Web API

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#csharp-web

Table of Contents

JSON

JSON

- **JavaScript Object Notation (JSON)** is an open-standard file format
  - Uses human-readable text to transmit data objects
  - Data objects consist of **attribute-value** pairs or **array** data types
    - Basically any serializable value
  - Easy for humans to **read** and **write**
  - Easy for machines to **parse** and **generate**
- **JSON is derived from JavaScript**
  - However, it is **language-independent**
  - Now many languages provide code to **generate** and **parse JSON**

```
{
    "firstName": “Peter",
    "courses": ["C#", "JS", "ASP.NET"]
    "age": 23,
    "hasDriverLicense": true,
    "date": "2012-04-23T18:25:43.511Z",
    // ...
}
```

JSON

- **JSON** is a very common **data format** used in web communication
  - Mainly in browser-server or server-server communication
  - The official internet media type (**MIME**) for **JSON** is `application/json`
  - **JSON** files use the extension **`.json`**
- **JSON** is commonly used as a replacement for XML in AJAX systems
  - **JSON** is shorter and easier to comprehend
  - **JSON** is quicker to read and write, and is more intuitive
  - **JSON** doesn't support schemas and namespaces

XML

XML

- **XML** defines a set of rules for encoding documents
  - Stands for **Ex**tensible **M**arkup **L**anguage
  - Similar to **JSON**
    - In terms of **human-readability** and **machine-parsability**
    - In terms of hierarchy (values within values)
- **XML** is a textual data format
  - Strong support for different human languages via **Unicode**
  - The design focuses strongly on actual documents

XML

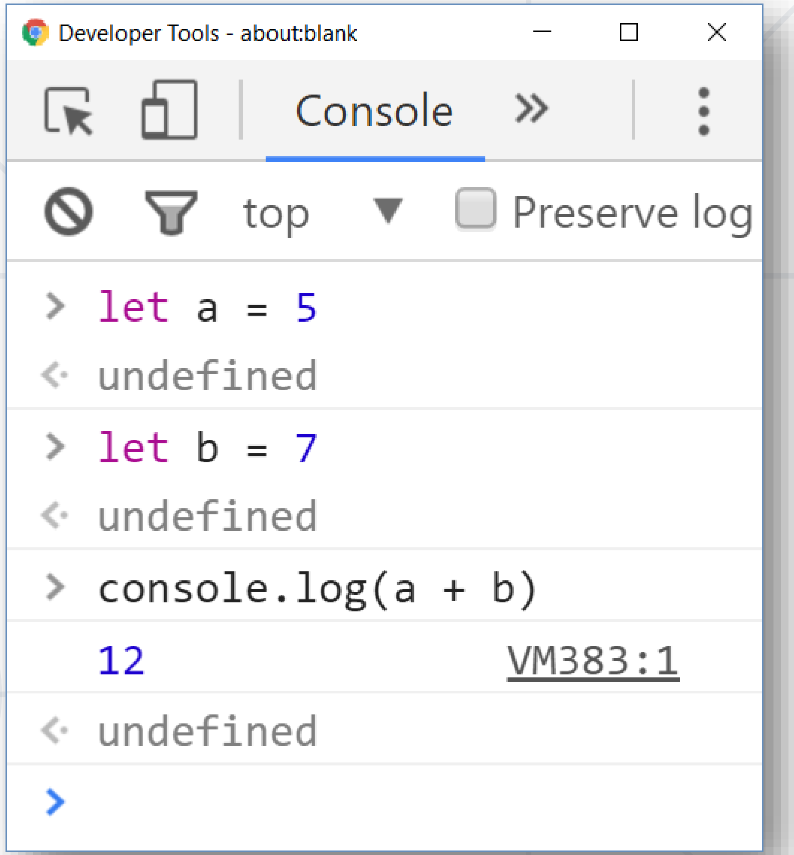
- **XML** has many applications
  - There are 2 **MIME** types for **XML** `application/xml` and `text/xml`
  - **XML** files use the extension **.xml**
- **XML** has many applications
  - Widely-used in **SOA** (e.g. WCF)
  - Used for **configuring** .NET apps
  - Used in **Microsoft Office** formats
  - **XHTML** was intended to be strict **HTML** format

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<records>
    <record id="1">
        <name>Ivo</name>
        <email>ivo@softuni.bg</email>
        <company>Software University</company>
    </record>
    <record id="2">
        <name>Niki</name>
        <email>admin@Nikolay.it</email>
        <company>ZenCodeo</company>
    </record>
</records>
```

JavaScript

Welcome to JavaScript

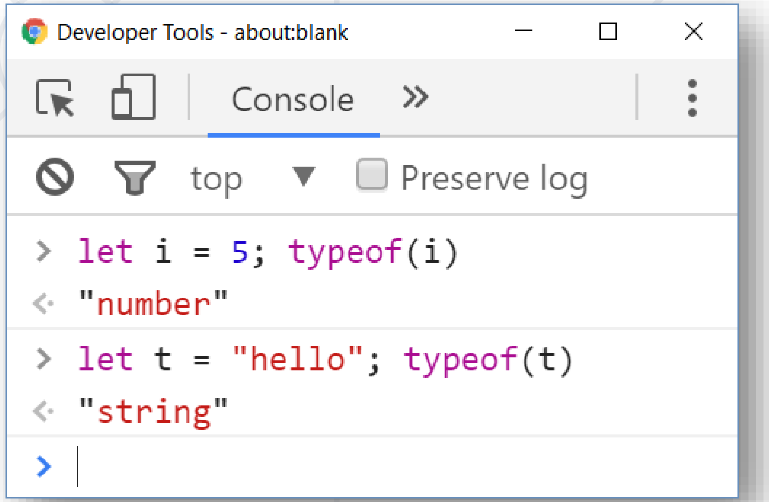
- JavaScript (JS) is a scripting language
  - Executes commands (script)
  - Can work in interactive mode
  - No compilation, just execute commands
- Alongside HTML & CSS, JavaScript is one of the 3 core technologies of the World Wide Web
  - JavaScript enables dynamics and interactivity in web pages
    - Has DOM and browser API (notifications, geolocation, ...) access



