JSON and the REST ASP.NET Core

Rasmus Lystrøm
Associate Professor
ITU
rnie@itu.dk

File Edit Selection View Go Debug Terminal Help ProgramTests.cs - HelloWorld - Visual Studio Code

ProgramTests.cs

HelloWorld.Tests > ProgramTests.cs > {} HelloWorld.Tests > HelloW

```
using System;
using System.IO;
using Xunit;

namespace HelloWorld.Tests
{
    0 references | Run All Tests | Debug All Tests
    public class ProgramTests
    {
        [Fact]
        0 references | Run Test | Debug Test
        public void Main_given_no_args_p
        {
            // Arrange
            using var writer = new Strin
            Console.SetOut(writer);

            // Act
            Program.Main(new string[0]);

            // Assert
            var output = writer.GetStrin

            Assert.Equal("Hello World!",
        }
    }
}
```

Program.cs

HelloWorld > Program.cs > ...

```
using System;

namespace HelloWorld
{
    1 reference
    public class Program
    {
        1 reference
        public static void Main(string[]
        {
            Console.WriteLine("Hello Wor
        }
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: pwsh

```
Loading personal and system profiles took 1008ms.
C:\HelloWorld> dotnet test
Test run for C:\HelloWorld\HelloWorld.Tests\bin\Debug\netcoreapp3.0\HelloWorld.Tests.dll(.NETCoreApp,Vers
ion=v3.0)
Microsoft (R) Test Execution Command Line Tool Version 16.3.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...

A total of 1 test files matched the specified pattern.

Test Run Successful.
Total tests: 1
     Passed: 1
 Total time: 3,4618 Seconds
C:\HelloWorld>
```

1 tests 0 0 Azure: rasmusl@microsoft.com HelloWorld.sln Ln 22, Col 50 Spaces: 4 UTF-8 CRLF C# 1

Agenda

Leftover: Async ≠ Parallel ≠ Threads

XML (History lesson)

JSON

REST

ASP.NET Core

Web API with ASP.NET Core

Speed
Multiprocessor
Parallel execution

Async ≠ Parallel ≠ Threads

Non-blocking UI,
background tasks,
asynchronous

Low-level building
block
Do not use directly!

XML

XML

eXtensible Markup Language

Markup language like HTML

Designed to carry data, not to display data

Tags are not predefined – You must define your own tags

Designed to be self-descriptive

XML Does Not Do Anything

XML was created to structure, store, and transport information

```
<?xml version="1.0" encoding="UTF-8"?>
<note id="1">
  <to>Kathleen B.</to>
  <from>Tom W.</from>
  <subject>Reminder</subject>
  <body>Don't forget small change!</body>
</note>
```

How Can XML be Used?

Separates data from HTML

Simplifies data sharing

Simplifies data transport

Simplifies platform changes

Used to create new (Internet) languages

- XHTML
- WSDL for describing web services
- RSS and ATOM for news feeds
- XAML

JSON

JSON

JavaScript Object Notation

Lightweight text-data interchange format

Language independent (uses JavaScript syntax)

"Self-describing" and easy to understand

JSON Syntax

(subset of the JavaScript object notation syntax)

Data is in name/value pairs

Data is separated by commas

Curly braces hold objects

Square brackets hold arrays

JSON Name/Value Pairs

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

```
"firstName": "John"
```

This is simple to understand, and equals to the JavaScript statement:

```
firstName = "John"
```

JSON Data Types

Number (integer or floating point)

String (in double quotes)

Boolean (true or false)

Array (in square brackets)

Object (in curly brackets)

null

Examples

Objects

