DNP Slides

Eduard Fekete

Character Count: 13173
Word Count: 1712

December 23, 2025

1 .NET Programming - Comprehensive Course Notes

1.1 1. Course Overview & Introduction

1.1.1 1.1 Course Structure

- **Focus**: Full stack development using C# .NET.
  - Server side (Web API)
  - Client side (Blazor)
  - Database (Entity Framework Core)

1.1.2 1.2 The .NET Ecosystem

- **.NET**: An umbrella term covering many languages, frameworks, libraries, and technologies.
- **CLR (Common Language Runtime)**: The "in-memory" application that manages execution. It translates Intermediate Language (IL) to Native Code via JIT (Just-In-Time) compilation.
- **IL (Intermediate Language)**: Higher-level languages like C# compile down to this hardware-agnostic instruction set.
- **Managed Code**: Code whose execution is managed by a runtime (CLR), providing services like garbage collection and type safety.

1.1.3 1.3 C# Basics (vs Java)

- **Similarities**: Object-oriented, strong static typing, garbage collection, exceptions.
- **Differences**:
  - **Namespaces**: Similar to Java packages. Scoped namespaces (file-level) are the modern approach (`namespace Examples;`).
  - **Properties**: Replaces verbose Getter/Setter methods.
    - *Auto-property*: `public string Name { get; set; }`
    - *ReadOnly*: `public string Name { get; private set; }`
    - *Required*: `public required string Name { get; set; }` (Forces initialization).
    - *Init-only*: `public string Name { get; init; }` (Can only be set during initialization).
  - **Object Initializers**: `Person p = new() { Name = "John" };`
  - **String Interpolation**: `$"Hello {name}"` instead of concatenation.
  - **Top-level statements**: `Console.WriteLine("Hello");` (No explicit Main method needed in simple programs).
- **Types**:
  - *Value Types*: Allocated on the stack (int, bool, structs).
  - *Reference Types*: Allocated on the heap (classes, strings, arrays).

1.2 2. Advanced C# Concepts

1.2.1 2.1 Collections

- **Arrays**: Fixed size. `int[] nums = [1, 2, 3];`
- **Lists**: Dynamic size. `List<int> nums = [1, 2, 3];`
- **Common Collections**: `Dictionary<TKey, TValue>`, `HashSet<T>`, `Queue<T>`, `Stack<T>`.
- **IEnumerable**: The base interface for iterating over collections.

1.2.2 2.2 Nullable Reference Types

- **Problem**: NullReferenceExceptions are common runtime errors.
- **Solution**: Explicitly mark types as nullable using `?`.
  - `string? input = Console.ReadLine();` (Input might be null).
  - `string name;` (Must not be null).

- **Compiler Warning**: If you try to dereference a potential null without checking, the compiler warns you.

1.2.3 2.3 Asynchronous Programming (Async/Await)

- **Concept**: Allows the program to perform other work while waiting for long-running operations (I/O, Database, Network) to complete, rather than blocking the thread.
- **Analogy (Breakfast)**:
  - *Synchronous*: Pour coffee -> wait -> Fry eggs -> wait -> Fry bacon -> wait -> Toast bread. (Coffee is cold by the end).
  - *Asynchronous*: Start coffee, start eggs, start bacon, start toast. Handle them as they finish.
- **Keywords**:
  - `Task`: Represents an ongoing operation (like a "Promise" in JS).
  - `async`: Modifies a method to allow usage of `await`.
  - `await`: Pauses the method execution until the awaited Task is complete, yielding control back to the caller.
- **Usage**: `csharp public async Task<string> GetDataAsync() { await Task.Delay(1000); // Simulate work return "Data"; }`
- **Rules**:
  - Async methods return `Task`, `Task<T>`, or `void` (avoid void unless for event handlers).
  - Conventions: Append "Async" to method names (e.g., `GetManyAsync`).