```
> let a = 5
<· undefined
> let b = 7
<· undefined
> console.log(a + b)
  12                VM383:1
<· undefined
>
```

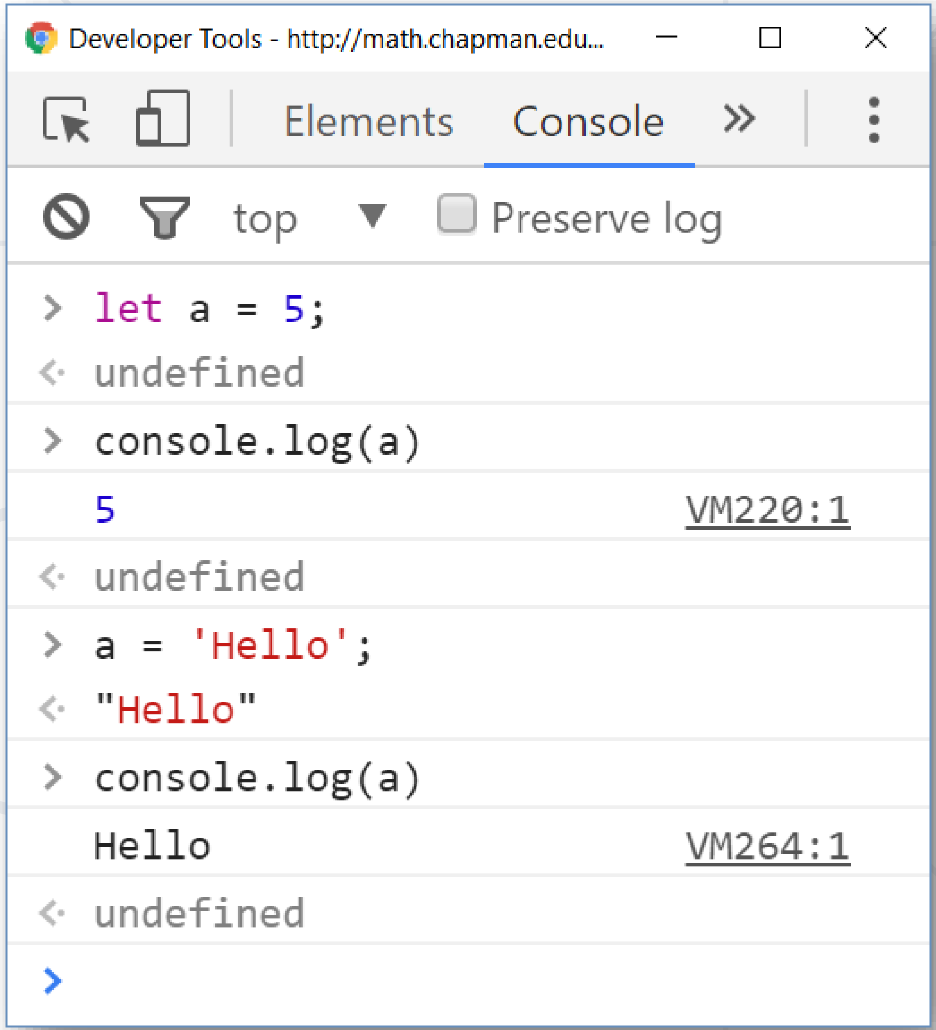
Welcome to JavaScript

- **JavaScript** (JS) is untyped language
  - **Untyped** (dynamically typed) == variables have no types (but…)
  - Data (values) still have a type



```
> let i = 5; typeof(i)
<· "number"
> let t = "hello"; typeof(t)
<· "string"
>
```

  - More Info: Link 1, Link 2, Link 3



