if logged in and API call
  if (isLoggedIn && isApiUrl) {
    request = request.clone({
      setHeaders: { Authorization: `Bearer ${jwt}` }
    });
  }

  return next(request);
}

// Register in app.config.ts: provideHttpClient(withInterceptors([authInterceptor]))
```

1.1.2. Auth Guard

```
// Protects routes by checking if JWT exists in sessionStorage
import { Router, CanActivate } from '@angular/router';
import { inject, Injectable } from '@angular/core';

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

  canActivate(): boolean {
    const jwt = sessionStorage.getItem('jwt');
    // Redirect to login if no JWT found
    if (!jwt) {
      this.router.navigate(['login']);
      return false;
    }
    return true;
  }
}
```

```
// Use in routes: { path: 'protected', component: X, canActivate: [AuthGuard] }
```

1.2. Directives

1.2.1. Filter Directives

```
// Synchronous validator directive for template-driven forms
@Directive({
  selector: '[appValidateHour]'
})
export class ValidateHour implements Validator {
  validate(control: AbstractControl): ValidationErrors | null {
    // Always check for empty/null first
    if(!control.value || control.value === "") return null;

    const parsed = parseInt(control.value);
    // Return error object if invalid, null if valid
    if(isNaN(parsed)) return { notANumber: true };
    if(parsed > 23 || parsed < 0) {
      return { numberOutOfRange: true };
    }
    return null;
  }
}

// Use in template: <input appValidateHour [(ngModel)]="hour" #hourInput="ngModel">
// Show errors: @if(hourInput.errors?.['numberOutOfRange']) {<div>Error</div>}
```

1.2.2. Async Directives

```
// Asynchronous validator for backend validation (e.g., username exists)
@Directive({
  selector: '[appRomanCheck]',
  providers: [
    {
      provide: NG_ASYNC_VALIDATORS,
      useExisting: RomanCheck,
      multi: true // Allows multiple async validators
    }
  ]
})
export class RomanCheck implements AsyncValidator {
  private readonly dataService: DataService = inject(DataService);

  // Must return Observable<ValidationErrors | null>
  validate(control: AbstractControl): Observable<ValidationErrors | null> {
    const value: string = control.value;
```

```

    return from(this.dataService.isValid(value)).pipe(
      map((result: boolean) => (result ? null : { invalid: true })))
    );
  }
}

// Control has status 'PENDING' while validating
// Use debounceTime(300) to delay validation until user stops typing

```

1.3. Routing

1.3.1. Routes

```

// Central route configuration - ORDER MATTERS!
export const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'login'}, // Default redirect
  { path: 'login', component: LoginComponent },
  { path: 'home', component: HomeComponent, canActivate: [AuthGuard]}, // Protected
  route
  { path: 'overview', component: InputDeviceOverviewComponent, canActivate:
[AuthGuard]}
  // { path: '**', component: NotFoundComponent } // Wildcard as LAST route
];

```

1.3.2. Navigation

```

// Programmatic navigation with query parameters
this.router.navigate(["/overview"], {
  queryParams: { searchTerm: this.searchTerm() }
});

// Relative navigation: router.navigate(['./relative'], {relativeTo: route})

```

1.3.3. Path Parameter

```

// Route definition with path parameter
export const routes: Routes = [
  { path: 'details/:id', component: InputDeviceDetails, canActivate: [AuthGuard]}
];

// Navigate with parameter
this.router.navigate(["/details", deviceId]);

// Read parameter in component (always string!)
const id: string | null = this.activeRoute.snapshot.paramMap.get("id");

```

```
const parsed = parseInt(id || ""); // Convert to number
```

1.3.4. Query Parameter

```
// Navigate with query parameters (?searchTerm=value)
this.router.navigate(["/overview"], {
  queryParams: { searchTerm: this.searchTerm() }
});

// Read query parameter in component (always string!)
const id: string | null = this.activeRoute.snapshot.queryParamMap.get("id");
const parsed = parseInt(id || "");
```

1.4. Forms

1.4.1. Reactive Forms

```
// Model-driven forms with FormBuilder
@Component({
  selector: 'app-login',
  template: `
    <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
      <label for="username">Username:</label>
      <input type="text" id="username" formControlName="username" />
      <label for="password">Password:</label>
      <input type="password" id="password" formControlName="password" />
      <button type="submit" [disabled]="!loginForm.valid">Login</button>
    </form>
  `,
})
export class LoginComponent {
  loginForm: FormGroup;

  constructor(private fb: FormBuilder) {
    // Build form with validators
    this.loginForm = this.fb.group({
      username: ['', Validators.required],
      password: ['', Validators.required],
    });
  }

  onSubmit() {
    if (this.loginForm.valid) {
      const { username, password } = this.loginForm.value;
      // Handle login logic
    }
  }
}
```

