C♯
JSON and the REST

Rasmus Lystrøm
Associate Professor
ITU

Agenda

XML
SOAP
JSON
YAML
REST
ASP.NET Core
Web API with ASP.NET Core
Testing ASP.NET Core
gRPC
GraphQL

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
<?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

SOAP

SOAP request

```
POST https://store.com/services/PriceService/getPrice
```

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="https://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

Source: https://www.w3schools.com/xml/xml_soap.asp

SOAP response

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPriceResponse xmlns:m="https://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

Source: https://www.w3schools.com/xml/xml_soap.asp

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:

```
"givenName": "John"
```

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

```
givenName = "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

```
{ "alterEgo": "Batman", "firstAppearance": 1939 }
```

```
[
  { "givenName": "Bruce", "surname": "Banner" },
  { "givenName": "Bruce", "surname": "Wayne" },
  { "givenName": "Bruce", "surname": "Gordon" }
]
```

```
null
```

```
"Batman"
```

Best of both worlds?

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

YAML

YAML

YAML Ain't Markup Language

Human-readable data-serialization language.

It is commonly used for configuration files and in applications where data is being stored or transmitted.

Superset of JSON

YAML Example

```yaml
version: '3.9'

services:
  sql-server:
    image: mcr.microsoft.com/mssql/server:2019-latest
    hostname: sql-server
    container_name: sql-server
    ports:
      - 1433:1433
    environment:
      ACCEPT_EULA: 'Y'
      SA_PASSWORD_FILE: /run/secrets/sa_password
      MSSQL_PID: Express
secrets:
  sa_password:
    file: ./local/sa_password.txt
```

REST

REpresentation

http://

HTTP request

URI: string

(Query): key/value pairs

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
418	I’m a Teapot
422	Unprocessable Entity
500	Internal Server Error
501	Not Implemented
503	Service Unavailable

RFC2324: Hyper Text Coffee Pot Control Protocol

In practice, most automated coffee pots cannot currently provide additions.

2.3.2 418 I'm a teapot

Any attempt to brew coffee with a teapot should result in the error code "418 I'm a teapot". The resulting entity body MAY be short and stout.

3. The "coffee" URI scheme

Because coffee is international, there are international coffee URI schemes. All coffee URL schemes are written with URL encoding of the UTF-8 encoding of the characters that spell the word for "coffee" in any of 29 languages, following the conventions for internationalization in URIs [URLI18N].

```
coffee-url  =  coffee-scheme ":" [ "//" host ]
               ["/" pot-designator ] ["?" additions-list ]
```

Source: https://tools.ietf.org/html/rfc2324

HTTP headers

Header Field	Description	Examples
Accept	Content-Types that are acceptable for the response	text/plain application/json application/xml
Content-Type	The media 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

REST request

