Track 3: .NET + Angular — Detailed Curriculum (29 Hours Core)

> *Total with AI + Agile + Project: 58 hours*
>
> *Backend: .NET 10 with ASP.NET Core Web API*
>
> *Frontend: Angular 20*
>
> *Database: PostgreSQL + MongoDB*
>
> *Target: Final-year BE/BTech/BCA/MCA students*

Module 1: Development Setup & Git (2 Hours)

Session 1 (2 hrs) — Dev Environment & Version Control

Topics:

- Installing .NET 10 SDK, VS Code with C# Dev Kit / Visual Studio Community
- Node.js, Angular CLI setup
- PostgreSQL installation & pgAdmin
- Git fundamentals refresher — init, add, commit, branch, merge
- GitHub account setup, SSH keys
- Pull request workflow — branch → commit → push → PR → review → merge
- `.gitignore` for .NET/Angular projects
- GitHub Copilot installation and activation

Hands-on:

- Create a .NET Web API project with `dotnet new webapi`
- Create a GitHub repo, make a branch, submit a PR, merge it
- First Copilot interaction — let it autocomplete a C# class

Module 2: C# & ASP.NET Core Backend (11 Hours)

Session 2 (2 hrs) — C# Refresher & ASP.NET Core Introduction

Topics:

- C# modern features students may not know — records, pattern matching, nullable reference types, global usings, file-scoped namespaces
- LINQ — `Where`, `Select`, `OrderBy`, `GroupBy`, `FirstOrDefault`, query vs method syntax
- Async/await — `Task<T>`, `async`, `await`, why it matters for APIs
- ASP.NET Core project structure — Program.cs, Controllers, Models, Services

- Minimal API vs Controller-based API — when to use which
- `[ApiController]`, `[Route]`, `[HttpGet]`, `[HttpPost]`
- Running and debugging — `dotnet run`, `dotnet watch`, launch profiles

Hands-on:

- Create an ASP.NET Core Web API project
- Write LINQ queries on a collection
- Build a "Hello World" controller with GET and POST endpoints
- Explore project structure and middleware pipeline

Session 3 (2 hrs) — REST API Fundamentals & Basic Security

Topics:

- REST principles — resources, HTTP methods (GET, POST, PUT, DELETE), status codes
- DTOs (Data Transfer Objects) — why not expose entities, request/response separation
- Model validation — `[Required]`, `[StringLength]`, `[EmailAddress]`, `[Range]`
- `FluentValidation` — rules, custom validators, dependency injection
- Action results — `Ok()`, `NotFound()`, `BadRequest()`, `CreatedAtAction()`
- Global exception handling — middleware, `ProblemDetails`
- Basic API security — why APIs need protection, authentication vs authorization
- JWT overview — what tokens are, how they secure APIs (detailed setup in Module 5)
- API testing with Postman and Swagger

Hands-on:

- Build a complete CRUD API for a `Student` resource
- Create DTOs — `CreateStudentRequest`, `StudentResponse`, `UpdateStudentRequest`
- Add validation with FluentValidation — name required, email format, age range
- Global exception middleware with structured error responses
- Test all endpoints in Swagger and Postman

Session 4 (2 hrs) — Database Integration with Entity Framework Core

Topics:

- Entity Framework Core setup — NuGet packages, `Npgsql.EntityFrameworkCore.PostgreSQL`
- `DbContext` — configuration, `OnModelCreating`, connection string
- Code-first — entity classes, `DbSet<T>`, data annotations vs Fluent API
- Relationships — one-to-many, many-to-many, navigation properties
- EF Core migrations — `dotnet ef migrations add`, `dotnet ef database update`
- CRUD with EF Core — `AddAsync`, `FindAsync`, `Update`, `Remove`, `SaveChangesAsync`
- Querying — `Include()`, `ThenInclude()`, `AsNoTracking()`

Hands-on:

- Connect Student API to PostgreSQL via EF Core
- Add a `Course` entity with one-to-many relationship
- Create and run migrations
- Implement repository pattern with EF Core
- Verify data in pgAdmin

Session 5 (2 hrs) — Service Layer Design & Dependency Injection

Topics:

- Service layer design — interfaces and implementations
- Dependency Injection — `AddScoped`, `AddTransient`, `AddSingleton`, constructor injection
- Interface-based architecture — why program to interfaces, testability benefits
- Repository pattern — generic repository, `IRepository<T>`
- Unit of Work pattern — coordinating multiple repositories
- Project structure — organizing by feature vs by layer

Hands-on:

- Refactor Student API to proper service/repository pattern
- Implement generic repository with Unit of Work
- Practice DI registration — experiment with Scoped vs Transient vs Singleton lifetimes
- Reorganize project structure by feature

Session 6 (2 hrs) — Pagination, Filtering & Data Mapping

Topics:

- AutoMapper — mapping entities to DTOs, profile configuration, reverse mapping
- Pagination — `Skip()`, `Take()`, `PagedResult<T>` response model
- Sorting — dynamic `OrderBy` with expressions, multi-column sort
- Filtering — query parameters, dynamic filters, search across fields
- Specification pattern intro — encapsulating query logic in reusable specs
- Response envelope — standardized API response format (`ApiResponse<T>`)

Hands-on:

- Add AutoMapper for entity-DTO mapping with profiles
- Add pagination and sorting to list endpoints
- Add dynamic filtering (by name, course, date range)
- Implement a specification for complex student queries
- Wrap responses in standardized envelope

Session 7 (2 hrs) — Testing & Code Quality

Topics:

- Why testing matters — confidence, refactoring safety, CI/CD readiness
- Unit testing with xUnit — `[Fact]`, `[Theory]`, `[InlineData]`, assertions
- Mocking with Moq — `Mock<IRepository>`, `Setup().Returns()`, `Verify()`
- Testing service layer — isolate business logic from infrastructure
- Integration testing with `WebApplicationFactory` — test real HTTP pipeline
- Test organization — unit vs integration, naming conventions
- Code coverage basics — what to measure, what coverage is meaningful

Hands-on:

- Write 5 unit tests — service layer with mocked repository
- Write 2 integration tests with `WebApplicationFactory`

- Test CRUD operations end-to-end (create, read, update, delete)
- Practice test-first: write a test, then implement the feature

Session 8 (1 hr) — API Documentation & Best Practices

Topics:

- Swagger/OpenAPI — `Swashbuckle` configuration, XML comments
- Documenting endpoints with `[ProducesResponseType]`, summaries
- API versioning — `Asp.Versioning.Http`
- CORS configuration — `AddCors()`, `UseCors()`, policy-based
- Configuration — `appsettings.json`, `appsettings.Development.json`, options pattern
- Logging — `ILogger<T>`, Serilog setup, structured logging

Hands-on:

- Customize Swagger with proper descriptions and JWT auth button
- Configure CORS for `http://localhost:4200` (Angular dev server)
- Move config to options pattern with `IOptions<T>`

Module 3: Database Deep Dive (3 Hours)

Session 9 (2 hrs) — PostgreSQL Mastery

Topics:

- Schema design principles — normalization (1NF, 2NF, 3NF), when to denormalize
- Data types — VARCHAR, INTEGER, BOOLEAN, TIMESTAMP, JSONB, UUID
- Constraints — PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL
- Indexes — B-tree, when to create, covering indexes, `EXPLAIN ANALYZE`
- Complex queries — JOINs (INNER, LEFT, RIGHT), subqueries, CTEs (`WITH`)
- Aggregate functions — GROUP BY, HAVING, window functions intro
- Transactions — ACID properties, isolation levels

Hands-on:

- Design a schema for the capstone project (3-4 tables with relationships)
- Write 5 complex queries with JOINs and aggregations
- Create indexes and compare `EXPLAIN ANALYZE` before/after
- Practice with CTEs for readable complex queries

Session 10 (1 hr) — Advanced ORM Patterns

Topics:

- N+1 query problem — what it is, how to detect (SQL logging), how to fix
- Eager loading — `Include()`, `ThenInclude()`, split queries
- Projections — `Select()` to DTOs, anonymous types for performance
- Raw SQL — `FromSqlRaw()`, `ExecuteSqlRaw()` for complex queries

- Interceptors — audit logging with `SaveChangesInterceptor`

Hands-on:

- Identify and fix N+1 query in Student-Course relationship
- Implement audit fields (CreatedAt, UpdatedAt) with interceptor
- Compare EF Core generated SQL vs hand-written for a report query

---

Module 4: Angular Frontend (9 Hours)

Session 11 (2 hrs) — Angular Fundamentals

Topics:

- Angular architecture — modules, components, templates, services
- Angular CLI — `ng new`, `ng generate`, `ng serve`
- TypeScript essentials for Angular — types, interfaces, decorators, generics
- Components — `@Component`, templates, styles, lifecycle hooks
- Data binding — interpolation `{{ }}`, property `[attr]`, event `(click)`, two-way `[(ngModel)]`
- Directives — `*ngIf`, `*ngFor`, `[ngClass]`, `[ngStyle]`
- Pipes — built-in pipes, custom pipes

Hands-on:

- Create Angular project, generate 3 components
- Build a student list with `*ngFor`, conditional display with `*ngIf`
- Create a custom pipe (e.g., capitalize name, format date)

Session 12 (2 hrs) — Routing & Navigation

Topics:

- Angular Router — `RouterModule`, route configuration, `<router-outlet>`
- Route parameters — `ActivatedRoute`, `:id` params
- Lazy loading modules — `loadChildren`, performance benefits
- Route guards — `CanActivate`, `CanDeactivate`, auth guards
- Navigation — `routerLink`, programmatic navigation with `Router`
- Layout components — shared header, sidebar, footer
- Child routes and nested layouts