1.2.4 2.4 JSON & File I/O

- **Serialization**: Converting objects to text (JSON).
  - `JsonSerializer.Serialize(obj, options)`
- **Deserialization**: Converting text (JSON) to objects.
  - `JsonSerializer.Deserialize<T>(json)`
- **Options**:
  - `WriteIndented = true` (Pretty print).
  - `PropertyNameCaseInsensitive = true` (Crucial when mapping camelCase JSON from Web APIs to PascalCase C# properties).
- **File Operations**:
  - `await File.WriteAllTextAsync("path.json", content);`
  - `await File.ReadAllTextAsync("path.json");`

1.2.5 2.5 Functional C# (LINQ Basics)

- **Lambda Expressions**: Anonymous functions. `(args) => expression`.
- **Filtering**:
  - `.Where(x => x.Age > 18)`: Returns all matching elements.
  - `.First(x => ...)`: Returns first match or throws exception.
  - `.FirstOrDefault(x => ...)`: Returns first match or null.
  - `.Single(x => ...)`: Returns the *only* match or throws if 0 or >1 exist.
- **Collection Expressions**: `List<int> list = [.. existingList.Where(...)];`

1.3 3. Web API & REST

1.3.1 3.1 HTTP & Resources

- **Web API**: An interface for interacting with the business logic over the web.
- **Resource**: Data exposed by the API, identified by a URL (e.g., `/posts`).
- **HTTP Verbs (CRUD)**:
  - **GET**: Read. (200 OK, 404 Not Found). Safe & Idempotent.

- **POST**: Create. (201 Created). Not Idempotent.
- **PUT**: Update/Replace. (204 No Content or 200 OK). Idempotent.
- **PATCH**: Partial Update.
- **DELETE**: Remove. (204 No Content or 200 OK).

1.3.2 3.2 REST Constraints

- **Uniform Interface**: Interact with all resources in a standard way.
- **Stateless**: The server does not remember client state between requests.
- **Client-Server**: Separation of concerns.

1.3.3 3.3 ASP.NET Core Implementation

- **Controllers**: Classes that handle HTTP requests.
  - Decorated with `[ApiController]` and `[Route("[controller]")]`.
- **Dependency Injection (DI)**:
  - Services (like Repositories) are injected into the controller constructor.
  - Registered in `Program.cs` (e.g., `builder.Services.AddScoped<IPostRepository, PostRepository>();`).
- **DTOs (Data Transfer Objects)**:
  - Simple classes used to transfer data between client and server.
  - Decouples the internal domain model from the external API contract.
  - Example: `CreatePostDto` (no ID, no timestamp) vs `Post` (has ID, timestamp).
- **Status Codes**:
  - `Results.Ok(data)`
  - `Results.Created("/path/to/resource", data)`
  - `Results.NotFound()`
  - `Results.BadRequest()`

1.4 4. HTTP Client

1.4.1 4.1 Consuming APIs

- **HttpClient**: The class used to send HTTP requests.
- **Best Practice**: Register `HttpClient` as a service in DI to manage socket connections efficiently. csharp builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://api.example.com") });

1.4.2 4.2 Methods

- **GET**: csharp HttpResponseMessage response = await client.GetAsync("todos"); string content = await response.Content.ReadAsStringAsync(); // Deserialize content...
- **POST**: csharp string json = JsonSerializer.Serialize(dto); StringContent content = new(json, Encoding.UTF8, "application/json"); await client.PostAsync("todos", content);

1.5 5. Blazor (Frontend)

1.5.1 5.1 Overview

- **Concept**: Build interactive web UIs using C# and .NET instead of JavaScript.
- **Hosting Models**:
  - *Blazor Server*: Runs on server, UI updates sent via SignalR socket.
  - *Blazor WebAssembly (Wasm)*: Runs in the browser via WebAssembly (Client-side).

- **Razor Syntax**: Combines HTML and C#.
  - `@code { ... }`: Block for C# logic.
  - `@variable`: Renders a C# variable.
  - `@if (...) { ... }`: Conditional rendering.
  - `@foreach (...) { ... }`: Loops.

1.5.2 5.2 Components

- **Definition**: .NET classes representing UI chunks (Pages, Dialogs, Forms).
- **Parameters**: Pass data to components using `[Parameter]`.
- **Routing**: Defined via `@page "/path"` directive.

1.5.3 5.3 Data Binding & Events

- **One-way**: `<input value="@myVar" />`
- **Two-way**: `<input @bind="myVar" />` (Updates variable when input changes).
- **Event Handling**: `<button @onclick="MyMethod">Click</button>`.

1.5.4 5.4 Dependency Injection in Blazor

- Inject services directly into Razor views: `csharp @inject IPostService PostService`

1.6 6. Authentication & Authorization

1.6.1 6.1 Concepts

- **Authentication (AuthN)**: Identifying *who* the user is (Login).
- **Authorization (AuthZ)**: Determining *what* the user is allowed to do.

1.6.2 6.2 Claims & Policies

- **Claim**: A key-value pair describing the user (e.g., `Role: Admin`, `Email: user@example.com`).
- **Identity**: A collection of claims.
- **Policy**: Rules based on claims (e.g., "Must be over 18", "Must be Admin").