```
> let a = 5;
<· undefined
> console.log(a)
  5                                  VM220:1
<· undefined
> a = 'Hello';
<· "Hello"
> console.log(a)
  Hello                              VM264:1
<· undefined
>
```

Welcome to JavaScript

- **JavaScript** was initially only implemented client-side in web browsers
  - JavaScript engines, nowadays, are embedded in many types of software
  - **Server-Side** JavaScript, **Mobile** applications, **Desktop** Applications, etc.
- **JavaScript** is one of the most popular technologies on the Web
  - If not the most popular, that is...
  - The rise of **SPA**s and **JavaScript-heavy** sites certainly prove that
- One of the most important techniques around JS is **AJAX**
  - **A**synchronous **J**avaScript **a**nd **X**ML
- **TypeScript** is a typed superset of JS that compiles to plain JS

Asynchronous JavaScript and XML

AJAX

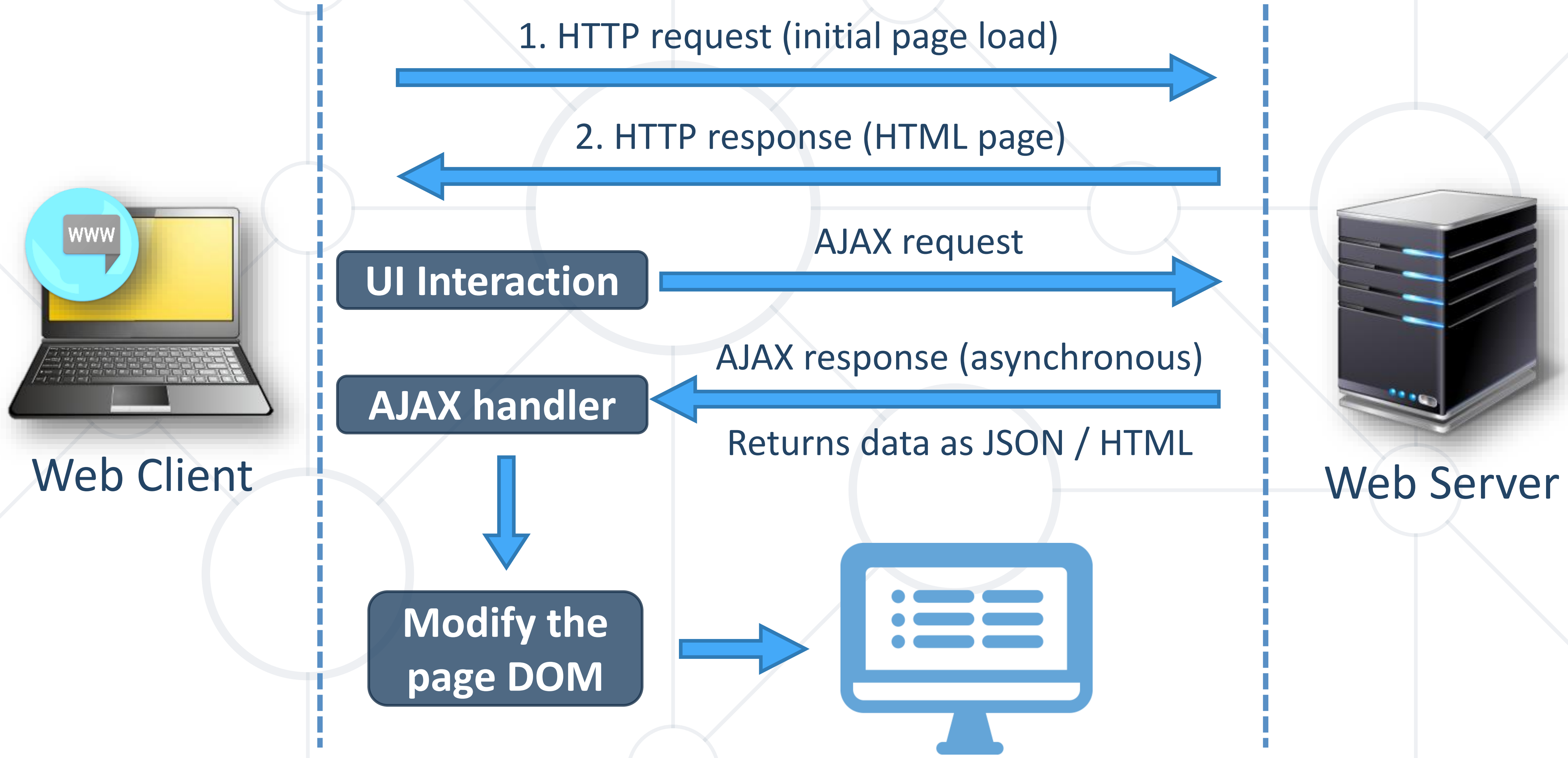
AJAX

- AJAX is not a programming language (despite its "individual" popularity)
- AJAX == set of web development techniques
  - Used to create asynchronous web applications
  - Using AJAX, you can send and retrieve data to and from a server asynchronously – in the background via HTTP requests
- AJAX is a developer's dream, because you can:
  - Read data from a web server - after the page has loaded
  - Update a web page (or parts of it) without reloading the page
  - Send data to a web server - in the background

AJAX

- **AJAX** works very simply, using a combination of
  - **`XMLHttpRequest`**
    - To request data from a server
  - **JavaScript** and **DOM**
    - To display and / or use data
- **AJAX** is a misleading name
  - **AJAX** apps might use **XML** to transport data
  - However it is **equally common** to transport data as **JSON**

AJAX: Workflow

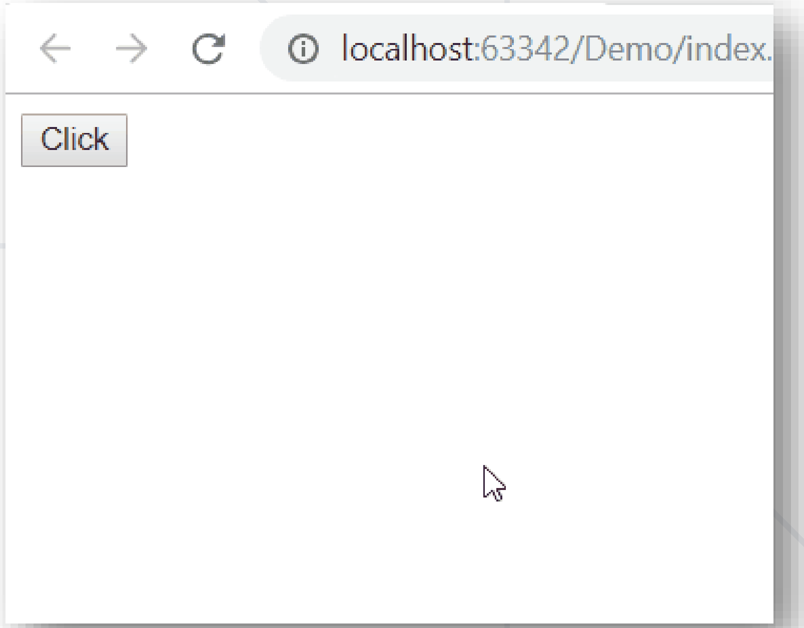

1. HTTP request (initial page load)
2. HTTP response (HTML page)
UI Interaction
AJAX request
AJAX response (asynchronous)
AJAX handler
Returns data as JSON / HTML
Modify the page DOM
Web Client
Web Server
www

AJAX in Plain JavaScript (Vanilla JS)

- An **AJAX Request** example:

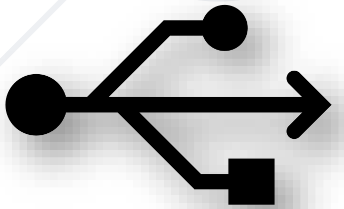
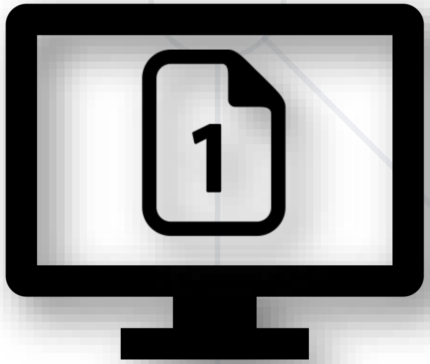
```
function loadHtml()
{
    // Initializes a XMLHttpRequest object
    let xhttp = new XMLHttpRequest();

    xhttp.onreadystatechange = function() {
        // readyState holds the status of the XMLHttpRequest
        // (4) means Request finished and Response is ready
        // status holds the status code
        if (this.readyState == 4 && this.status == 200) {
            // Load the Response text into the body of the document
            document.body.innerHTML = this.responseText;
        }
    }; // Defines a function, called when the readyState Is changed

