Module 4 Teaching Guide

NAVIGATION & DATA TRANSFORMATION

Student CRM Angular project ⬚ beginner-friendly walkthrough for adding a routed analytics page.

> **Goal for learners:** understand how to create a new routed page, read query parameters, and transform in-memory data into actionable summaries that the UI can display.

Before You Start

- Open the workspace folder: `student-crm/`.
- Install dependencies once with `npm install` (skip if already done).
- Run `npm start` to launch the dev server when demonstrating live updates.
- Review the Home page so learners see the starting point they are enhancing.

What We Will Build

A new **Student Insights** page available at `/students/stats`. It will display:

1. Total active and inactive students based on the in-memory service.
2. A per-track summary (Front-end, Back-end, Data, DevOps).
3. An optional track filter controlled by the query string (e.g., `?track=Data`).
4. A student list that reflects the current filter.

Step-by-Step Implementation

Step 1 ⬚ Create the Feature Folder

1. Inside src/app/students/, add a folder named `student-stats`.
2. Create three files there:
    - `student-stats.ts` ⬚ component logic.

- `student-stats.html` template layout.
- `student-stats.css` component styles.

Tip for beginners: in VS Code use New Folder followed by New File three times to avoid typos.

Step 2 Build the Component Class

Paste the following into `student-stats.ts` and walk through it slowly with the class.

```
import { Component, OnDestroy, OnInit } from '@angular/core';
import { ActivatedRoute, ParamMap, Router } from '@angular/router';
import { Subscription } from 'rxjs';
import { Student, StudentService } from '../../core/student';

@Component({
  selector: 'app-student-stats',
  standalone: false,
  templateUrl: './student-stats.html',
  styleUrls: ['./student-stats.css']
})
export class StudentStatsComponent implements OnInit, OnDestroy {
  private readonly subs = new Subscription();
  readonly tracks: Student['track'][] = ['Front-end', 'Back-end', 'D

  students: Student[] = [];
  summaries: Array<{ track: Student['track']; total: number; active:
  activeCount = 0;
  inactiveCount = 0;
  selectedTrack?: Student['track'];

  constructor(
    private readonly svc: StudentService,
    private readonly route: ActivatedRoute,
    private readonly router: Router
  ) {}

  ngOnInit(): void {
    this.students = this.svc.list();
    this.recomputeStats();
    this.subs.add(
```

```
    this.route.queryParamMap.subscribe(params => this.applyTrackFi
  );
}

ngOnDestroy(): void {
  this.subs.unsubscribe();
}

get filteredStudents(): Student[] {
  return this.selectedTrack
    ? this.students.filter(s => s.track === this.selectedTrack)
    : this.students;
}

selectTrack(track?: Student['track']): void {
  this.router.navigate([], {
    relativeTo: this.route,
    queryParams: track ? { track } : { track: null },
    queryParamsHandling: 'merge'
  });
}

private applyTrackFilter(params: ParamMap): void {
  const raw = params.get('track');
  this.selectedTrack = this.tracks.includes(raw as Student['track'
}

private recomputeStats(): void {
  const byTrack = new Map(this.tracks.map(track => [track, { track

  for (const student of this.students) {
    const summary = byTrack.get(student.track);
    if (!summary) continue;
    summary.total += 1;
    summary.active += student.active ? 1 : 0;
    summary.inactive += student.active ? 0 : 1;
  }

  const summaryValues = Array.from(byTrack.values());
  this.summaries = summaryValues.filter(item => item.total > 0);

  const totals = summaryValues.reduce((acc, item) => ({
    total: acc.total + item.total,
    active: acc.active + item.active,
```

```
      inactive: acc.inactive + item.inactive
    }), { total: 0, active: 0, inactive: 0 });

    this.activeCount = totals.active;
    this.inactiveCount = totals.inactive;
  }
}
```

> Reinforce the concepts: lifecycle hooks for setup/cleanup, a reusable getter for filtered data, and a helper method that reduces raw data into summaries.

Step 3 ⬩ Design the Template Layout

Fill `student-stats.html` with the structure below. Point out the different kinds of binding.

```
# Student Insights

Routed analytics page for Module 4.

  Back to Home
```

```
Total Students
{{ students.length }}

Active
{{ activeCount }}

Inactive
{{ inactiveCount }}

Filter by track:
All
{{ track }}
```

Track Breakdown

TRACK	STUDENTS	ACTIVE	I
{{ summary.track }}	{{ summary.total }}	{{ summary.active }}	-

```
Students {{ selectedTrack ? '(' + selectedTrack + ')' : ''

      URL filter: ?track={{ selectedTrack || 'all' }}



No students match this filter yet.




    •
              {{ student.name }}
                      {{ STUDENT.ACTIVE ? 'ACTIVE' : 'INACTIVE' }}
              {{ student.track }}
```

