CS6004NT Application Development

WEEK - 04

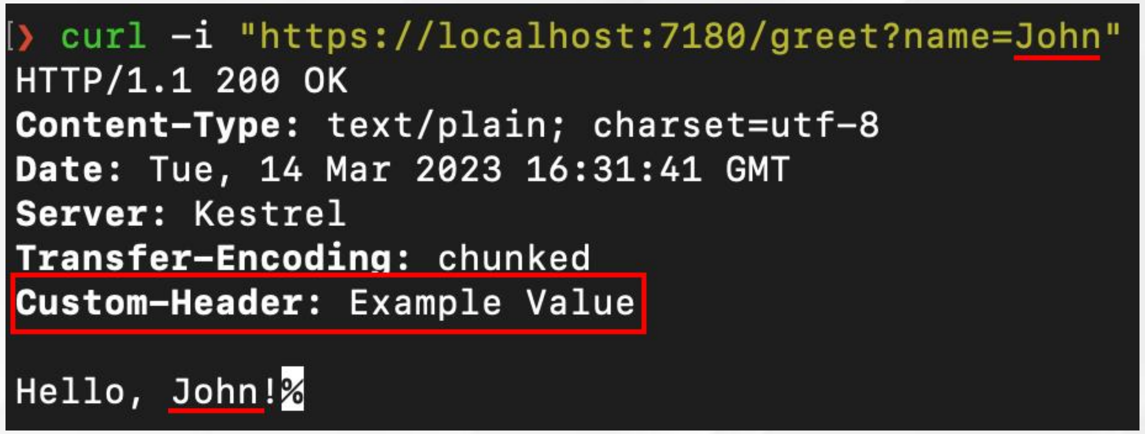
Web API

HttpContext

HttpContext represents the current HTTP request and response context in a web application. It provides access to information such as the HTTP headers, request method, URL, user identity, and response status.

Example:

```
[HttpGet("/greet")]
public string GetGreeting()
{
    HttpContext.Response.Headers.Add("Custom-Header", "Example Value");
    var name = HttpContext.Request.Query["name"];
    var greeting = $"Hello, {name}!";
    return greeting;
}
```



```
> curl -i "https://localhost:7180/greet?name=John"
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Date: Tue, 14 Mar 2023 16:31:41 GMT
Server: Kestrel
Transfer-Encoding: chunked
Custom-Header: Example Value

Hello, John!%
```

Middleware

Middleware is software that intercepts and processes requests and responses in a software system. In the context of web development, middleware typically sits between the client and server, allowing you to add functionality to the request/response pipeline.

Example:

```
var app = builder.Build();
app.Use(async (context, next) =>
{
    var stopwatch = new Stopwatch();
    var httpMethod = context.Request.Method;
    var urlPath = context.Request.Path;
    stopwatch.Start();
    await next(context); // execute the next delegate/middleware in the pipeline
    stopwatch.Stop();
    var ms = stopwatch.ElapsedMilliseconds;
    Console.WriteLine($" {httpMethod} : {urlPath} : {ms} ms"); // GET: /books : 33 ms
});
app.UseHttpsRedirection();
```