    // Specify the request (method, url, async, etc...)
    xhttp.open("GET", "/api/Data", true);
    // Send the request to the server
    xhttp.send();
}
```

```
...
<button onclick="loadHtml()">
    Click
</button>
...
```



AJAX in SPA

- **Single-Page apps** are pretty common nowadays
  - Based on dynamic and asynchronous content changing
  - **AJAX** is pretty much used in almost every **SPA**
    - **SPA**s use **AJAX** to provide better and dynamic-data-filled apps
- **AJAX** is used to make a smooth changes on the page
  - This ensures a better UX design and dynamic UI

jQuery

Write Less, Do More!

What is JQuery?

- jQuery is a cross-browser JavaScript library
  - Dramatically simplifies DOM manipulation
  - Free, popular, open-source software: https://jquery.com

```
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
```

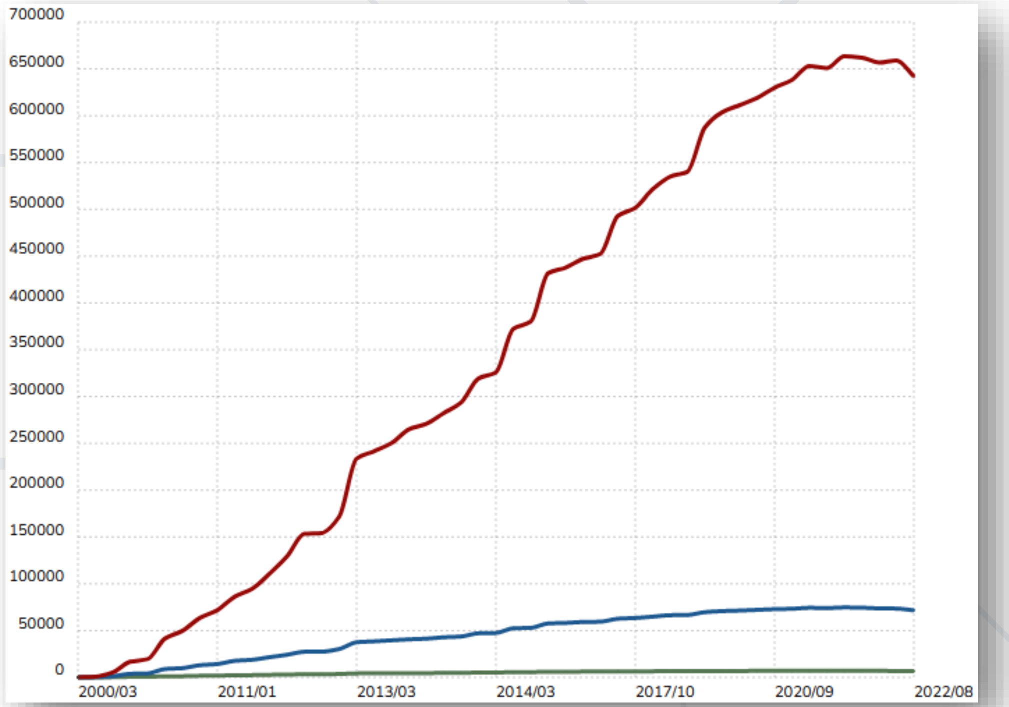
Load jQuery from its official CDN

```
$('li').css('background', '#DDD');
```

Change the CSS for all `<li>` tags

Why jQuery?

- Extremely popular
  - 83 000 000 sites use jQuery (68.8% of top 1 million sites)
  - http://trends.builtwith.com/javascript/jQuery
- Easy to learn
- Large community
- Cross-browser support
- Official web site: http://jquery.com



Selection with jQuery

- **jQuery**'s selectors return a collection of matched items
  - Works with CSS3 selectors with few jQuery-specific
  - Even if there is only one item

```
$('div') //Gets all elements with the provided tag
$('.menu-item') // Gets all elements with the provided class
$('#navigation') // Gets the element with the provided id
$('ul.menu li') // Gets all elements corresponding to the query selector
```

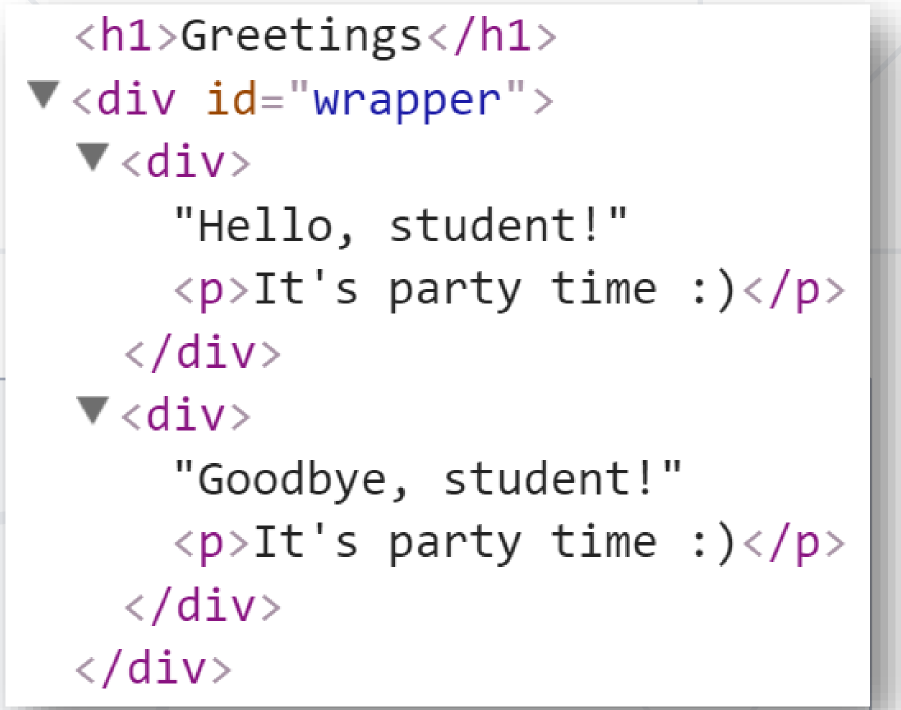
  - http://learn.jquery.com/using-jquery-core/selecting-elements/
- Selected elements can be processed as a group