Hands-on:

- Set up routing: Dashboard, Student List, Student Detail, Login
- Implement lazy-loaded feature modules
- Create an auth guard that redirects unauthenticated users to login
- Build a layout with header navigation

Session 13 (2 hrs) — State Management & Forms

Topics:

- Reactive Forms — `FormGroup`, `FormControl`, `FormBuilder`, validators
- Custom validators — sync and async
- Form submission and error display patterns
- Component communication — `@Input()`, `@Output()`, `EventEmitter`
- Services as state containers — `BehaviorSubject`, shared state
- RxJS essentials — `Observable`, `Subject`, `pipe`, `map`, `filter`, `switchMap`

Hands-on:

- Build a student registration form with reactive forms and validation
- Display inline validation errors
- Create a shared auth service with `BehaviorSubject` for login state
- Use RxJS operators to transform API responses

Session 14 (2 hrs) — HTTP & API Integration

Topics:

- `HttpClient` — GET, POST, PUT, DELETE
- Interceptors — adding JWT token to every request
- Error handling — global error interceptor, user-friendly messages
- Loading states — spinners, skeleton screens
- Environment configuration — `environment.ts` for API URLs
- CORS — understanding and debugging cross-origin issues
- File uploads — `FormData`, multipart requests

Hands-on:

- Connect Angular to ASP.NET Core API
- Implement JWT auth flow — login, store token, attach to requests
- Build Student CRUD screens — list (with pagination), create form, edit, delete
- Add loading spinner and error toast notifications

Session 15 (1 hr) — UI Polish & Responsive Design

Topics:

- Angular Material or Tailwind CSS — quick setup for professional UI
- Responsive layouts — CSS Grid, Flexbox, media queries
- Table components — sorting, filtering, pagination
- Charts — ng2-charts or Chart.js for dashboards
- Build optimization — `ng build --prod`, bundle analysis
- Deployment basics — building for production, static hosting

Hands-on:

- Style the student management app with Angular Material or Tailwind
- Add a dashboard with 2 chart widgets (students per course, enrollment trends)
- Verify responsive design on mobile viewport

- Run production build

Module 5: Authentication & Authorization (2 Hours)

Session 16 (2 hrs) — Full Authentication & Authorization

> *Taught after frontend so students can implement the complete auth flow end-to-end (backend + frontend).*

Topics:

- Authentication concepts — stateless APIs, tokens vs sessions
- JWT (JSON Web Tokens) — structure, claims, signing, validation
- ASP.NET Core Identity basics — `UserManager`, `SignInManager`
- JWT setup — `AddAuthentication()`, `AddJwtBearer()`, token generation
- Password hashing — built-in with ASP.NET Core Identity
- `[Authorize]` attribute, role-based — `[Authorize(Roles = "Admin")]`
- Claims-based authorization — policies, requirements
- Protecting endpoints — anonymous vs authenticated vs admin-only
- Frontend auth flow — login page, token storage, route guards, interceptors

Hands-on:

- Add user registration and login endpoints to backend
- Implement JWT token generation and validation
- Protect Student CRUD — only authenticated users can create/update/delete
- Add ADMIN role — only admins can delete students
- Connect Angular login page to backend auth API
- Test full flow: register → login → get token → access protected routes

---

Module 6: AI Tools for Developers (15 Hours)

> *See AI-TOOLS-MODULE.md for detailed curriculum.*

> *This module is shared across all three tracks.*

Module 7: Agile & Scrum Workshop (2 Hours)

> *See CAPSTONE-PROJECT.md for the Agile workshop session.*

> *Teaches Scrum framework, user stories, estimation, GitHub Projects setup.*

> *Shared across all three tracks.*

Module 8: Capstone Project Sprint (12 Hours)

> *See CAPSTONE-PROJECT.md for project guidelines and sprint structure.*

> *Track 3 stack: .NET 10 (ASP.NET Core) + Angular + PostgreSQL + AI features*

> *Run in 2 real Agile sprints with standups, feature branches, PR reviews, and retrospective.*

Track 3 Summary

Module	Hours	Key Technologies
Dev Setup & Git	2	.NET 10 SDK, VS Code/Visual Studio, Git, GitHub, Copilot
C# & ASP.NET Core	13	ASP.NET Core, EF Core, xUnit, Moq, Swagger
Database	3	PostgreSQL, EF Core Migrations, advanced ORM patterns
Angular	9	Angular 20, TypeScript, RxJS, Reactive Forms, Angular Material
Authentication	2	JWT, ASP.NET Core Identity, role-based access
AI Tools	15	Copilot, ChatGPT, Prompt Eng, OpenAI API, RAG, MCP
Agile & Scrum	2	Scrum, user stories, estimation, GitHub Projects
Capstone Project	12	Full-stack app in Agile sprints with AI features
Total	58	