```
{ "firstName": "John", "lastName": "Doe" }
```

Array

```
[
  { "firstName": "John", "lastName": "Doe" },
  { "firstName": "Jane", "lastName": "Doe" },
  { "firstName": "John", "lastName": "Smith" }
]
```

Best of both worlds?

https://www.ibm.com/support/knowledgecenter/en/SS9H2Y_7.7.0/com.ibm.dp.doc/json_jsonxconversionexample.html

REST

REpresentation

http://

HTTP request

URI: string

Method: string

Header: key/value pairs

Body: string/binary

HTTP response

Status-Code: number

Header: key/value pairs

Body: string/binary

REST

Maps your CRUD actions to HTTP verbs

Action	Verb
Create	POST
Read (Retrieve)	GET
Update (Replace)	PUT
Update (Modify)	PATCH
Delete	DELETE

HTTP status codes

Code	Meaning
200	OK
201	Created
202	Accepted
204	No Content
301	Moved Permanently
302	Found (Previously "Moved temporarily")
307	Temporary Redirect
308	Permanent Redirect

Code	Meaning
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
409	Conflict
415	Unsupported Media Type
422	Unprocessable Entity
500	Internal Server Error
501	Not Implemented
503	Service Unavailable

HTTP headers

Header Field	Description	Examples
Accept	Content-Types that are acceptable for the response	text/plain application/json application/xml
Content-Type	The MIME type of the body of the request (POST, PUT, and PATCH)	application/x-www-form-urlencoded application/json; charset=utf-8
Authorization	Authentication credentials for HTTP authentication	Bearer ey…

Why REST?

Simple, both conceptually and programmatically

Simpler and cleaner than SOAP

REST is the new black

HTTP request

URI: https://futurama.com/api/characters

Method: POST

Header:

Content-Type: application/json; charset=utf-8

Authorization: Bearer ey...

Body:

```
{
  "name": "Bender",
  "species": "Robot",
  "planet": "Earth"
}
```

HTTP response

Status-Code: 201

Header:

Content-Type: application/json; charset=utf-8

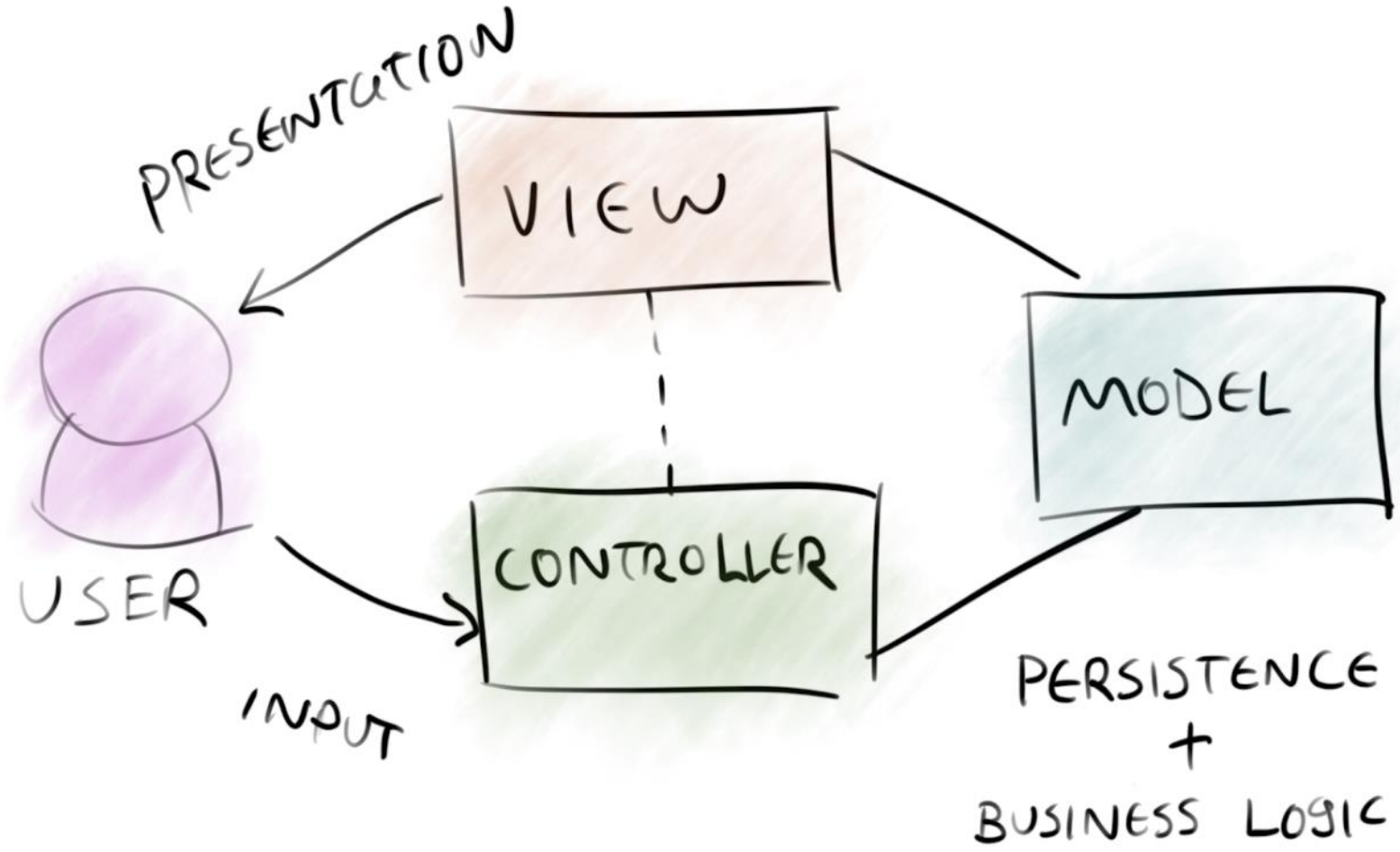
Location: https://futurama.com/api/characters/42

Body:

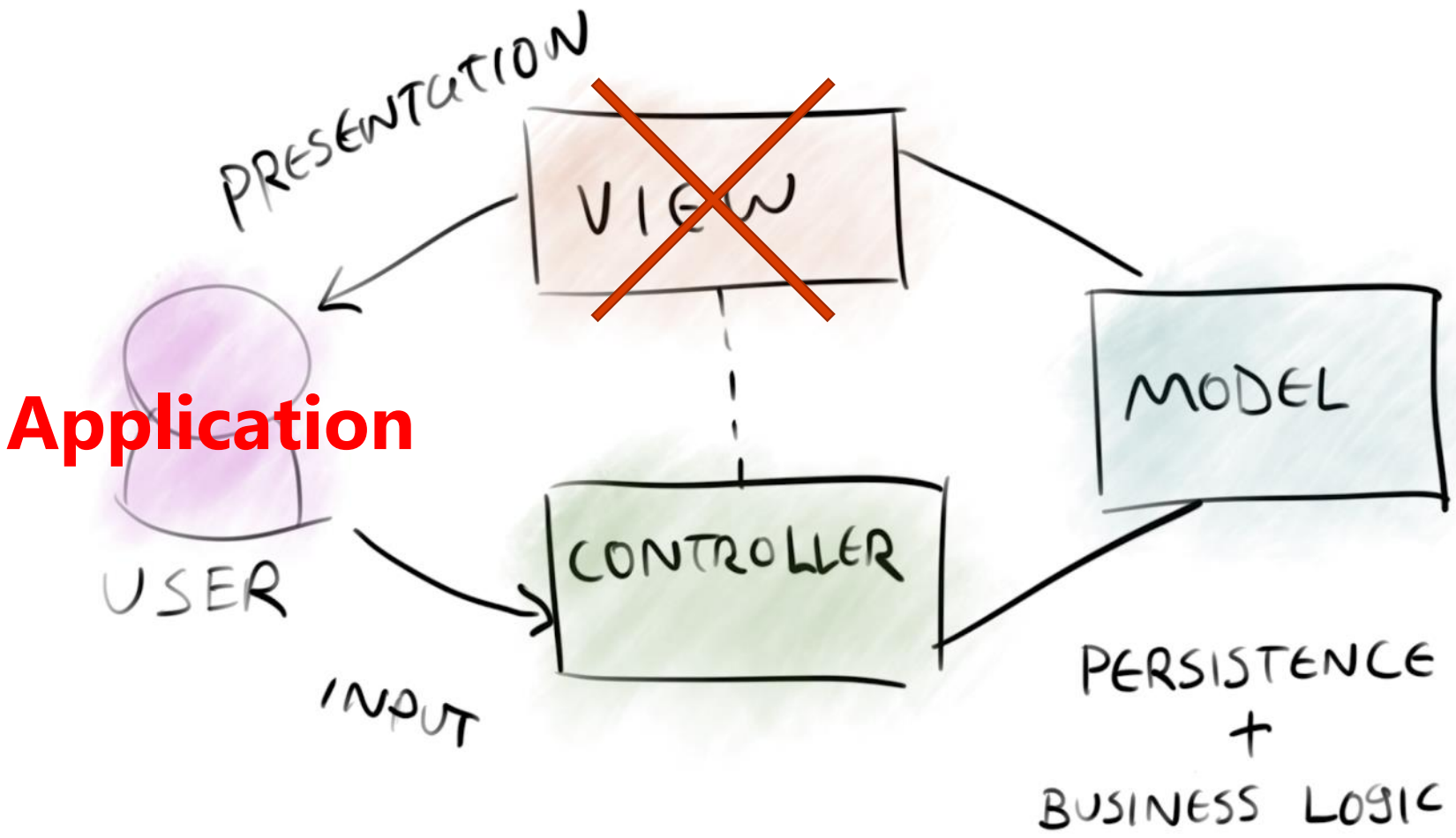
```
{
  "id": 42,
  "name": "Bender",
  "species": "Robot",
  "planet": "Earth"
}
```

ASP.NET Core

Model – View – Controller



ASP.NET Web API



ASP.NET Core Web API

Demo

Controller

Class

Derive from `ControllerBase`

Decorate with `[ApiController]` and `[Route("[controller]")]`

Method

Decorate with `[HttpGet]`, `[HttpPost]`, `[HttpPut]`, `[HttpDelete]`

Return *specific type*, `IActionResult`, or `ActionResult<T>` - or `Task<...>`.

ASP.NET (Web API) best practices

Be RESTful

Use proper status codes

Use meaningful routes

Use proper HTTP methods

Don’t throw exceptions

Use *user secrets* in development