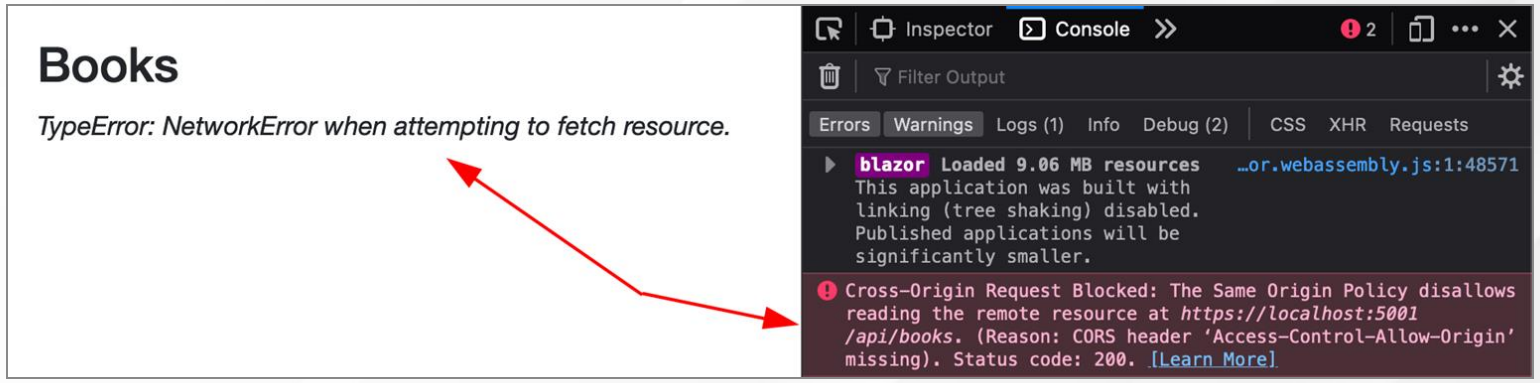
Middleware

CORS Middleware

Program.cs:

```
var app = builder.Build();

app.UseCors(policy =>
    policy.WithOrigins("http://localhost:3000", "https://localhost:3001")
     .AllowAnyMethod()
     .WithHeaders(HeaderNames.ContentType)
);
```



Books

TypeError: NetworkError when attempting to fetch resource.

```
blazor Loaded 9.06 MB resources        …or.webassembly.js:1:48571
This application was built with
linking (tree shaking) disabled.
Published applications will be
significantly smaller.
Cross-Origin Request Blocked: The Same Origin Policy disallows
reading the remote resource at https://localhost:5001
/api/books. (Reason: CORS header 'Access-Control-Allow-Origin'
missing). Status code: 200. [Learn More]
```

Middleware

Error Handling Middleware

ErrorHandlingMiddleware.cs:

```
using System.Net;
using System.Text.Json;
using BookReview.Api.Domain.Exceptions;

namespace BookReview.Api.Infrastructure.Middlewares;
public class ErrorHandlingMiddleware
{
    private readonly RequestDelegate _next;
    private readonly ILogger<ErrorHandlingMiddleware> _logger;

    public ErrorHandlingMiddleware(RequestDelegate next, ILogger<ErrorHandlingMiddleware> logger)
    {
        _next = next;
        _logger = logger;
    }
    public async Task InvokeAsync(HttpContext context)
    {
        try
        {
            await _next(context);
        }
        catch (DomainException ex)
        {
            _logger.LogError(ex, "An domain exception occurred.");
            await HandleExceptionAsync(context, ex.StatusCode, ex.Message);
        }
        catch (Exception ex)
        {
            var message = "An error occurred on the server.";
            _logger.LogError(ex, "An unhandled exception occurred.");
            await HandleExceptionAsync(context, HttpStatusCode.InternalServerError, message);
        }
    }

    private async Task HandleExceptionAsync(HttpContext context, HttpStatusCode statusCode, string message)
    {
        context.Response.StatusCode = (int)statusCode;
        context.Response.ContentType = "application/json";
        await context.Response.WriteAsync(JsonSerializer.Serialize(new { message }));
    }
}
```

Middleware

Error Handling Middleware

DomainException.cs:

```
using System.Net;

namespace BookReview.Api.Domain.Exceptions;
public class DomainException : Exception
{
    public HttpStatusCode StatusCode { get; set; }

    public DomainException(string message, HttpStatusCode statusCode = HttpStatusCode.BadRequest)
        : base(message)
    {
        StatusCode = statusCode;
    }
}
```

Program.cs:

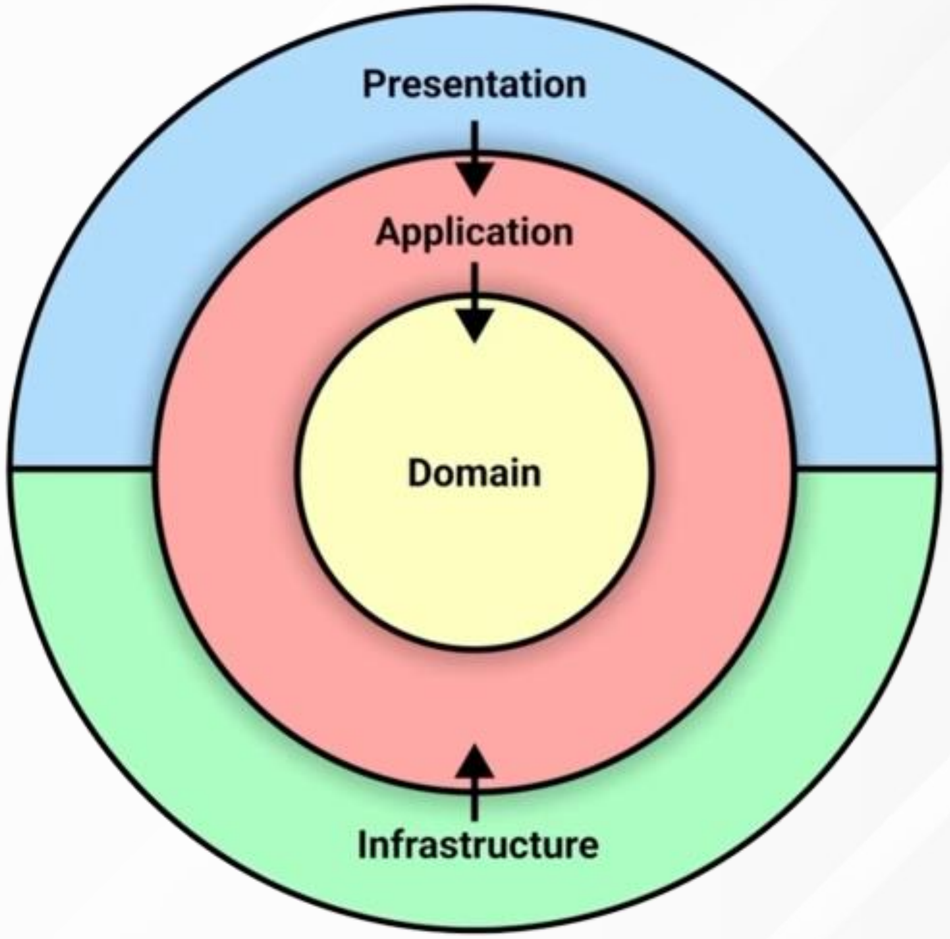
```
...
app.UseMiddleware<ErrorHandlingMiddleware>();
app.Run();
```

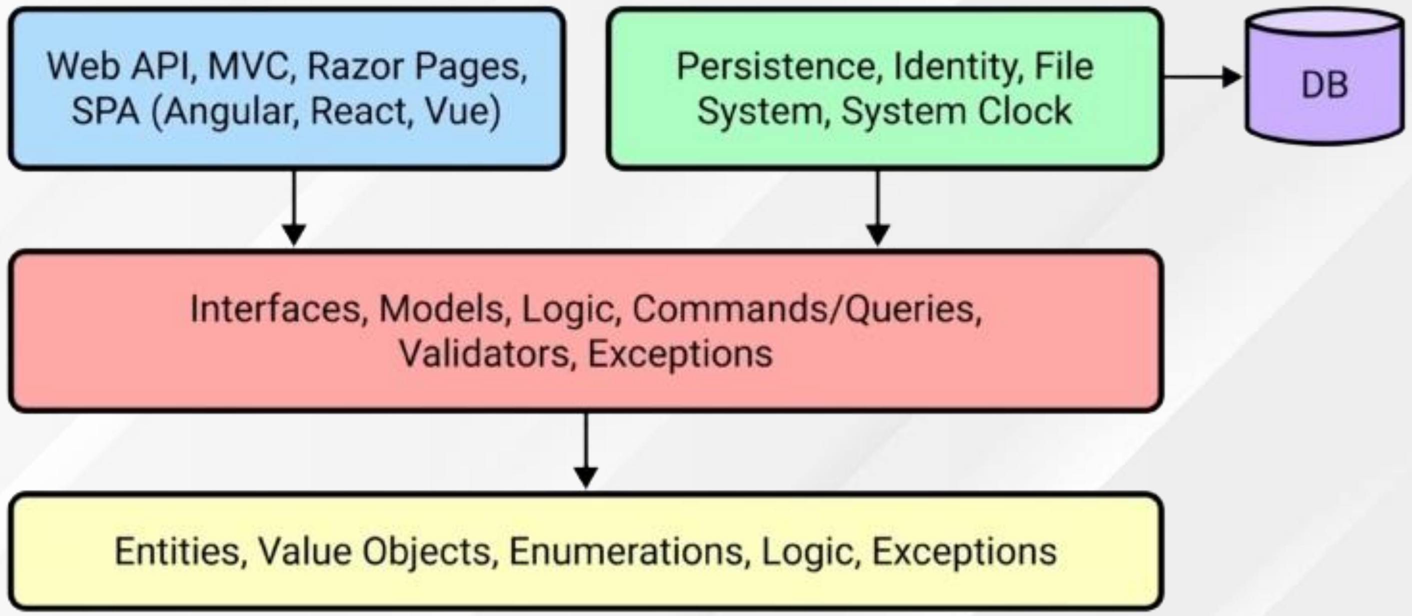
Project Structure / Architecture

```
.
├── BookReview.Api
|   ├── Domain
|   ├── Services
|   ├── Infrastructures
|   ├── Controllers
|   ├── …
…
```

```
…
├── BookReview.BlazorWasm
|   ├── Pages
|   ├── Shared
|   ├── Services
|   ├── …
├── BookReview.Contracts
|   ├── *Request.cs
|   ├── *Response.cs
|   ├── …
└── BookStore.sln
```