```
$('div').css('background', 'blue'); // Make all DIVs blue
```

Adding Elements with jQuery

- Select the parent element, then use:
  - append() / prepend()
  - appendTo() / prependTo()

```
<div id="wrapper">
  <div>Hello, student!</div>
  <div>Goodbye, student!</div>
</div>
```



```
<h1>Greetings</h1>
▼<div id="wrapper">
  ▼<div>
      "Hello, student!"
      <p>It's party time :)</p>
    </div>
  ▼<div>
      "Goodbye, student!"
      <p>It's party time :)</p>
    </div>
  </div>
```

```
$('#wrapper div').append("<p>It's party time :)</p>");
```

```
$('<h1>Greetings</h1>').prependTo('body');
```

Creating / Removing Elements

```
let div = $('<div>');
div.text('I am a new div.');
div.css('background', 'blue');
div.css('color', 'white');
$(document.body).append(div);
```

```
let paragraph = $('<p>Some text</p>');
paragraph.appendTo(div);
```

```
$('div').remove();
```

JQuery Events: Attach / Remove

- **Attaching** events on certain elements

```
$('a.button').on('click', buttonClicked);
function buttonClicked() {
    $('.selected').removeClass('selected');
    $(this).addClass('selected');
    // "this" is the event source (the hyperlink clicked)
}
```

- **Removing** event handler from certain elements

```
$('a.button').off('click', buttonClicked);
```

jQuery AJAX

Simplified AJAX Calls with jQuery

jQuery vs Native XMLHttpRequest – GET

- **jQuery** dramatically simplifies how developers make **AJAX** calls

```
$.ajax({
    method: 'GET',
    url: 'myservice/username',
    data: { id: '42' }
})
.done(function success(data) {
    alert('User\'s name is ' + data);
})
.fail(function fail(data, status) {
    alert('Request failed. Returned status
of ' + status);
});
```

```
var xhr = new XMLHttpRequest();

xhr.open('GET', 'myservice/username?id=42');

xhr.onload = function() {
    if (xhr.status === 200) {
        alert('User\'s name is ' +
xhr.responseText);
    }
    else {
        alert('Request failed.  Returned
status of ' + xhr.status);
    }
};