```
GET https://store.com/prices/Apples
```

REST response

```
1.9
```

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

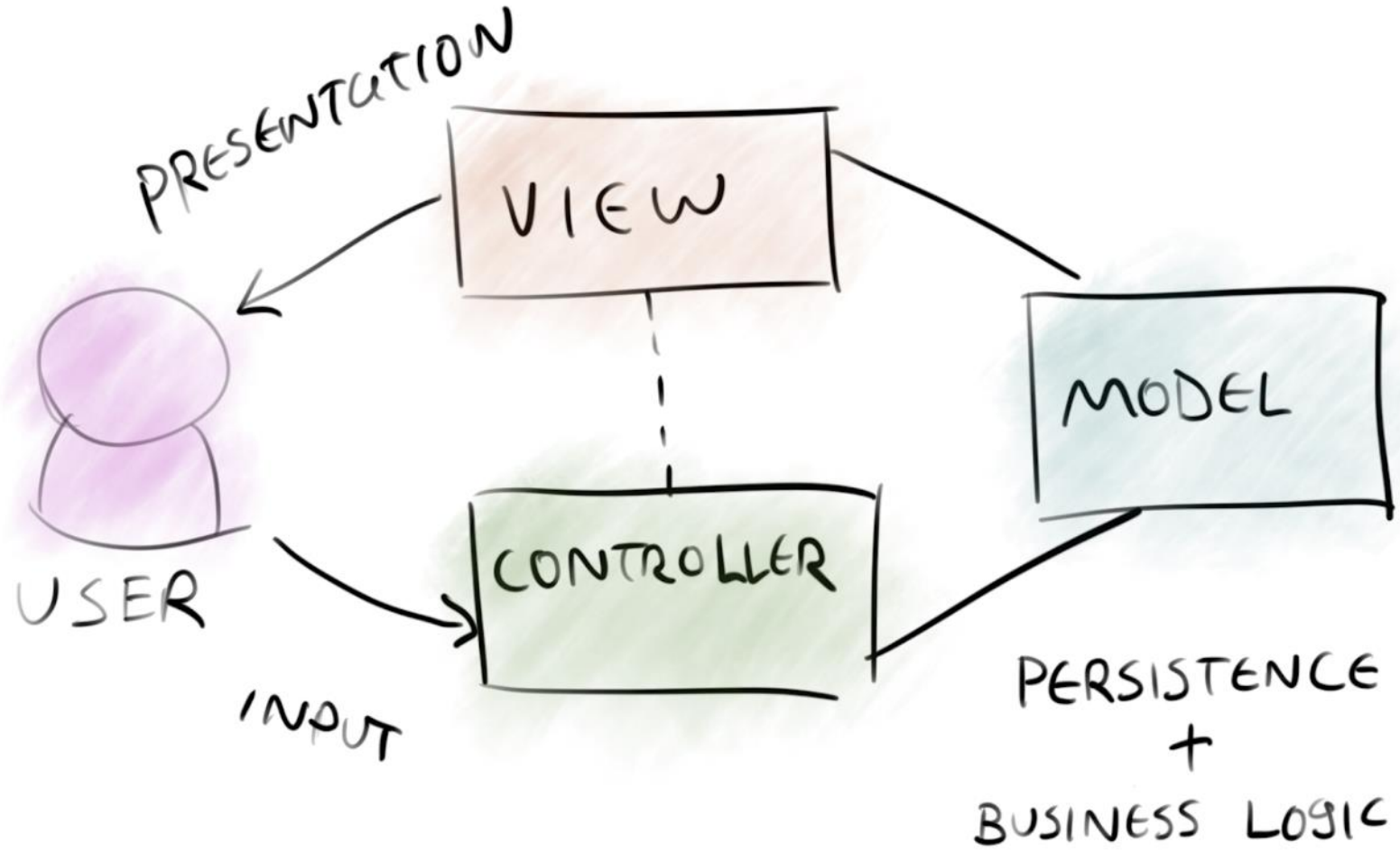
Location: https://futurama.com/api/characters/42

Body:

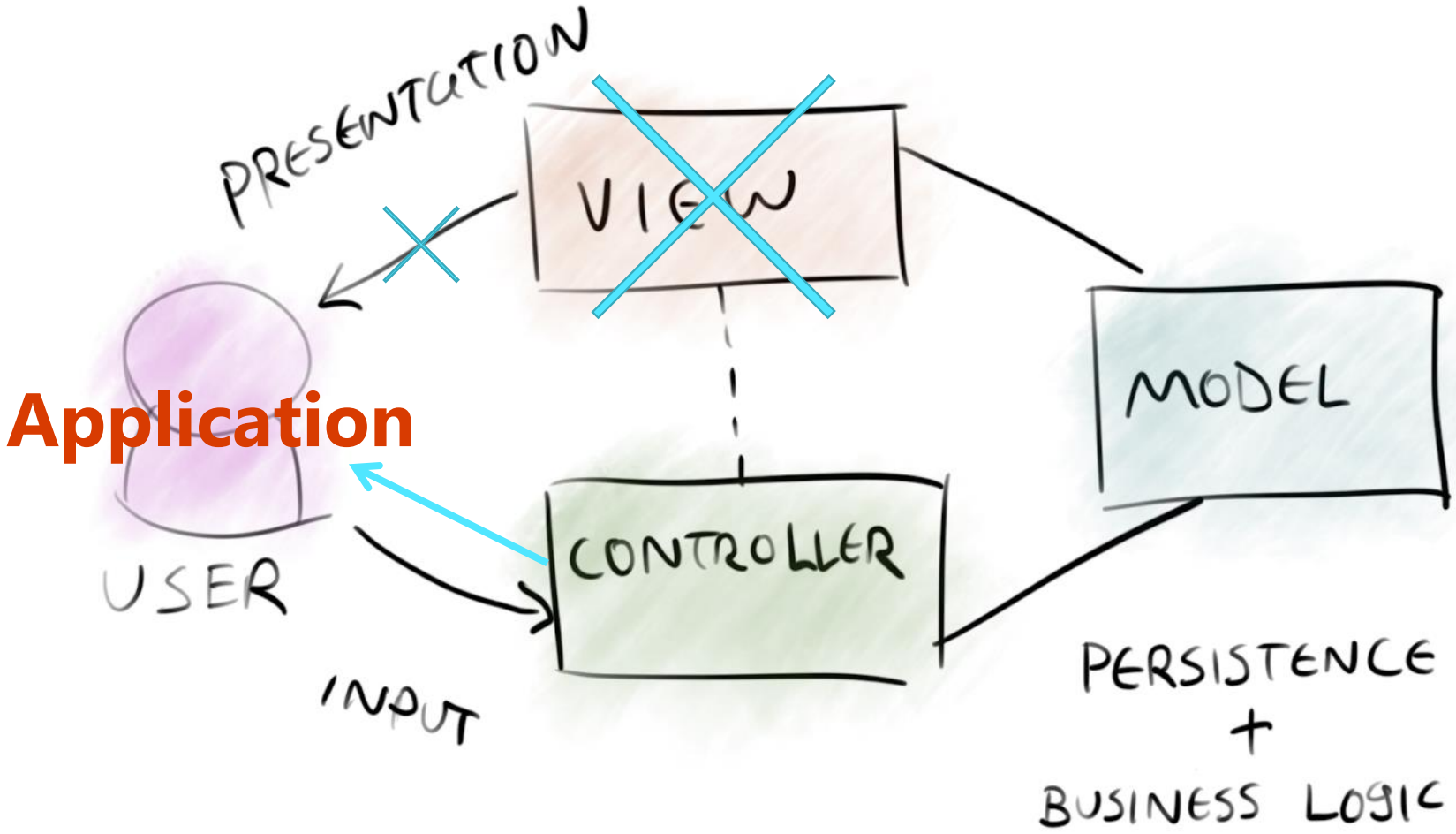