Clean Architecture




Web API, MVC, Razor Pages, SPA (Angular, React, Vue)
Persistence, Identity, File System, System Clock
DB
Interfaces, Models, Logic, Commands/Queries, Validators, Exceptions
Entities, Value Objects, Enumerations, Logic, Exceptions

Project Setup

```
# Create BookReview solution

dotnet new sln -n BookReview

# Create projects

dotnet new webapi -n BookReview.Api --framework net6.0

dotnet new blazorwasm -o BookReview.BlazorWasm --framework net6.0

dotnet new classlib -n BookReview.Contracts --framework net6.0

# Add projects to the BookReview solution

dotnet sln add BookReview.Api/BookReview.Api.csproj

dotnet sln add BookReview.BlazorWasm/BookReview.BlazorWasm.csproj

dotnet sln add BookReview.Contracts/BookReview.Contracts.csproj

# Add reference of Contracts project to Api and BlazorWasm projects

dotnet add BookReview.Api/BookReview.Api.csproj reference BookReview.Contracts/BookReview.Contracts.csproj

dotnet add BookReview.BlazorWasm/BookReview.BlazorWasm.csproj reference BookReview.Contracts/BookReview.Contracts.csproj
```

Project Setup

Book Service

```
namespace BookReview.Api.Services;
public class BookService : IBookService
{
    private List<Book> _books = new List<Book>
    {
        new Book
        { Id = 1, Title = "Animal Farm", Author = "George Orwell", Year = 1945 },
        ...
        new Book
        { Id = 4, Title = "The Book Thief", Author = "Markus Zusak", Year = 2005 }
    };

    public IEnumerable<Book> GetAllBooks()
    {
        return _books;
    }
...
```

Program.cs

```
...
builder.Services.AddSingleton<IBookService,
BookService>();
var app = builder.Build();
```

Project Setup

Book Controller

```
namespace BookReview.Api.Controllers;

[ApiController]
public class BookController : ControllerBase
{
    private readonly ILogger<BookController> _logger;
    private readonly IBookService _bookService;
    public BookController(ILogger<BookController> logger, IBookService bookService)
    {
        _logger = logger;
        _bookService = bookService;
    }

    [HttpGet("/api/books")]
    public ActionResult<IEnumerable<BookResponse>> GetAllBooks()
    ...
```