```
}  
  
// Import: ReactiveFormsModule
```

1.4.2. Template Driven Forms

```
// Template-based forms with ngModel (two-way binding)  
@Component({  
  selector: 'app-login',  
  template: `  
    <form (ngSubmit)="onSubmit()" #loginForm="ngForm">  
      <label for="username">Username:</label>  
      <input  
        type="text"  
        id="username"  
        name="username"  
        required  
        [(ngModel)]="username"  
      />  
      <label for="password">Password:</label>  
      <input  
        type="password"  
        id="password"  
        name="password"  
        required  
        [(ngModel)]="password"  
      />  
      <button type="submit" [disabled]="!loginForm.form.valid">Login</button>  
    </form>  
  `,  
})  
export class LoginComponent {  
  username: string = '';  
  password: string = '';  
  
  onSubmit() {  
    // Handle login logic  
  }  
}  
  
// Import: FormsModule
```

1.5. Signals

1.5.1. Models

```
// Two-way binding with signals in standalone components
```



```

// Child Component
@Component({
  selector: 'app-device-form',
  template: `
    <form>
      <label for="name">Device Name:</label>
      <input id="name" type="text" [(ngModel)]="deviceName" />

      <label for="description">Description:</label>
      <input id="description" type="text" [(ngModel)]="deviceDescription" />
    </form>
  `,
})
export class DeviceFormComponent {
  deviceName = model<string>>(''); // Creates signal with default value
  deviceDescription = model<string>>('');
}

// Parent Component - binds to child's model signals
@Component({
  template: `
    <app-device-form
      [(deviceName)]="name"
      [(deviceDescription)]="description"
    />
  `,
})
export class ParentComponent {
  name = signal<string>('Initial Device');
  description = signal<string>('Initial Description');
}

```

1.5.2. Input

```

// Signal-based inputs (replacement for @Input decorator)
@Component({
  selector: 'app-device-card',
})
export class DeviceCardComponent {
  device = input.required<InputDevice>(); // Required - compile error if missing
  isRequired = input<boolean>(false); // Optional with default
  title = input<string>('Default Title');

  // Read values: this.device(), this.title()
}

// Parent Component
@Component({
  template: `

```

```

    <app-device-card
      [device]="selectedDevice()"
      [isRequired]="true"
      [title]="'Device Details'"
    />
  ,
})
export class ParentComponent {
  selectedDevice = signal<InputDevice>({ id: 1, name: 'Device 1' });
}

```

1.5.3. Output

```

// Signal-based outputs (replacement for @Output EventEmitter)
@Component({
  selector: 'app-device-form',
  template: `
    <button (click)="handleSave()">Save</button>
    <button (click)="handleCancel()">Cancel</button>
  `,
})
export class DeviceFormComponent {
  deviceSaved = output<InputDevice>(); // Event emitter with data
  cancelled = output<void>(); // Event emitter without data

  handleSave() {
    const device: InputDevice = { id: 1, name: 'Device 1' };
    this.deviceSaved.emit(device); // Emit event with data
  }

  handleCancel() {
    this.cancelled.emit(); // Emit event without data
  }
}

// Parent Component - subscribe to events
@Component({
  template: `
    <app-device-form
      (deviceSaved)="onDeviceSaved($event)"
      (cancelled)="onCancelled()"
    />
  `,
})
export class ParentComponent {
  onDeviceSaved(device: InputDevice) {
    console.log('Device saved:', device);
  }
}

```

```

onCancelled() {
  console.log('Form cancelled');
}
}

```

1.6. HTTP Client

```

// Central HTTP service for API communication
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';
import { Observable, observeOn } from 'rxjs';
import { Game } from '../models/game.model';

const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type': 'application/json'
  })
}

@Injectable({
  providedIn: 'root'
})
export class DataService {
  private apiUrl = "https://h-aitenbichler.cloud.htl-leonding.ac.at/restserver";

  public readonly header = new HttpHeaders({
    'Content-Type': 'application/json'
  });

  constructor(private http: HttpClient) { }

  // Helper method: convert array to query params (?sudoku=1&sudoku=2)
  private toQueryParamArray(content: string[]): string {
    return "?sudoku=" + content.join("&sudoku=");
  }

  // Register in app.config.ts: provideHttpClient()
  // Observables are lazy - must .subscribe() to execute!

```

1.6.1. GET

```

// GET list of items
getGames(): Observable<Game[]> {
  return this.http.get<Game[]>(`${this.apiUrl}/game`);
}

```

```
// GET single item by ID
getGame(id: number): Observable<Game> {
  return this.http.get<Game>(`${this.apiUrl}/game/${id}`);
}
```

1.6.2. POST