```
{
  "id": 42,
  "name": "Bender",
  "species": "Robot",
  "planet": "Earth"
}
```

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 `async 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

using Microsoft.OpenApi.Models;

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 (later)

Azure AD B2C

3rd part

Do not write your own security layer!!!

Testing ASP.NET Core

Stub → Mock → Fake

Stubs

```
public class FooServiceFalseStub : IFooService
{
    public bool Bar(Foo foo)
    {
        return false;
    }
}
```

Stubs

```
[Fact]
public void Exec_when_IFooService_Update_returns_false_returns_false()
{
    IFooService service = new FooServiceTrueStub();

    var baz = new Baz(service);

    var result = baz.Exec(new Foo());

    Assert.False(result);
}
```

Mocks

```
[Fact]
public void Exec_when_IFooService_Update_returns_false_returns_false()
{
    var mock = new Mock<IFooService>();
    IFooService service = mock.Object;

    var baz = new Baz(service);

    var result = baz.Exec(new Foo());

    Assert.False(result);
}
```

Mocks

```
[Fact]
public void Exec_when_IFooService_Update_true_returns_true()
{
    var mock = new Mock<IFooService>();
    mock.Setup(m => m.Update(It.IsAny<Foo>())).Returns(true);

    var baz = new Baz(mock.Object);

    var result = baz.Exec(new Foo());

    Assert.True(result);
}
```

Testing ASP.NET Core with Moq

Demo

gRPC

gRPC Remote Procedure Calls

gRPC is a modern, open source, high-performance remote procedure call (RPC) framework that can run anywhere. gRPC enables client and server applications to communicate transparently, and simplifies the building of connected systems.

```
$ dotnet new grpc
```

Source: https://github.com/grpc/grpc

GraphQL

GraphQL

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

Source: https://graphql.org/