```
dotnet user-secrets init
dotnet user-secrets set "ConnectionStrings:<connectionstring-name>" "<connnection-string>"
```

Use built-in IoC container

```
services.AddDbContext<MyContext>(o =>
    o.UseSqlServer(Configuration.GetConnectionString("MyConnectionStringName"))
);
services.AddScoped<IMyContext, MyContext>();
services.AddScoped<IMyRepository, MyRepository>();
```

Consider running migrations on load

```
using (var serviceScope = app.ApplicationServices
    .GetRequiredService<IServiceScopeFactory>().CreateScope())
{
    var context = serviceScope.ServiceProvider.GetService<SuperheroContext>();
    context.Database.Migrate();
}
```

Support the OpenAPI Specification (Swagger)

```
dotnet add package Swashbuckle.AspNetCore --version 5.0.0-rc4

services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "My API", Version = "v1" });
});

app.UseSwagger();
app.UseSwaggerUI(c =>
{
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
    c.RoutePrefix = string.Empty;
});
```

Support HTTPS *only*

```
dotnet dev-certs https --trust
```

```
app.UseHttpsRedirection();
```

Standardize on lowercase urls

```
services.AddRouting(options => options.LowercaseUrls = true);
```

Secure your Web API

Azure AD (lecture 11)

Azure AD B2C

Other 3

Do not write your own security layer!!!