```
// POST create new item
postGame(game: Game): Observable<Game> {
  return this.http.post<Game>(`${this.apiUrl}/game`, game);
}
```

1.6.3. PUT

```
// PUT full update (replaces entire resource)
putGame(id: number, game: Game) {
  return this.http.put(`${this.apiUrl}/game/${id}`, game,
    {
      headers: this.header,
      observe: 'response' // Returns HttpResponse instead of body
    });
}
```

1.6.4. PATCH

```
// PATCH partial update (updates only specified fields)
patchGame(id: number, partialGame: Partial<Game>) {
  return this.http.patch(`${this.apiUrl}/game/${id}`, partialGame,
    {
      headers: this.header,
      observe: 'response'
    });
}
```

1.6.5. DELETE

```
// DELETE remove item
deleteGame(id: number) {
  return this.http.delete(`${this.apiUrl}/game/${id}`, {
    headers: this.header,
    observe: 'response'
  });
}
```

1.7. Template Syntax

1.7.1. @for

```
<!-- Iterate over array - track helps Angular optimize rendering -->
@for (device of devices; track device) {
  <div>{{ device.name }}</div>
}

<!-- Alternative: track by property -->
@for (device of devices; track device.id) {
  <div>{{ device.name }}</div>
}
```

1.7.2. @if

```
<!-- Conditional rendering -->
@if (isLoggedIn) {
  <div>Welcome back, user!</div>
} @else {
  <div>Please log in to continue.</div>
}

<!-- Without else -->
@if (errorMessage) {
  <div class="alert alert-danger">{{ errorMessage }}</div>
}
```

1.8. Templates

1.8.1. Table

```
<!-- Data table with iteration -->
<table>
  <thead>
    <tr>
      <th>Device</th>
      <th>Description</th>
      <th>Reservation</th>
      <th></th>
    </tr>
  </thead>
  @for (device of filteredDevices; track device) {
    <tbody>
      <tr class="border-bottom">
        <td>{{ device.name }}</td>
```

```

        <td>{{ device.description }}</td>
        <td>{{ formatReservationDates(device.reservations) }}</td>
        <td>
            <button class="btn btn-secondary" (click)="detailInputDevice(device.id)">
                Reserve
            </button>
        </td>
    </tr>
</tbody>
</table>

```

1.8.2. Input Form

```

// Complete form component with validation and submit
import { Component, OnInit, inject } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from
'@angular/forms';
import { Router, ActivatedRoute } from '@angular/router';
import { DeviceService } from '../services/device.service';
import { InputDevice } from '../models/input-device.model';

@Component({
  selector: 'app-device-form',
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: './device-form.component.html'
})
export class DeviceFormComponent implements OnInit {
  private fb = inject(FormBuilder);
  private router = inject(Router);
  private route = inject(ActivatedRoute);
  private deviceService = inject(DeviceService);

  deviceForm!: FormGroup;
  isEditMode = false;
  deviceId?: number;
  errorMessage = '';

  ngOnInit(): void {
    this.initializeForm();
    this.checkEditMode();
  }

  private initializeForm(): void {
    // Initialize form with validators
    this.deviceForm = this.fb.group({
      name: ['', [Validators.required, Validators.minLength(3)]],
      description: ['', [Validators.required, Validators.maxLength(500)]],

```

```

        serialNumber: ['', [Validators.required, Validators.pattern(/^[A-Z0-9-]+$/)]],
        location: ['', Validators.required],
        status: ['available', Validators.required]
    });
}

private checkEditMode(): void {
    // Check if editing existing item via route parameter
    const id = this.route.snapshot.paramMap.get('id');
    if (id) {
        this.isEditMode = true;
        this.deviceId = parseInt(id);
        this.loadDevice(this.deviceId);
    }
}

private loadDevice(id: number): void {
    // Load existing device for editing
    this.deviceService.getDevice(id).subscribe({
        next: (device) => {
            this.deviceForm.patchValue({
                name: device.name,
                description: device.description,
                serialNumber: device.serialNumber,
                location: device.location,
                status: device.status
            });
        },
        error: (error) => {
            this.errorMessage = 'Failed to load device';
            console.error(error);
        }
    });
}

onSubmit(): void {
    // Validate and submit form
    if (this.deviceForm.invalid) {
        this.deviceForm.markAllAsTouched(); // Show all errors
        return;
    }

    const device: InputDevice = this.deviceForm.value;

