

КОМЕНТАРИ И ДОКУМЕНТАЦИЯ

В ASP.NET CORE MVC (C#)

Пълен комплект учебни материали за 12 учебни часа (12 клас, Модул 3)

Тема: Четим код, смислени коментари, XML документация, README и архитектурни бележки

Технологии: .NET 8, ASP.NET Core MVC, EF Core, SQLite

Дата: 13.01.2026

1. Обхват и цели на темата

Темата запознава учениците с добри практики за писане на коментари и документация в реален уеб проект. Фокусът е върху ASP.NET Core MVC, но принципите са универсални: документацията трябва да обяснява "зашо" и договорите на публичния код, а не да повтаря очевидното от самия код.

Очаквани резултати

- различават коментари, XML документация, README и архитектурна документация
- пишат кратки и точни коментари само там, където добавят стойност (правила, ограничения, edge cases)
- документират публични методи със XML documentation (///) и поддържат съответствие с кода
- структурират MVC проект на слоеве (Controllers -> Services -> Data) и описват архитектурата
- подготвят README с ясни стъпки за стартиране и примери за употреба
- извършват code review по чеклист и прилагат корекции

Предварителни знания и среда

- C# основи: класове, методи, изключения, асинхронност (Task/async/await)
- основи на MVC: Controller, View, Model
- инструменти: Visual Studio 2022 или VS Code, .NET SDK 8

2. Тематичен план по часове (12 учебни часа)

Час	Тема	Теория (ключови точки)	Практика / резултат
1	Четим код vs коментари	Разлика: коментари, документация, README; правило "кодът показва как, док-цията казва защо".	Рефакторинг на лоши имена без коментари; кратко обсъждане.
2	Видове коментари и стил	//, /* */; TODO/NOTE; кога коментарът вреди.	Класифициране на коментари (полезен/вреден); поправка.
3	Анти-патерни и рефакторинг	Закоментиран код; неверни коментари; магически стойности.	Почистване на анти-примери; замяна с по-добър код/имена.
4	XML Documentation (///) - основи	<summary>, <param>, <returns>, <remarks>, <exception>; договор на метод.	Документиране на 5 service метода; проверка чрез IntelliSense.
5	Документация на Controllers	Какво описваме: поведение, NotFound, валидация; <remarks> за правила.	Документиране на 3 action-a: Details, Borrow, Return.
6	Генериране на документация	XML doc файл в проекта; (по избор) Swagger за малък API контролер.	Включване на GenerateDocumentationFile; добавяне на API + Swagger (по желание).
7	README за ASP.NET проект	Структура: Overview, Setup, Run, Config, Usage, Limitations, Future.	Създаване на README по шаблон; peer-check (друг ученик стартира проекта).
8	Архитектурна бележка (1 стр.)	Слоеве и отговорности; правила за зависимости; структура на папки.	Файл docs/architecture.md с диаграма и 3 бизнес правила.
9	Валидация и edge cases	DataAnnotations; документиране на грешки; кога да хвърляме exception.	Добавяне на валидации и съответни XML docs; показване на UI errors.
10	Code review работилница	Чеклист; конкретна обратна връзка; синхрон между код и документация.	Размяна на код; Review Report (мин. 5 препоръки) + корекции.
11	Мини-проект - реализация	Работен час: функционалност + документация.	Довършване: Borrow/Return в service; XML docs; README + архитектура.
12	Зашита и демонстрация	Как се представя документацията; критерии за оценяване.	Демо (2 сценария) + показване на IntelliSense + README + архитектура.

Забележка: Час 6 е с опция за Swagger (API) само като демонстрация на "жива" документация. Основната част остава MVC с Views и Services.

3. Лабораторни упражнения (листове за ученици)

Упражнение 1 (Час 1): Рефакторинг за четимост

Цел: Да се подобри четимостта без добавяне на излишни коментари.

1. Преименувай методи и променливи така, че да описват ясно отговорността.
2. Раздели дълги методи на по-малки (Extract Method), ако е нужно.
3. Премахни коментари, които повтарят кода (например "return view").

Резултат: commit с ясно описание (какво и защо е подобрено).

