ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

DEVELOPER GUIDE

MÔN HỌC: **Thiết kế phần mềm**

Học kỳ II (2024 – 2025)

LỚP: **22_3**

MSSV	HỌ TÊN
22120325	NGUYỄN NHẬT TÂN
22120328	TRẦN NHẬT TÂN
22120337	LƯƠNG THỊ DIỆU THẢO

TP. HỒ CHÍ MINH, THÁNG 6 NĂM 2025

Contents

I. Coding Standards

- This project follows standard C# conventions:
  - PascalCase for class, method, and property names
  - camelCase for private fields and local variables
  - Use of async/await for asynchronous methods
  - Separate concerns: controller, service, repository logic
  - Organize using statements (sort, remove unused)
  - Use meaningful variable, method, and class names
  - Keep methods short and focused on a single responsibility
- React code follows standard JS/TS practices:
  - Functional components with hooks
  - Modular CSS and component-based structure
  - Linting with ESLint and Prettier support (if configured)
  - Always handle API errors gracefully and display meaningful messages to users

II. Overview of Architecture

- The project is organized into a clean 3-layered architecture:
  - Controller Layer: Accepts HTTP requests, invokes services
  - Service Layer: Business logic, data manipulation, validation
  - Data Access Layer: Entity Framework context and DB access
- Frontend (React) consumes REST API and handles user interaction.

III. Source Code Organization

Group23-Ex-Sat/

```
├── Backend/
│   ├── Controllers/          // API endpoints
│   ├── Services/             // Business logic
│   ├── Models/               // EF Core models
│   ├── DTOs/                 // Transfer objects
│   ├── Validators/           // Custom input validators
│   ├── CsvMapping/           // Import/export helpers
│   ├── Migrations/           // EF migrations
│   ├── Resources/            // Localization Values
```

```
│   ├── logs/                    // Log files
│   └── appsettings.json         // Configurations
├── Frontend/
│   └── src/                     // React components
│   ├── components/              // Reusable UI Components
│   ├── pages/                   // Page Components / Views
│   ├── utils/                   // Helpers / Formatters
│   ├── locales/                 // Localization Values
│   ├── contexts/                // React Context Providers
│   ├── App.js
│   ├── index.js
├── Report/                      // Documentation
├── example.csv/.json            // Sample data
└── .env                         // Environment settings
```

IV. Getting Started with App Development

- Tools Required for Development:
  - .NET 9 runtime & .NET 9 SDK
  - Visual Studio 2022 or Above / Visual Studio Code
  - Microsoft SQL Server
  - NodeJs
  - Npm
  - Git
- In Backend/appsettings.json, configure DefaultConnection to your SQL Server
- On Window, run this as admin:

```
Dism /online /Enable-Feature /FeatureName:"NetFx3"
```

- Backend:
  - Open cmd in Backend folder
  - dotnet restore
  - dotnet tool install --global dotnet-ef
  - dotnet ef database update
  - dotnet run

- Frontend:
    - Open cmd in Frontend folder
    - npm install
    - npm start
- Visit: http://localhost:3000

V. Database Schema

- Uses Entity Framework Core for ORM:
    - Student, Department, .etc entities
    - Each model maps to a table
    - Use data annotations ([Key], [Required], [MaxLength], etc.)
    - Migrations track schema evolution
- To add migration:
    - dotnet ef migrations add <Name>
    - dotnet ef database update

VI. Updating an Existing Entity

- To add a new property:
    - Add the property in Models/<Entity>.cs
    - Update DTOs/<Entity>.cs to match
    - If used in validation, update Validators/
    - Generate a new migration:
        - dotnet ef migrations add AddNewPropertyTo<Entity>
        - dotnet ef database update

VII. Registering New Routes

- Routes are registered via ASP.NET Core attribute routing:

```
[Route("api/[controller]")]

[ApiController]

public class StudentController : ControllerBase { }
```

- To register new routes, add methods to controllers using [HttpGet], [HttpPost], etc.

VIII. Inversion of Control & Dependency Injection

- ASP.NET Core built-in DI is used:
    - Services are registered in Program.cs:

```
services.AddScoped<IStudentService, StudentService>();
```

- Inject into controllers:

```
public StudentController(IStudentService service) {
   _service = service;
}
```

IX. Data Validation

- Validation is handled via:
  - **Data Annotations** for basic validations inside DTOs:

```
[Required]
[StringLength(100)]
public string FullName { get; set; }
```

  - Custom logic in Validators/
  - Run during import or POST/PUT calls
  - Examples include email domain, phone format, status

X. Settings API

- App configuration is stored in appsettings.json and .env for secrets like:
  - Email domain rules
  - Phone number formats
- Use IConfiguration to access:

```
var domain = _config["ValidEmailSuffix"];
```

XI. Unit Testing

This project use **xUnit** framework for testing Backend:

- Tests are located under Backend.Tests/.
- Use mocking libraries like Moq to mock dependencies.

```
[Fact]
public async Task GetStudentById_ExistingStudent_ShouldReturnStudent ()
{
   var studentId = "22100142";
   var student = new Student { StudentId = studentId, FullName = "John Doe"
};

   _studentRepositoryMock.Setup(x => x. GetStudentById(studentId)).
ReturnsAsync(student);
```

```
        var result = await _studentService. GetStudentById (studentId);

        Assert.Equal(student, result);
    }
```

- Run tests: dotnet test

XII. Web API Documentation

- Use Swagger for API documentation.
- Swagger is enabled in Program.cs:

```
builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

    c.SwaggerDoc("v1", new OpenApiInfo { Title = "StudentManagement API",
Version = "v1" });

});
```

- Access API docs at: https://localhost:<port>/swagger/index.html