    // Choose PUT or POST based on edit mode
    const request$ = this.isEditMode
        ? this.deviceService.putDevice(this.deviceId!, device)
        : this.deviceService.postDevice(device);

    request$.subscribe({
        next: (response) => {

```

```

        this.router.navigate(['/devices']);
    },
    error: (error) => {
        this.errorMessage = 'Failed to save device';
        console.error(error);
    }
});
}

onCancel(): void {
    this.router.navigate(['/devices']);
}

// Helper: check if control has specific error
hasError(controlName: string, errorType: string): boolean {
    const control = this.deviceForm.get(controlName);
    return !(control?.hasError(errorType) && (control?.dirty || control?.touched));
}

// Helper: get error message for control
getErrorMessage(controlName: string): string {
    const control = this.deviceForm.get(controlName);
    if (!control || !control.errors) return '';

    if (control.hasError('required')) return `${controlName} is required`;
    if (control.hasError('minlength')) {
        const minLength = control.errors['minlength'].requiredLength;
        return `${controlName} must be at least ${minLength} characters`;
    }
    if (control.hasError('maxlength')) {
        const maxLength = control.errors['maxlength'].requiredLength;
        return `${controlName} must not exceed ${maxLength} characters`;
    }
    if (control.hasError('pattern')) return `${controlName} has invalid format`;

    return 'Invalid input';
}
}

```

```

<!-- filepath: device-form.component.html -->
<!-- Form template with validation feedback -->
<div class="container mt-4">
    <div class="card">
        <div class="card-header">
            <h2>{{ isEditMode ? 'Edit Device' : 'New Device' }}</h2>
        </div>
        <div class="card-body">
            <!-- Error alert -->
            @if (errorMessage) {
                <div class="alert alert-danger">{{ errorMessage }}</div>
            }
        </div>
    </div>
</div>

```



```

}

<form [formGroup]="deviceForm" (ngSubmit)="onSubmit()">
  <!-- Name field with validation -->
  <div class="mb-3">
    <label for="name" class="form-label">Device Name *</label>
    <input
      type="text"
      id="name"
      class="form-control"
      formControlName="name"
      [class.is-invalid]="hasError('name', 'required') || hasError('name',
'minlength'))"
    />
    @if (hasError('name', 'required') || hasError('name', 'minlength')) {
      <div class="invalid-feedback">{{ getErrorMessage('name') }}</div>
    }
  </div>

  <!-- Description field -->
  <div class="mb-3">
    <label for="description" class="form-label">Description *</label>
    <textarea
      id="description"
      class="form-control"
      rows="3"
      formControlName="description"
      [class.is-invalid]="hasError('description', 'required') ||
hasError('description', 'maxlength'))"
    ></textarea>
    @if (hasError('description', 'required') || hasError('description',
'maxlength')) {
      <div class="invalid-feedback">{{ getErrorMessage('description') }}</div>
    }
  </div>

  <!-- Serial Number field -->
  <div class="mb-3">
    <label for="serialNumber" class="form-label">Serial Number *</label>
    <input
      type="text"
      id="serialNumber"
      class="form-control"
      formControlName="serialNumber"
      placeholder="ABC-123-XYZ"
      [class.is-invalid]="hasError('serialNumber', 'required') ||
hasError('serialNumber', 'pattern'))"
    />
    @if (hasError('serialNumber', 'required') || hasError('serialNumber',
'pattern')) {
      <div class="invalid-feedback">{{ getErrorMessage('serialNumber') }}</div>
    }
  </div>

```

```

    }
  </div>

  <!-- Location field -->
  <div class="mb-3">
    <label for="location" class="form-label">Location *</label>
    <input
      type="text"
      id="location"
      class="form-control"
      formControlName="location"
      [class.is-invalid]="hasError('location', 'required')"
    />
    @if (hasError('location', 'required')) {
      <div class="invalid-feedback">{{ getErrorMessage('location') }}</div>
    }
  </div>

  <!-- Status dropdown -->
  <div class="mb-3">
    <label for="status" class="form-label">Status *</label>
    <select
      id="status"
      class="form-select"
      formControlName="status"
      [class.is-invalid]="hasError('status', 'required')"
    >
      <option value="available">Available</option>
      <option value="in-use">In Use</option>
      <option value="maintenance">Maintenance</option>
      <option value="retired">Retired</option>
    </select>
    @if (hasError('status', 'required')) {
      <div class="invalid-feedback">{{ getErrorMessage('status') }}</div>
    }
  </div>

  <!-- Action buttons - submit disabled if form invalid -->
  <div class="d-flex gap-2">
    <button
      type="submit"
      class="btn btn-primary"
      [disabled]="deviceForm.invalid"
    >
      {{ isEditMode ? 'Update' : 'Create' }}
    </button>
    <button
      type="button"
      class="btn btn-secondary"
      (click)="onCancel()"
    >

```

```

        Cancel
      </button>
    </div>
  </form>
</div>
</div>
</div>

```

1.8.3. Details View