Упражнение 2 (Час 2): Полезни vs вредни коментари

Цел: Да се разпознават анти-коментари и да се заменят с по-добър код.

Задачи:

4. Маркирай 10 коментара като полезни или вредни.
5. Поправи поне 5: или ги премахни, или ги преписши така, че да обясняват "защо".
6. Където е възможно - замени коментар с по-добро име на метод/променлива.

Упражнение 3 (Час 3): Анти-патерни

Цел: Да се премахне "закоментиран код" и да се изчисти логиката.

Критерии:

- Няма закоментирани блокове с старяла логика.
- TODO/NOTE са конкретни: какво липства и защо е важно.
- Бизнес логиката не е в Controller, а в Service.

Упражнение 4-5 (Час 4-5): XML docs за услуги и контролери

Цел: Документиране на публични методи с XML documentation, така че IntelliSense да е полезен.

7. Добави <summary>, <param>, <returns> на 5 service метода.
8. Добави <remarks> за бизнес правило и ограничения.
9. Документирай 3 controller actions (Details, Borrow, Return).

Упражнение 6 (Час 6): Генериране на документация

Цел: Проектът да генерира XML doc файл; по желание - Swagger за API контролер.

Стъпки:

10. Включи GenerateDocumentationFile в .csproj.
11. Провери дали се генерира XML файл при билд.
12. По желание: добави малък API контролер и включи Swagger с IncludeXmlComments.

4. Мини-проект: SchoolLibrary (ASP.NET Core MVC)

Функционални изисквания

- Books: List, Create, Edit, Details (CRUD без Delete е допустимо по избор).
- Borrow book: възможно само ако книгата не е заета (IsBorrowed == false).
- Return book: възможно само ако книгата е заета.
- Валидация: Title и StudentName са задължителни; Year да е в разумен диапазон (по избор).

Документационни изисквания

- XML docs за всички public service методи (IBookService/IBorrowService).
- XML docs за поне 3 controller actions.
- README.md с точни стъпки за стартиране + примерна употреба.
- docs/architecture.md (1 страница): слоеве, структура и ключови правила.
- Коментари: само когато обясняват "защо" или tricky логика; без дублиране на кода.

Предложена структура на проекта

```
SchoolLibrary/
  Controllers/
    BooksController.cs
    BorrowController.cs
    Api/BooksApiController.cs    (по желание)
  Services/
    IBookService.cs
    BookService.cs
    IBorrowService.cs
    BorrowService.cs
  Data/
    ApplicationDbContext.cs
  Models/
    Book.cs
    BorrowRecord.cs
  ViewModels/
    BookCreateVm.cs
    BorrowVm.cs
  Views/
  docs/
    architecture.md
  README.md
```

Модели (пример)

```
public class Book
{
    public int Id { get; set; }
    public string Title { get; set; } = "";
    public string Author { get; set; } = "";
    public int Year { get; set; }
    public bool IsBorrowed { get; set; }
}

public class BorrowRecord
```

```
{  
    public int Id { get; set; }  
    public int BookId { get; set; }  
    public Book Book { get; set; } = null!;  
    public string StudentName { get; set; } = "";  
    public DateTime BorrowedOn { get; set; }  
    public DateTime? ReturnedOn { get; set; }  
}
```

5. Код примери: добри и лоши практики

Лош пример: коментари, които повтарят кода

```
// returns the view  
return View(model);
```

Поправка: премахни коментара. Ако има нужда от обяснение, то обикновено е знак за лошо име или липсваща абстракция.

Лош пример: бизнес логика в Controller

```
public async Task<IActionResult> Borrow(int id, string student)  
{  
    // borrow book  
    var book = await _db.Books.FindAsync(id);  
    if (book == null) return RedirectToAction("Index");  
  
    // if borrowed do nothing  
    if (book.IsBorrowed) return RedirectToAction("Index");  
  
    book.IsBorrowed = true;  
    _db.SaveChanges();  
    return RedirectToAction("Index");  
}
```

Очаквана поправка: изнеси правилото и записа в BorrowService; Controller да валидира входа и да върне подходящ резултат.