1.6.3 6.3 JWT (JSON Web Token)

- **Structure**: Header.Payload.Signature (Base64 encoded).
- **Characteristics**: Self-contained, stateless, digitally signed.
- **Flow**:
  1. Client sends credentials (username/password).
  2. Server validates and generates JWT containing Claims.
  3. Server signs JWT with a secret key.
  4. Client stores JWT and sends it in the `Authorization: Bearer <token>` header for subsequent requests.
  5. Server validates the signature on every request to authorize access.

1.6.4 6.4 Implementation Notes

- **Blazor**: Update authentication state based on the presence/validity of the JWT. Use `CascadingAuthenticationState` to expose user info to UI components.
- **Web API**: Use `[Authorize]` attribute on Controllers or Actions to protect endpoints.

1.7 7. Entity Framework Core (Data Access)

1.7.1 7.1 ORM (Object Relational Mapper)

- **Purpose**: Maps C# objects to Database tables. Eliminates manual SQL writing.
- **Approaches**:
  - *Code First*: Define C# classes -> Generate DB. (Used in this course).
  - *Database First*: Existing DB -> Generate C# classes.

1.7.2 7.2 DbContext

- **Definition**: The bridge between the application and the database.
- **DbSet**: Represents a table (e.g., `public DbSet<Book> Books { get; set; }`).
- **Configuration**: `csharp protected override void OnConfiguring(DbContextOptionsBuilder options) => options.UseSqlite("Data Source=app.db");`

1.7.3 7.3 Relationships

- **1:Many**: A `Programme` has a `List<Course>`. A `Course` has a foreign key to `Programme`.
- **Many:Many**: Represented by a collection on both sides. EF Core handles the join table automatically or it can be explicit.
- **Navigation Properties**: Properties that link to other entities (e.g., `book.Reviews`).

1.7.4 7.4 Migrations

- **Workflow**:
  1. Modify Domain Model.
  2. `dotnet ef migrations add <Name>`: Creates migration script.
  3. `dotnet ef database update`: Applies script to DB.

1.7.5 7.5 CRUD Operations

- **Create**: `context.Books.Add(book); await context.SaveChangesAsync();`
- **Read**: `context.Books.FindAsync(id);`
- **Update**: Modify object properties, then `await context.SaveChangesAsync();` (Change Tracker handles it).
- **Delete**: `context.Books.Remove(obj); await context.SaveChangesAsync();`

1.7.6 7.6 Advanced Querying (LINQ to SQL)

- **Projections (`Select`)**: Convert DB entities to DTOs/ViewModels. `csharp ctx.Books.Select(b => new { b.Title, b.Price }); // Anonymous type`
- **Eager Loading (`Include`)**: Load related data. `csharp ctx.Books.Include(b => b.Author).ThenInclude(a => a.Address);`
- **Filtering**: `Where(b => b.Price < 50)`.
- **Sorting**: `OrderBy(b => b.Title).ThenByDescending(b => b.Price)`.
- **Paging**: `.Skip(10).Take(10)`.
- **Flattening**: `SelectMany` converts a list of lists into a single flat list (e.g., getting all reviews from all books).
- **Aggregation**: `Count()`, `Sum()`, `Average()`, `Max()`.
- **Execution**: Query is executed only when `ToList()`, `First()`, `Count()`, etc., are called.

1.8 8. Exam Information

1.8.1 8.1 Format

- **Type**: 4-hour written exam.
- **Content**: Practical programming exercise (similar to course assignments).
- **Environment**: Individual. Use Rider, VS Code, or Visual Studio.
- **Submission**: Zip full solution (delete `bin` and `obj` folders first) to Wiseflow.

1.8.2 8.2 Rules & Allowed Aids

- **Allowed**: Internet, course materials, past assignments.
- **Forbidden**:
  - Communication with others.
  - **AI Tools**: No ChatGPT, Copilot, "Agentic AI", or "Intellicode". **Disable these plugins** to avoid accidental disqualification during screen recording.
  - Public GitHub (accessing your own private repo is fine if static, but no active pushing/sharing).

1.8.3 8.3 Tips for Passing

- **Theory is not enough**: You must be able to write code.
- **Focus Areas**:
  - Web Applications (Blazor).
  - RESTful Web Services (Web API).
  - Data Access (EF Core).
- **Strategy**:
  - Don't invent new features not asked for.
  - If stuck, move on. Partial credit is given for "almost working" code.
  - Follow diagrams if provided.
  - Practice with previous exam sets and time yourself.