```

// Detail view component - loads single item by ID
import { Component, OnInit, inject, signal } from '@angular/core';
import { Router, ActivatedRoute, RouterLink } from '@angular/router';
import { CommonModule } from '@angular/common';
import { DeviceService } from '../services/device.service';
import { InputDevice } from '../models/input-device.model';

@Component({
  selector: 'app-device-details',
  standalone: true,
  imports: [CommonModule, RouterLink],
  templateUrl: './device-details.component.html'
})
export class DeviceDetailsComponent implements OnInit {
  private router = inject(Router);
  private route = inject(ActivatedRoute);
  private deviceService = inject(DeviceService);

  device = signal<InputDevice | null>(null);
  isLoading = signal<boolean>(true);
  errorMessage = signal<string>('');

  ngOnInit(): void {
    this.loadDevice();
  }

  private loadDevice(): void {
    // Extract ID from route parameter
    const idParam = this.route.snapshot.paramMap.get('id');

    if (!idParam) {
      this.errorMessage.set('Invalid device ID');
      this.isLoading.set(false);
      return;
    }

    const deviceId = parseInt(idParam);

    if (isNaN(deviceId)) {

```

```

        this.errorMessage.set('Invalid device ID format');
        this.isLoading.set(false);
        return;
    }

    // Load device from API
    this.deviceService.getDevice(deviceId).subscribe({
        next: (device) => {
            this.device.set(device);
            this.isLoading.set(false);
        },
        error: (error) => {
            console.error('Failed to load device:', error);
            if (error.status === 404) {
                this.errorMessage.set('Device not found');
            } else {
                this.errorMessage.set('Failed to load device details');
            }
            this.isLoading.set(false);
        }
    });
}

onEdit(): void {
    const id = this.device()?.id;
    if (id) {
        this.router.navigate(['/devices', 'edit', id]);
    }
}

onDelete(): void {
    const device = this.device();
    if (!device) return;

    if (confirm(`Are you sure you want to delete ${device.name}?`)) {
        this.deviceService.deleteDevice(device.id).subscribe({
            next: () => {
                this.router.navigate(['/devices']);
            },
            error: (error) => {
                console.error('Failed to delete device:', error);
                this.errorMessage.set('Failed to delete device');
            }
        });
    }
}

onBack(): void {
    this.router.navigate(['/devices']);
}

```

```
// Helper: CSS class based on status
getStatusBadgeClass(status: string): string {
  const statusClasses: { [key: string]: string } = {
    'available': 'bg-success',
    'in-use': 'bg-warning',
    'maintenance': 'bg-info',
    'retired': 'bg-secondary'
  };
  return statusClasses[status] || 'bg-secondary';
}

// Helper: format date
formatDate(date: string | Date): string {
  return new Date(date).toLocaleDateString('de-AT', {
    year: 'numeric',
    month: '2-digit',
    day: '2-digit'
  });
}
}
```

```
<!-- Details view template with loading and error states -->
<div class="container mt-4">
  <!-- Loading spinner -->
  @if (isLoading()) {
    <div class="text-center">
      <div class="spinner-border" role="status">
        <span class="visually-hidden">Loading...</span>
      </div>
      <p class="mt-2">Loading device details...</p>
    </div>
  }

  <!-- Error message -->
  @if (errorMessage()) {
    <div class="alert alert-danger">
      {{ errorMessage() }}
      <button class="btn btn-sm btn-outline-danger ms-3" (click)="onBack()">
        Back to Overview
      </button>
    </div>
  }

  <!-- Device details -->
  @if (device() && !isLoading() && !errorMessage()) {
    <div class="card">
      <div class="card-header d-flex justify-content-between align-items-center">
        <h2 class="mb-0">Device Details</h2>
        <div class="d-flex gap-2">
          <button class="btn btn-primary" (click)="onEdit()">
```

```

        <i class="bi bi-pencil"></i> Edit
    </button>
    <button class="btn btn-danger" (click)="onDelete()">
        <i class="bi bi-trash"></i> Delete
    </button>
    <button class="btn btn-secondary" (click)="onBack()">
        <i class="bi bi-arrow-left"></i> Back
    </button>
</div>
</div>

<div class="card-body">
    <div class="row">
        <div class="col-md-6">
            <dl class="row">
                <dt class="col-sm-4">Name:</dt>
                <dd class="col-sm-8">{{ device()!.name }}</dd>

                <dt class="col-sm-4">Serial Number:</dt>
                <dd class="col-sm-8">
                    <code>{{ device()!.serialNumber }}</code>
                </dd>

                <dt class="col-sm-4">Status:</dt>
                <dd class="col-sm-8">
                    <span class="badge {{ getStatusBadgeClass(device()!.status) }}">
                        {{ device()!.status }}
                    </span>
                </dd>