Project Setup

Blazor Book Service

```
namespace BookReview.BlazorWasm.Services;
public class BookService : IBookService
{
    private readonly HttpClient _httpClient;
    private readonly ILogger<BookService> _logger;

    public BookService(HttpClient httpClient, ILogger<BookService> logger)
    {
        _httpClient = httpClient;
        _logger = logger;
    }

    private async Task CheckForErrorResponse(HttpResponseMessage response)
    {
        if (!response.IsSuccessStatusCode)
        {
            var errorResponse = await response.Content.ReadFromJsonAsync<ErrorMessageResponse>();
            _logger.LogError($"Http status code: {response.StatusCode} message: {errorResponse?.Message}");
            throw new Exception(errorResponse?.Message);
        }
    }
```

```csharp
public async Task<IEnumerable<BookResponse>> GetBooks()
{
    var response = await _httpClient.GetAsync("/api/books");

    await CheckForErrorResponse(response);

    var result = await response.Content.ReadFromJsonAsync<IEnumerable<BookResponse>>();
    return result ?? Enumerable.Empty<BookResponse>();
}

public async Task<BookResponse> CreateBook(BookRequest book)
{
    var response = await _httpClient.PostAsJsonAsync<BookRequest>("/api/books", book);

    await CheckForErrorResponse(response);

    var result = await response.Content.ReadFromJsonAsync<BookResponse>();
    return result!;
}
...
```

Project Setup

Blazor DI

Program.cs

```
builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:5001") });
builder.Services.AddScoped<IBookService, BookService>();
await builder.Build().RunAsync();
```

BooksBase.cs

```
namespace BookReview.BlazorWasm.Pages;
public class BooksBase : ComponentBase
{
    [Inject]
    public IBookService BookService { get; set; }
    ...
    protected override async Task OnInitializedAsync()
    {
        try
        {
            Books =  await BookService.GetBooks();
        }
        ...
```

Books.razor

```
@page "/books"
@inherits BooksBase
...
            @foreach (var book in Books)
            {
                <tr>
                    <td>@book.Title</td>
                    ...
```

CRUD

Create (POST)

1. Controller Action Method

```
[HttpPost("/api/books")]
public ActionResult<BookResponse> CreateBook([FromBody] BookRequest book)
{
    var newBook = new Book {
        Title = book.Title,
        Author = book.Author,
        Year = book.Year
    };
    var result = _bookService.CreateBook(newBook);
    var response = new BookResponse {
        Id = result.Id,
        Title = result.Title,
        Author = result.Author,
        Year = result.Year
    };

    return Ok(response);
}
```

CRUD

Create (POST)

2. Service Method

```
public Book CreateBook(Book book)
{
    if (_books.Any(x => x.Title.Equals(book.Title)))
    {
        throw new DomainException($"Book titled '{book.Title}' already exists.");
    }

    book.Id = _books.OrderBy(x => x.Id).Last().Id + 1;
    _books.Add(book);

    return book;
}
```

CRUD

Read (GET)

1. Controller Action Method

```
[HttpGet("/api/books/{id:int}")]
public ActionResult<BookResponse> GetBookById(int id)
{
    var book = _bookService.GetBookById(id);
    var response = new BookResponse {
        Id = book.Id,
        Title = book.Title,
        Author = book.Author,
        Year = book.Year,
    };

    return Ok(response);
}
```

CRUD

Read (GET)

2. Service Method

```
public Book GetBookById(int id)
{
    var book = _books.FirstOrDefault(x => x.Id == id);
    if (book is null)
    {
        throw new DomainException($"Book id '{id}' does not exists.", HttpStatusCode.NotFound);
    }

    return book;
}
```

CRUD

Update (PUT)

1. Controller Action Method

```
[HttpPut("/api/books/{id:int}")]
public IActionResult UpdateBook(int id, BookRequest book)
{
    var updateBook = new Book {
        Title = book.Title,
        Author = book.Author,
        Year = book.Year
    };

    _bookService.UpdateBook(id, updateBook);

    return NoContent();
}
```

CRUD

Update (PUT)

2. Service Method

```
public void UpdateBook(int id, Book book)
{
    var bookIndex = _books.FindIndex(x => x.Id == id);

    if (bookIndex < 0)
    {
        throw new DomainException(
            $"Book titled '{book.Title}' does not exists.",
            HttpStatusCode.NotFound
        );
    }

    _books[bookIndex] = book;
}
```

CRUD

Delete (DELETE)

1. Controller Action Method

```
[HttpDelete("/api/books/{id:int}")]
public IActionResult DeleteItem(int id)
{
    _bookService.DeleteBook(id);

    return NoContent();
}
```