Добър пример: XML docs в service слой

```
/// <summary>  
/// Attempts to borrow a book for a student.  
/// </summary>  
/// <param name="bookId">Book identifier.</param>  
/// <param name="studentName">Student full name.</param>  
/// <returns>  
/// True when the book was successfully borrowed; false when the book is already borrowed or  
/// missing.  
/// </returns>  
/// <remarks>  
/// Rule: a book can be borrowed only if IsBorrowed is false.  
/// </remarks>  
public async Task<bool> BorrowAsync(int bookId, string studentName)  
{  
    if (string.IsNullOrWhiteSpace(studentName)) return false;  
  
    var book = await _db.Books.FindAsync(bookId);  
    if (book is null || book.IsBorrowed) return false;  
  
    book.IsBorrowed = true;  
    _db.BorrowRecords.Add(new BorrowRecord  
    {  
        BookId = bookId,  
        StudentName = studentName.Trim(),  
        BorrowedOn = DateTime.UtcNow
```

```
});  
  
await _db.SaveChangesAsync();  
return true;  
}
```

NOTE коментар за tricky решение (пример)

```
// NOTE: Using UTC timestamps to avoid issues when the server timezone changes (DST).  
record.BorrowedOn = DateTime.UtcNow;
```

6. Шаблони за документация

README.md шаблон

```
# SchoolLibrary

## Overview
Web application for managing books and borrowing records in a school library.

## Features
- Manage books (Create/Edit/List/Details)
- Borrow and return books with basic validation

## Tech Stack
- ASP.NET Core MVC (.NET 8)
- EF Core + SQLite

## Setup & Run
### Requirements
- .NET SDK 8

### Steps
1. Restore packages:
   - dotnet restore
2. Create database:
   - dotnet ef database update
3. Run:
   - dotnet run
4. Open the app:
   - http://localhost:xxxx

## Configuration
- Connection string is in appsettings.json

## Example Usage
1. Add a book from Books -> Create
2. Borrow a book from Borrow -> Borrow
3. Return a book from Borrow -> Return

## Known Issues / Limitations
- No authentication (demo project)

## Future Improvements
- Add user accounts (students/librarians)
- Add search and paging
```

docs/architecture.md шаблон

```
# Architecture

## Layers
- Controllers: validate input and coordinate requests
- Services: business logic (borrow/return rules)
- Data: EF Core DbContext and entities
- Views: UI
```

```
## Folder Structure
- /Controllers
- /Services
- /Data
- /Models
- /ViewModels
- /docs

## Key Business Rules
- A book can be borrowed only when IsBorrowed == false
- Returning a book sets ReturnedOn and IsBorrowed = false
- StudentName is required for borrowing
```

Code Review Report шаблон

```
# Code Review Report

## Reviewed by:
- Name:

## What is good (3 bullets)
- ...
- ...
- ...

## Issues (min 5) with file + suggestion
1. File: ... | Problem: ... | Suggested fix: ...
2. ...
3. ...
4. ...
5. ...

## Documentation mismatches (if any)
- ...
```

7. Чеклисти и рубрика за оценяване

Чеклист: Коментари

- Няма коментари, които повтарят очевиден код.
- Има коментари само за: бизнес правила, ограничения, edge cases, tricky решения.
- Няма закоментиран код (старите варианти се пазят в Git, не във файла).
- TODO/NOTE са конкретни и полезни.

Чеклист: XML docs

- Има <summary> за public methods и интерфейси.
- <param> описва значението на параметъра (не повтаря името).
- <returns> описва какво означава резултатът (успех/неуспех).
- <remarks> съдържа бизнес правило или важно ограничение.
- Документацията съвпада с текущото поведение на кода.

Чеклист: README + архитектура

- README позволява проектът да се стартира без допълнителни въпроси.
- Има requirements, setup, run, configuration, usage.
- Architecture документът е 1 страница и описва слоевете и правилата.
- Има limitations и future improvements (реалистични).

Рубрика (100 точки)

Критерий	Точки	Бележки
Функционалност (CRUD + Borrow/Return)	40	Коректни сценарии + валидация
Качество на кода (слоеве, имена, кратки методи)	20	Бизнес логика в services
XML документация	20	Полезни summary/param/returns/remarks
README + docs/architecture.md	15	Стартиране по стъпки + яснота
Code review участие	5	Попълнен report + приложени корекции