                <dt class="col-sm-4">Location:</dt>
                <dd class="col-sm-8">{{ device()!.location }}</dd>
            </dl>
        </div>

        <div class="col-md-6">
            <dl class="row">
                <dt class="col-sm-4">Created:</dt>
                <dd class="col-sm-8">{{ formatDate(device()!.createdAt) }}</dd>

                <dt class="col-sm-4">Last Updated:</dt>
                <dd class="col-sm-8">{{ formatDate(device()!.updatedAt) }}</dd>

                <dt class="col-sm-4">ID:</dt>
                <dd class="col-sm-8">
                    <code>{{ device()!.id }}</code>
                </dd>
            </dl>
        </div>
    </div>
</div>

```

```

<div class="row mt-3">
  <div class="col-12">
    <h5>Description</h5>
    <p class="text-muted">{{ device()!.description }}</p>
  </div>
</div>

<!-- Optional: Reservations list -->
@if (device()!.reservations && device()!.reservations.length > 0) {
  <div class="row mt-3">
    <div class="col-12">
      <h5>Reservations</h5>
      <ul class="list-group">
        @for (reservation of device()!.reservations; track reservation.id) {
          <li class="list-group-item">
            <strong>{{ reservation.userName }}</strong>
            <br />
            <small class="text-muted">
              {{ formatDate(reservation.startDate) }} -
              {{ formatDate(reservation.endDate) }}
            </small>
          </li>
        }
      </ul>
    </div>
  </div>
}

<!-- Optional: Specifications -->
@if (device()!.specifications) {
  <div class="row mt-3">
    <div class="col-12">
      <h5>Specifications</h5>
      <dl class="row">
        @for (spec of device()!.specifications | keyvalue; track spec.key) {
          <dt class="col-sm-3">{{ spec.key }}:</dt>
          <dd class="col-sm-9">{{ spec.value }}</dd>
        }
      </dl>
    </div>
  </div>
}
</div>

<div class="card-footer text-muted">
  <small>
    Device ID: {{ device()!.id }} |
    Last updated: {{ formatDate(device()!.updatedAt) }}
  </small>
</div>
</div>

```

```
}  
</div>
```


2. Backend

2.1. CDI (Dependency Injection)

```
// Register services BEFORE var app = builder.Build()

// AddTransient: new instance each time (stateless services)
builder.Services.AddTransient<IMessageService, MessageService>();
builder.Services.AddTransient<MessageService>();

// AddScoped: one instance per HTTP request (e.g., DbContext)
builder.Services.AddScoped<IDataService, DataService>();

// AddSingleton: one instance for entire app lifetime (caching, config)
builder.Services.AddSingleton<ICacheService, CacheService>();

// Interface registration allows mocking in tests
```

2.2. Swagger

2.2.1. Swagger Builder

```
// Complete API setup with Swagger and CORS
var builder = WebApplication.CreateBuilder(args);

// Swagger configuration - AddEndpointsApiExplorer BEFORE AddSwaggerGen
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Register services
builder.Services.AddSingleton<IRomanNumerals, RomanNumerals>();

// CORS policy - allow Angular app to call API
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAngular",
        policy => policy
            .WithOrigins("http://localhost:4200") // Angular dev server
            .AllowAnyHeader()
            .AllowAnyMethod());
});

var app = builder.Build();

// Enable CORS middleware - BEFORE UseAuthorization!
app.UseCors("AllowAngular");
```

```
// Enable Swagger UI
app.UseSwagger();
app.UseSwaggerUI();
```

2.2.2. Cors

```
// CORS configuration - BEFORE app.Build()
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAngular",
        policy => policy
            .WithOrigins("http://localhost:4200") // Frontend URL
            .AllowAnyHeader() // Allow all headers
            .AllowAnyMethod()); // Allow GET, POST, PUT, DELETE, etc.
});

// Enable CORS middleware - AFTER app.Build(), BEFORE UseAuthorization()
app.UseCors("AllowAngular");

// Production: specify exact origins, headers, methods
// Development: can use .AllowAnyOrigin() but NOT with credentials!
```

2.3. Minimal API

2.3.1. GET

```
// GET endpoint - read data
app.MapGet("/isValid", (string literal, IRomanNumerals service) =>
{
    try
    {
        service.ConvertFromRomanLiteral(literal);
        return true;
    }
    catch (Exception ex) when (ex is ArgumentOutOfRangeException or ArgumentException)
    {
        return false;
    }
});

// Route parameter: /devices/{id}
// Query parameter: /devices?status=active
```

2.3.2. POST