xhr.send();
```

jQuery AJAX

- **jQuery** simplifies how developers make AJAX calls
- Low-Level Interface
  - **`jQuery.ajax()`**
    - Perform an asynchronous HTTP (Ajax) request
  - **`jQuery.ajaxPrefilter()`**
    - Handle custom AJAX options or modify existing options before each request is sent and before they are processed by `$.ajax()`
  - **`jQuery.ajaxSetup()`**
    - Set default values for future Ajax requests. Its use is **not recommended**
  - **`jQuery.ajaxTransport()`**
    - Creates an object that handles the actual transmission of AJAX data

jQuery AJAX

- Shorthand Methods
  - `jQuery.get()`
    - Load data from the server using a HTTP GET request
  - `jQuery.getJSON()`
    - Load JSON-encoded data from the server using a GET HTTP request
  - `jQuery.getScript()`
    - Load a JavaScript file from the server using a GET HTTP request, then execute it
  - `jQuery.post()`
    - Load data from the server using a HTTP POST request
  - `.load()`
    - Load data from the server and place the returned HTML into the matched element

Web Services

Communication between Systems and Components

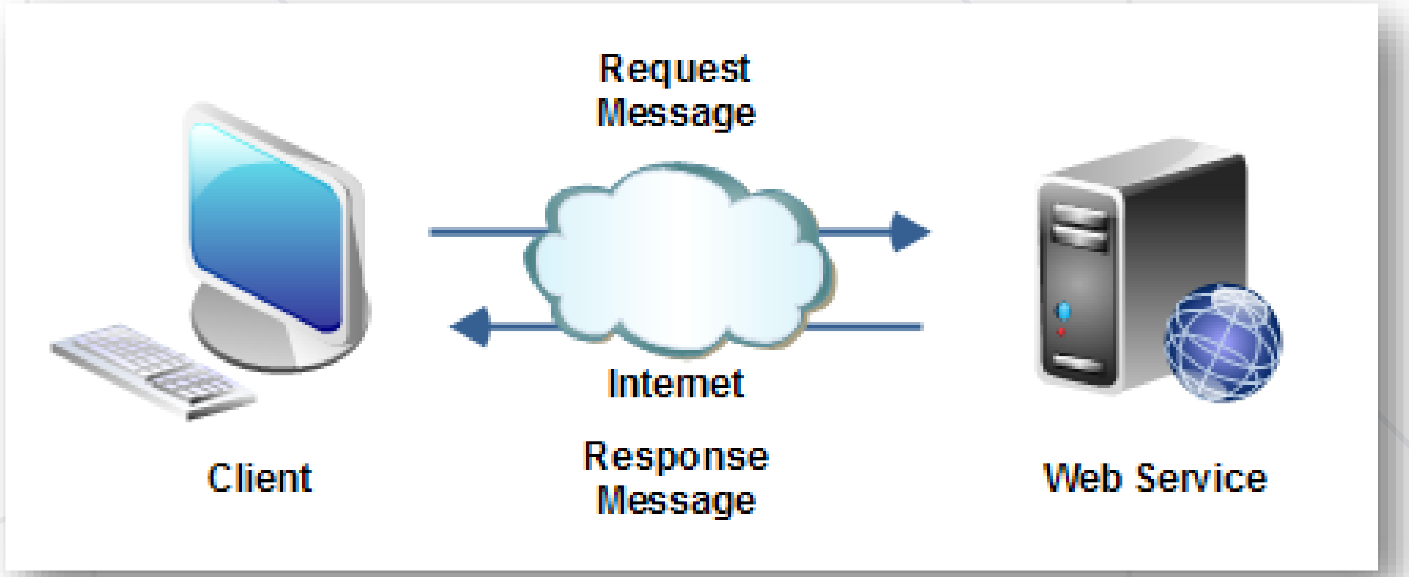
What is API?

- **API** == **A**pplication **P**rogramming **I**nterface
  - Designed for communication between system components
  - Set of **functions** and **specifications** that software programs and components follow to talk to each other
  - Examples
    - JDBC – Java API for apps to talk with database servers
    - Windows API – Windows apps talk with Windows OS
    - Web Audio API – play audio inthe Web browser with JS

Car
Web Apps
Enterprise Application
Mobile Device
API
Game
Sensors
TV
Wearable Device
API

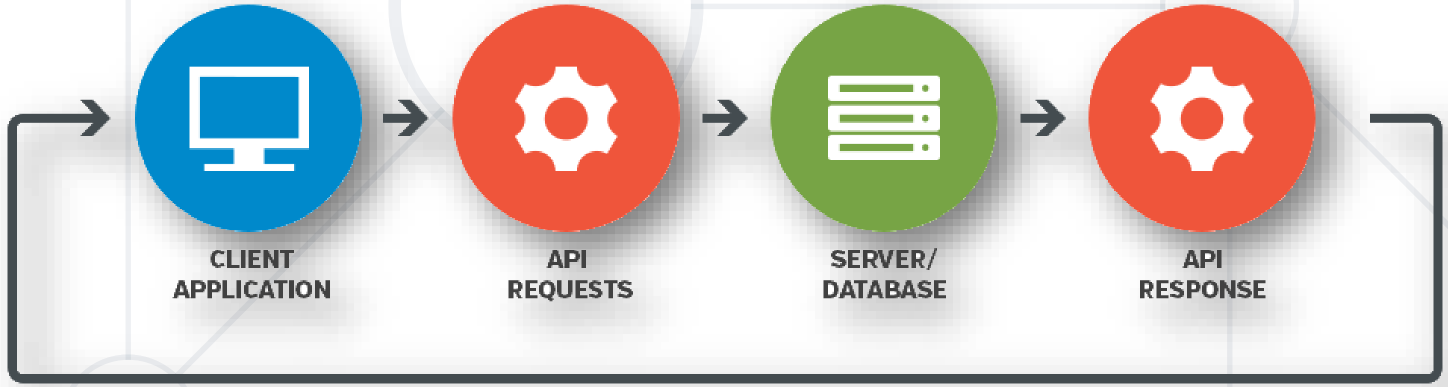
What is Web Service?

- **Web services** implement **communication** between software **systems** or **components** over the **network**
  - Using standard **protocols**, such as HTTP, JSON and XML
  - Exchanging **messages**, holding data and operations
- All **web services are APIs**, but not all APIs are web services



Web Services and APIs

- **Web services** expose **back-end APIs** over the **network**
  - May use different **protocols** and **data formats**: HTTP, REST, GraphQL, gRPC, SOAP, JSON-RPC, JSON, BSON, XML, YML, …
- **Web services** are hosted on a Web server (HTTP server)
  - Provide a set of functions, invokable from the Web (Web API)
- **RESTful APIs** is the most popular Web service standard



Web API / Server-Side API

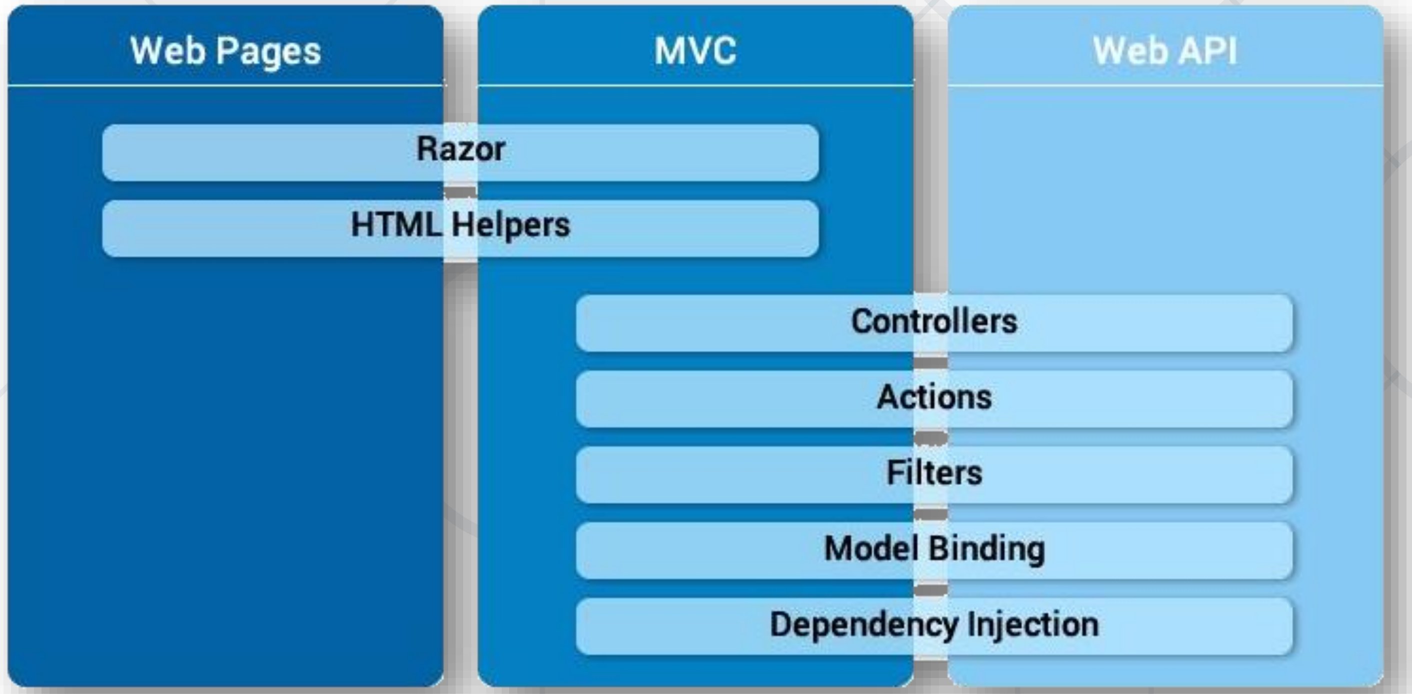
Server-Side Application Programming Interface

Web API / Server-Side API

- **Web API** == application programming interface, exposed in Internet
  - Used by **Web browsers (SPA)**, **mobile applications**, **games**, **desktop applications**, **Web server**, etc.
- **Server-side Web APIs** consist of publicly exposed endpoints
  - The **endpoints** correspond to a defined request-response message system
  - Communication is typically expressed in **JSON** or **XML** format
  - Communication is typically performed over an Internet protocol
    - Most commonly, **HTTP** – through a **HTTP-based** web server

ASP.NET Core Web API

- Creating a **Web API** with **ASP.NET Core** is pretty straightforward
  - You build controllers and they have actions
  - In this case though, the **actions** are in the role of **endpoints**