Step 4 ⬩ Style the Page

Copy these styles into `student-stats.css`. This keeps the look and feel consistent with the rest of the CRM.

```
.stats {
  display: grid;
  gap: 24px;
}

.stats__header {
  display: flex;
  align-items: center;
  justify-content: space-between;
  flex-wrap: wrap;
}

.stats__title { margin: 0; font-size: 1.75rem; }
.stats__subtitle { margin: 4px 0 0; color: #6b7280; }

.link {
  border: none;
  background: transparent;
  color: #2563eb;
  cursor: pointer;
  font-weight: 600;
  text-decoration: underline;
}

.stats__overview {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(140px, 1fr));
  gap: 16px;
}

.card {
  padding: 16px;
  border-radius: 12px;
  background: #f9fafb;
  border: 1px solid #e5e7eb;
  display: flex;
  flex-direction: column;
  gap: 4px;
}
```

```
.card.highlight {
  background: linear-gradient(135deg, #2563eb, #3b82f6);
  color: #fff;
  border: none;
}

.label {
  font-size: 0.85rem;
  text-transform: uppercase;
  letter-spacing: 0.04em;
}

.value { font-size: 1.9rem; font-weight: 700; }

.stats__filters {
  display: flex;
  align-items: center;
  flex-wrap: wrap;
  gap: 8px;
}

.chip {
  border: 1px solid #d1d5db;
  background: #fff;
  padding: 6px 16px;
  border-radius: 999px;
  cursor: pointer;
  transition: all 0.15s ease-in-out;
}

.chip.active {
  background: #1d4ed8;
  border-color: #1d4ed8;
  color: #fff;
}

.chip:hover { border-color: #2563eb; }

.stats__table table {
  width: 100%;
  border-collapse: collapse;
}

.stats__table th,
```

```
.stats__table td {
  padding: 10px;
  border-bottom: 1px solid #e5e7eb;
  text-align: left;
}

.stats__table th {
  text-transform: uppercase;
  font-size: 0.75rem;
  letter-spacing: 0.05em;
  color: #6b7280;
}

.table__title { margin: 0 0 10px; }

.stats__list {
  border: 1px solid #e5e7eb;
  border-radius: 12px;
  padding: 16px;
  background: #fff;
}

.list__header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 10px;
}

.student-list {
  list-style: none;
  padding: 0;
  margin: 0;
  display: grid;
  gap: 10px;
}

.student-row {
  display: grid;
  grid-template-columns: 1fr auto auto;
  align-items: center;
  gap: 12px;
  padding: 10px;
  border: 1px solid #e5e7eb;
```

```
  border-radius: 8px;
}

.student-name { font-weight: 600; }

.badge {
  font-size: 0.75rem;
  padding: 3px 10px;
  border-radius: 999px;
  text-transform: uppercase;
  letter-spacing: 0.05em;
}

.badge.on { background: #dcfce7; color: #15803d; }
.badge.off { background: #fee2e2; color: #b91c1c; }

@media (max-width: 640px) {
  .student-row { grid-template-columns: 1fr; gap: 6px; }
  .list__header { flex-direction: column; align-items: flex-start; g
}
```

Step 5 ⬩ Register the Component and Route

FILE	WHAT TO ADD
`src/app/students/students-module.ts`	Import `StudentStatsComponent` and `RouterModule`, then declare and export the component.
`src/app/app-routing-module.ts`	Append the route `{ path: 'students/stats', component: StudentStatsComponent }`.
`src/app/app.html`	Add the navigation link labeled ⬩Insights⬩ with `routerLink="/students/stats"`.

> Mention the common pitfall: if the component is not declared, Angular will throw ⬩component is not part of any NgModule⬩. Always wire it up in the feature module.

Step 6 ⬩ Verify Everything Works

1. Run `npm run build` or keep `ng serve` running to spot template errors instantly.
2. Open `http://localhost:4200` and click **Insights** in the header.
3. Toggle the track chips and highlight how the URL query string changes (e.g., `?track=DevOps`).
4. Return to Home, add a new student, then revisit Insights to prove the aggregates update automatically.

> Encourage learners to watch the browser console; they will see the new component render without errors if everything is wired correctly.

Teaching Checklist

- Differentiate between navigation state (route & params) and shared application state (students in the service).
- Show how to clean up subscriptions with `ngOnDestroy` to avoid memory leaks.
- Explain the reducer pattern used in `recomputeStats` and why we avoid mutating the original service array.
- Demonstrate `*ngFor` and `*ngIf` reacting to computed data.
- Discuss fallbacks: if someone types an unsupported track in the URL, the filter resets to ⬩All⬩.

Practice Ideas

1. Add a second query parameter (e.g., `?onlyActive=true`) and update the getter to honor both filters.

2. Move the aggregation logic into a dedicated method on `StudentService` or convert the service to return an Observable.
3. Create an extra card that displays the newest student using `Math.max` on IDs.

Appendix ⬩ Useful Commands

```
# Install dependencies (run once)
npm install

# Start the dev server
npm start

# Build production bundles
npm run build
```

Prepared for Module 4 of the Angular training series. Generated 2025-11-01.