```
// POST endpoint - create new resource
app.MapPost("/devices", (InputDevice device, IDeviceService service) =>
{
    try
    {
        var created = service.CreateDevice(device);
        // Return 201 Created with Location header
        return Results.Created($" /devices/{created.Id}", created);
    }
    catch (Exception ex)
    {
        return Results.BadRequest(ex.Message);
    }
});
```

2.3.3. PUT

```
// PUT endpoint - full update (replaces entire resource)
app.MapPut("/devices/{id}", (int id, InputDevice device, IDeviceService service) =>
{
    try
    {
        var updated = service.UpdateDevice(id, device);
        return updated != null ? Results.Ok(updated) : Results.NotFound();
    }
    catch (Exception ex)
    {
        return Results.BadRequest(ex.Message);
    }
});
```

2.3.4. PATCH

```
// PATCH endpoint - partial update (only specified fields)
app.MapPatch("/devices/{id}/status", (int id, string status, IDeviceService service)
=>
{
    try
    {
        var updated = service.UpdateDeviceStatus(id, status);
        // 204 No Content on success
        return updated ? Results.NoContent() : Results.NotFound();
    }
    catch (Exception ex)
    {

```

```

        return Results.BadRequest(ex.Message);
    }
});

```

2.3.5. DELETE

```

// DELETE endpoint - remove resource
app.MapDelete("/devices/{id}", (int id, IDeviceService service) =>
{
    try
    {
        var deleted = service.DeleteDevice(id);
        // 204 No Content on success
        return deleted ? Results.NoContent() : Results.NotFound();
    }
    catch (Exception ex)
    {
        return Results.BadRequest(ex.Message);
    }
});

```

2.4. Authentication

```

// JWT authentication setup
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true, // Check token expiration
            ValidateIssuerSigningKey = true,
            ValidIssuer = builder.Configuration["Jwt:Issuer"],
            ValidAudience = builder.Configuration["Jwt:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]))
        };
    });

// Protect endpoints with .RequireAuthorization()
app.MapGet("/protected", () => "Secret data").RequireAuthorization();

```

2.5. Services

2.5.1. Service Interface

```
// Define service contract
namespace YourProject.Services;

public interface IYourService
{
    // Define method signatures
    Task<List<Item>> GetAllItemsAsync();
    Task<Item?> GetItemByIdAsync(int id);
    Task<Item> CreateItemAsync(Item item);
    Task<bool> UpdateItemAsync(int id, Item item);
    Task<bool> DeleteItemAsync(int id);
}
```

2.5.2. Service Implementation

```
// Implement service logic
namespace YourProject.Services;

public class YourService : IYourService
{
    // Dependencies injected via constructor
    private readonly IRepository _repository;

    public YourService(IRepository repository)
    {
        _repository = repository;
    }

    public async Task<List<Item>> GetAllItemsAsync()
    {
        return await _repository.GetAllAsync();
    }

    // ...implement other methods
}
```

2.5.3. Service Registration

```
// Register service in Program.cs BEFORE app.Build()

// With interface (recommended for testing)
builder.Services.AddTransient<IYourService, YourService>();

// Without interface
builder.Services.AddTransient<YourService>();
```

```
// Scoped (per request) - for DbContext
builder.Services.AddScoped<IYourService, YourService>();

// Singleton (app lifetime) - for caching
builder.Services.AddSingleton<IYourService, YourService>();
```

2.6. Unit Tests

2.6.1. XUnit

```
// XUnit test class - [Fact] for single test, [Theory] for parameterized
using Xunit;
using YourProject.Services;

namespace YourProject.Tests;

public class YourServiceTests
{
    private readonly IYourService _yourService;

    public YourServiceTests()
    {
        // Setup runs before each test
        _yourService = new YourService();
    }

    [Fact] // Single test case
    public void YourMethod_ShouldReturnExpectedResult()
    {
        // Arrange - setup test data
        var input = "test";

        // Act - execute method
        var result = _yourService.YourMethod(input);

        // Assert - verify result
        Assert.Equal("expected", result);
    }

    [Theory] // Parameterized test
    [InlineData(1, "one")]
    [InlineData(2, "two")]
    public void YourMethod_WithParameters(int input, string expected)
    {
        var result = _yourService.YourMethod(input);
        Assert.Equal(expected, result);
    }
}
```

```
// NuGet: xunit, xunit.runner.visualstudio
```

2.6.2. NUnit

```
// NUnit test class - similar to XUnit but different attributes
using NUnit.Framework;
using YourProject.Services;

namespace YourProject.Tests;

public class YourServiceTests
{
    private IYourService _yourService;

    [SetUp] // Runs before each test
    public void Setup()
    {
        _yourService = new YourService();
    }

    [Test] // Single test case
    public void YourMethod_ShouldReturnExpectedResult()
    {
        // Arrange
        var input = "test";