```
[Route("api/[controller]")]
[ApiController]
public class ProductsController
   : ControllerBase
{
    ...
}
```

ASP.NET Core Web API Controller

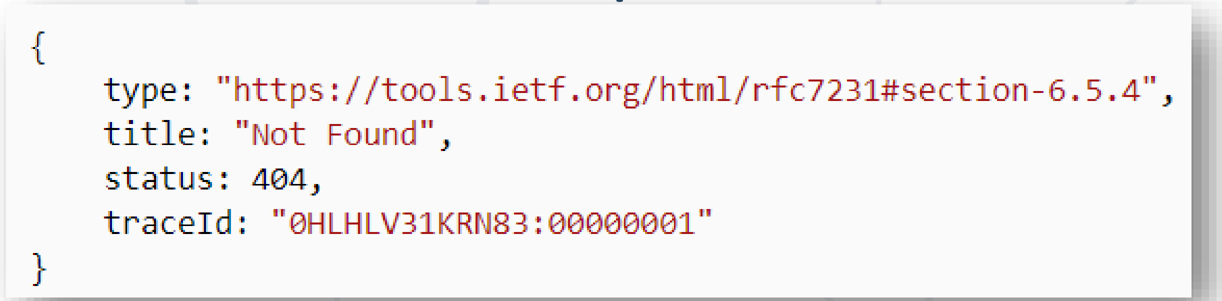
- Web API controllers should
  - Inherit the **ControllerBase** class
  - Be annotated with **[ApiController]** and **[Route]** attribute

```
[Route("api/[controller]")]
[ApiController]
public class ProductController : ControllerBase
{
    private readonly IProductService productService;

    public ProductController(IProductService productService)
    {
        this.productService = productService;
    }
}
```

The **Model** and the **Service** can be anything. The techniques surrounding the **controller** are what is essential as they define the **API**

ASP.NET Core Web API (ApiController)

- The `[ApiController]` annotation provides several convenient features
  - Automatic HTTP 400 responses (for model state errors)
  - Binding source parameter inference
  - Multipart / Form-data request inference
  - Attribute routing requirement
  - Problem details responses for error status codes



```
{
    type: "https://tools.ietf.org/html/rfc7231#section-6.5.4",
    title: "Not Found",
    status: 404,
    traceId: "0HLHLV31KRN83:00000001"
}
```

ASP.NET Core Web API (ApiController)

- Automatic HTTP 400 Responses
  - **Model validation** errors automatically trigger an HTTP 400 response

```
if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}
```

This is no longer necessary

- Binding source parameter inference (Binding Source Attributes)
  - The attributes define the **location of the parameter's value**

[FromBody]	[FromQuery]
[FromForm]	[FromRoute]
[FromHeader]	[FromServices]

Example

```
[HttpPost]
public IActionResult Create(
    Product product, // [FromBody] is inferred
    string name) // [FromQuery] is inferred
{
}
```

ASP.NET Core Web API (ApiController)

- Multipart / Form-data request inference
  - Achieved by putting `[FromForm]` attribute on action parameters
  - `multipart/form-data` request content type is **inferred**
- Attribute routing requirement
  - Attribute routing becomes a **requirement**

```
[Route("api/[controller]")]
[ApiController]
public class ProductsController : ControllerBase
```

ASP.NET Core Web API (ApiController)

- Problem details responses for error status codes
  - Since ASP.NET Core **2.2**, **MVC** transforms error results
  - Errors are transformed into **`ProblemDetails`**
- **`ProblemDetails`** is
  - A type based on a HTTP Api Specification for error presentation
  - A standardized format for machine-readable error details

```
if (product == null)
{
    return NotFound();
}
```

```
{
    type: "https://tools.ietf.org/html/rfc7231#section-6.5.4",
    title: "Not Found",
    status: 404,
    traceId: "0HLHLV31KRN83:00000001"
}
```

ASP.NET Core Web API (ApiController)

- These features are built-in and active by default
  - But the default behavior can be overridden