CRUD

Delete (DELETE)

2. Service Method

```
public void DeleteBook(int id)
{
    var bookIndex = _books.FindIndex(x => x.Id == id);

    if (bookIndex < 0)
    {
        throw new DomainException($"Book id '{id}' does not exists.", HttpStatusCode.NotFound);
    }

    _books.RemoveAt(bookIndex);
}
```

EF Core

Entity Framework Core

EF Core

1. **Database First**: A database already exists, so you build a model that matches its structure and features.
2. **Code First**: No database exists, so you build a model and then use EF Core to create a database that matches its structure and features.

Installing NuGet packages for EF Core

```
dotnet add package Microsoft.EntityFrameworkCore.Tools
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
# or
dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL
```

EF Core

Database First

Scaffolding:

```
dotnet ef dbcontext scaffold \
"Server=(localdb)\\mssqllocaldb;Database=YourDatabaseName;Trusted_Connection=True;" \
Microsoft.EntityFrameworkCore.SqlServer \
--output-dir Models \
--context-dir Data \
--context BookReviewDbContext \
--table Books \
--table Reviews
```

Entity classes with annotations for Index, Primary Key, Forgion Key relationships are generated and a DbContext-derived class with DbSet for mentioned tables are generated.

EF Core

Code First

BookReviewDbContext.cs:

```
using BookReview.Api.Domain.Models;
using Microsoft.EntityFrameworkCore;

namespace BookReview.Api.Infrastructure;
public class BookReviewDbContext : DbContext
{
    public BookReviewDbContext(DbContextOptions<BookReviewDbContext> options) : base(options) { }

    public DbSet<Book> Books { get; set; }
}
```

EF Core

Code First

appsettings.json:

```json
{
  "ConnectionStrings": {
    "BookReviewDbContext": "Server=localhost;Port=5432;Database=book-review;User Id=himalay;Password=XpXuHKw5AszQ;"
},
  ...
}
```

Program.cs:

```csharp
using Microsoft.EntityFrameworkCore;
using BookReview.Api.Infrastructure;

...
builder.Services.AddDbContext<BookReviewDbContext>(
    options => options.UseNpgsql(builder.Configuration.GetConnectionString("BookReviewDbContext"))
);

builder.Services.AddScoped<IBookService, BookService>();
```

EF Core

Code First

BookRepository.cs:

```
namespace BookReview.Api.Infrastructure.Data.Data;
public class BookRepository : IBookRepository
{
    private readonly BookReviewDbContext _dbContext;

    public BookRepository(BookReviewDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public async Task<List<Book>> GetAllAsync()
    {
        return await _dbContext.Books.ToListAsync();
    }
```

```csharp
    public async Task<Book?> GetByIdAsync(int id)
    {
        return await _dbContext.Books.FindAsync(id);
    }

    public async Task<bool> AnyAsync(Expression<Func<Book, bool>>? predicate = null)
    {
        return predicate == null
            ? await _dbContext.Books.AnyAsync()
            : await _dbContext.Books.AnyAsync(predicate);
    }

    public async Task AddAsync(Book book)
    {
        await _dbContext.Books.AddAsync(book);
        await _dbContext.SaveChangesAsync();
    }

    public async Task UpdateAsync(Book book)
    {
        _dbContext.Books.Update(book);
        await _dbContext.SaveChangesAsync();
    }

    public async Task DeleteAsync(Book book)
    {
        _dbContext.Books.Remove(book);
        await _dbContext.SaveChangesAsync();
    }
}
```

Program.cs:

```csharp
builder.Services.AddScoped<IBookRepository, BookRepository>();
builder.Services.AddScoped<IBookService, BookService>();
```

EF Core

Code First

BookService.cs:

```csharp
namespace BookReview.Api.Services;
public class BookService : IBookService
{
    private readonly IBookRepository _bookRepository;

    public BookService(IBookRepository bookRepository)
    {
        _bookRepository = bookRepository;
    }

    public async Task<IEnumerable<Book>> GetAllBooksAsync()
    {
        return await _bookRepository.GetAllAsync();
    }
    ...
```

Thank You