        // Act
        var result = _yourService.YourMethod(input);

        // Assert
        Assert.AreEqual("expected", result);
    }

    [TestCase(1, "one")] // Parameterized test
    [TestCase(2, "two")]
    public void YourMethod_WithParameters(int input, string expected)
    {
        var result = _yourService.YourMethod(input);
        Assert.AreEqual(expected, result);
    }
}

// NuGet: NUnit, NUnit3TestAdapter
```

2.6.3. FluentAssertions

```
// FluentAssertions - more readable assertions
```

```

using FluentAssertions;
using Xunit;
using YourProject.Services;

namespace YourProject.Tests;

public class YourServiceTests
{
    private readonly IYourService _yourService;

    public YourServiceTests()
    {
        _yourService = new YourService();
    }

    [Fact]
    public void YourMethod_ShouldReturnExpectedResult()
    {
        // Arrange
        var input = "test";

        // Act
        var result = _yourService.YourMethod(input);

        // Assert - readable and chainable
        result.Should().Be("expected");
        result.Should().NotNull();
        result.Should().StartWith("exp");

        // Collections
        var list = new[] { 1, 2, 3 };
        list.Should().HaveCount(3);
        list.Should().Contain(2);

        // Exceptions
        Action act = () => _yourService.ThrowingMethod();
        act.Should().Throw<ArgumentException>()
            .WithMessage("Invalid argument");
    }
}

// NuGet: FluentAssertions

```

2.6.4. Moq

```

// Moq - mock dependencies for isolated unit tests
using Moq;
using Xunit;
using YourProject.Services;

```



```

namespace YourProject.Tests;

public class YourServiceTests
{
    private readonly Mock<IDependencyService> _dependencyServiceMock;
    private readonly IYourService _yourService;

    public YourServiceTests()
    {
        // Create mock of dependency
        _dependencyServiceMock = new Mock<IDependencyService>();

        // Inject mock into service under test
        _yourService = new YourService(_dependencyServiceMock.Object);
    }

    [Fact]
    public void YourMethod_ShouldReturnExpectedResult()
    {
        // Arrange - setup mock behavior
        var input = "test";
        _dependencyServiceMock
            .Setup(ds => ds.SomeMethod(It.IsAny<string>())) // Match any string
            .Returns("mocked result");

        // Act
        var result = _yourService.YourMethod(input);

        // Assert
        Assert.Equal("expected", result);

        // Verify mock was called exactly once
        _dependencyServiceMock.Verify(
            ds => ds.SomeMethod(It.IsAny<string>()),
            Times.Once);
    }

    [Fact]
    public void YourMethod_WithSpecificParameter()
    {
        // Setup with specific parameter value
        _dependencyServiceMock
            .Setup(ds => ds.SomeMethod("specific"))
            .Returns("specific result");

        // Setup throws exception
        _dependencyServiceMock
            .Setup(ds => ds.ThrowingMethod())
            .Throws<InvalidOperationException>();
    }
}

```

```
}  
  
// NuGet: Moq
```

2.6.5. Testen von Minimal APIs

```
// Integration tests for Minimal API using WebApplicationFactory  
using Microsoft.AspNetCore.Mvc.Testing;  
using System.Net.Http;  
using System.Threading.Tasks;  
using Xunit;  
  
namespace YourProject.Tests;  
  
// IClassFixture shares WebApplicationFactory between tests  
public class YourApiTests : IClassFixture<WebApplicationFactory<Program>>  
{  
    private readonly HttpClient _client;  
  
    public YourApiTests(WebApplicationFactory<Program> factory)  
    {  
        // Create HTTP client for testing API  
        _client = factory.CreateClient();  
    }  
  
    [Fact]  
    public async Task GetEndpoint_ShouldReturnSuccessStatusCode()  
    {  
        // Act - send GET request  
        var response = await _client.GetAsync("/your-endpoint");  
  
        // Assert - check status code and content  
        response.EnsureSuccessStatusCode(); // 2xx status  
        var content = await response.Content.ReadAsStringAsync();  
        Assert.NotNull(content);  
    }  
  
    [Fact]  
    public async Task PostEndpoint_ShouldCreateResource()  
    {  
        // Arrange  
        var json = """"{"name": "test"}""";  
        var content = new StringContent(json, Encoding.UTF8, "application/json");  
  
        // Act  
        var response = await _client.PostAsync("/devices", content);  
  
        // Assert  
        response.StatusCode.Should().Be(HttpStatusCode.Created);  
    }  
}
```

```
        var location = response.Headers.Location;
        location.Should().NotBeNull();
    }
}

// NuGet: Microsoft.AspNetCore.Mvc.Testing
// Program.cs must be public: public partial class Program { }
```