```
builder.Services.AddControllersWithViews()
    .ConfigureApiBehaviorOptions(options =>
    {
        // Suppress Multipart/form-data inference
        options.SuppressConsumesConstraintForFormFileParameters = true;
        // Suppress binding source attributes
        options.SuppressInferBindingSourcesForParameters = true;
        // Suppress automatic HTTP 400 errors
        options.SuppressModelStateInvalidFilter = true;
        // Suppress problem details responses
        options.SuppressMapClientErrors = true;
        // ...
    });
```

ASP.NET Core Web API (Return Types)

- **ASP.NET Core** offers several options for **API Endpoint** return types
  - **Specific** Type
    - The simplest action type
  - **`IActionResult`** Type
    - Appropriate when multiple **`ActionResult`** types are possible in the corresponding action

```
[HttpGet]
public IEnumerable<Product> Get()
{
    return this.productService.GetAllProducts();
}
```

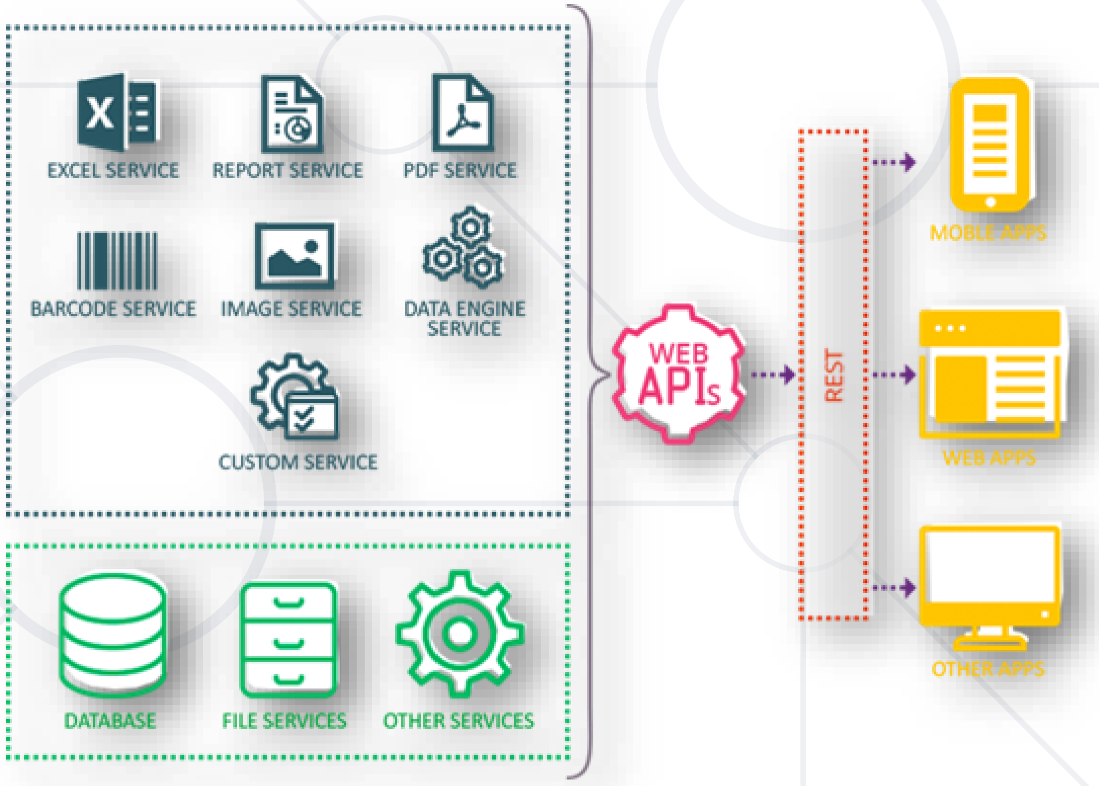
```
[HttpGet("{id}")]
[ProducesResponseType(200, Type = typeof(Product))]
[ProducesResponseType(404)]
public IActionResult GetById(int id)
{
    var product = this.productService.GetById(id);

    if (product == null) return NotFound();

    return Ok(product);
}
```

ASP.NET Core Web API (Return Types)

- It is recommended to use `ActionResult<T>` as a return type

```
[HttpGet]
public ActionResult<IEnumerable<Product>> Get()
{
    return this.productService.GetAllProducts();
}
```

```
[HttpGet("{id}")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public ActionResult<Product> GetById(int id)
{
    var product =
       this.productService.GetById(id);

    if (product == null) return NotFound();

    return product;
}
```



Web API Methods

Demo

Angular

What is Angular?

- **Angular** is a framework for building complex front-end apps
- Focused on end-to-end tooling and best practices
- Developed by the **Angular** team at **Google**

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>`
})

export class AppComponent { name = 'Angular'; }
```

Angular

- **Angular** is rewritten on Microsoft's **TypeScript** language
  - A typed **superset** of **JavaScript** that compiles to plain JS
  - Any Browser! Any Host! Any OS! Anywhere! Open Source!
- **Angular** does not have a concept of "**scope**" or **controllers**
  - Instead it uses a hierarchy of **Components**
  - This is its main difference from AngularJS (the first Angular)
  - Most modern Front-End frameworks tend to use this **architecture**

Angular Features

- Cross Platform
  - Single Page Applications (SPA)
  - Progressive Web Apps
  - Native Mobile Apps (Cordova, Ionic)
  - Desktop Apps (Electron)
- Great Tooling (CLI, IDEs, Templates)
- Huge Community
- Easy Testing, Animations, Accessibility
- Can work with any back-end (Web API, Node.js, etc.)

Summary

- JSON & XML == data formats used in web communication to transmit data objects
- JavaScript
- AJAX == set of dev techniques
- jQuery == cross-browser JS library
- Web Services implement communication between software systems or components over the network
- ASP.NET Core Web API
- Angular == framework for building complex front-end apps

Software University

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SOFTWARE GROUP

DXC TECHNOLOGY

createX

Postbank

Решения за твоето утре

SUPER HOSTING .BG

DRAFT KINGS

VIVACOM

INDEAVR

Serving the high achievers

AMBITIONED

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg