Heritage college

Lucas stephenson

lstephenson@cegep-heritage.qc.ca

Advanced *Web Applications* and *Web Services*

Students learn to create well-formed and valid XML documents and schemas, to display XML data in a web browser, and to transform XML data into different formats. They learn about AJAX technology and are introduced to a framework for building applications using service-oriented architecture. They develop applications that create and consume web services in .NET.

Course description

To coincide with modern web development practices, the class will cover additional web service standards/practices, including JSON/JSONP and a meta-api (GraphQL).

Addendum

Evaluation Activities	50%	Schedule
Lab Assignments	45	Multiple per week
Midterm	5	2nd week
Final Evaluation Activity (ies)	50%	Schedule
Course Assignment and Presentation	40	Last day
Final Exam	10	Last day
Total	**100**	

Course Evaluation

2020

Important Dates

May 4th	: Star wars day
May 18th	: No class (Victoria Day)
May 12th	: Midterm
May 13th	: Assignment Instructions
May 25th	: Last day (presentations and quiz)

COVID-19

- Class is recorded and will be available on teams, shortly after class
  - Live attendance is not mandatory
    - Suggested, so you can ask questions, etc.
- I am available on Teams chat, most of the time, please use this to communicate with me!

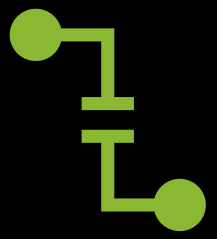
COVID-19

- For each lab
  - Make a video/screencast
  - 1-3 min
  - Show, then explain what you did and issues you had/how you resolved
  - Share with class in Microsoft teams
  - **Very Casual**

COVID-19

- Project can be done as a **group** (suggested) or as an individual
  - No restrictions on who you work with or group size
- Class will be in shorter "chunks"
  - Time to ask bigger questions, stand up/stretch, go to the washroom.

Advanced *Web Applications* and *Web Services* → This Course



***primarily* concerned with**

Transfer and consumption of data

Not logic heavy

Storing and Transferring Data

How do we store data?

What if we need to transfer it?

Data Formats

01

Most of the time we are referring to text-based data

- We have used CSV files, JSON files and some XML files

02

These formats can be

- Made up by you
- Follow a standard

Comma-Separated Values

- Store multiple records in a file
- Each line represents a text representation of a single record
- Uses commas to separate values
- Solution to column aligned data, first used to solve data entry speed in the early 70s
    - Prior mainframes used fixed width columns (every field had to use a specific number of characters


```
"1997","Ford","E350"
```

Comma-Separated Values

- Pros
  - Very simple format
  - A RDBMS table is easily stored as a CSV file
  - A RDBMS database is easily stored as multiple CSV files
  - Concise
  - Easy to decode
- Cons
  - Special characters, commas, line breaks, numbers, quotation marks, aren't well agreed upon
  - There is no way to specify type
  - Cannot contain relationships in a single document

Remember Java?

```
private static List<Record> readCSV(String fileName) {
   List<Record> records = new ArrayList<>();
   Path pathToFile = Paths.get(fileName);

   try (BufferedReader br = Files.newBufferedReader(pathToFile, StandardCharsets.US_ASCII)) {
      String line = br.readLine();
      while (line != null) {
         String[] columns = line.split(",");
         Record record = new Record(columns);
         records.add(record);
         line = br.readLine();
      }
   } catch (IOException ioe) {
      ioe.printStackTrace();
   }
   return records;
}
```

XML

eXtensible Markup Language

- Designed to store and transport data
  - Promote data types and meaning of values
- Self-Descriptive
  - All data is *marked up* with tags
- Based on standard generalized markup language (SGML)
  - Related to html
  - XML does not itself define any tags
  - Subset of SGML (to make it simpler)


```
<vehicle>
   <year>1997</year>
   <make>Ford</make>
   <model>E350</model>
</vehicle>
```

XML

eXtensible Markup Language

- Pros
  - Meaning of data is (more) clear
  - Can support complex data structures
    - Lists/sub lists
- Cons
  - More verbose
  - Specificity can be a problem
  - Does not match RDMS tables
  - Requires more complex decoder
    - You cannot write these easily or quickly
    - Most platforms have pre-written code

JSON

JavaScript Object Notation

- Designed to replace/simplify XML
  - Reduces characters
- Self-Descriptive
  - All data can be *identified with key/value pairs*
  - Many non-method JavaScript *literals* are valid
  - *Not valid:*
    - **Map, Set, Date, Error, Regular Expression, Function, Promise, and undefined**


```
{
    "vehicle": {
        "year": "1997",
        "make": "Ford",
        "model": "E350"
    }
}
```

JSON

JavaScript Object Notation

- Pros
  - Meaning of data is (more) clear
  - Can support complex data structures
    - Lists/sub lists
  - Fewer Characters than XML
  - Easy to consume with web technologies (especially JavaScript)
- Cons
  - Specificity can be a problem
  - Does not match RDMS tables

Normalization

CSV files map closely to records in a standard RDBMS table

XML and JSON files do *not*

- NoSQL aka relational database engines were created
- Store *documents* (instead of records)
  - Usually in JSON
- More difficult to query
  - Less rigid (no specified fields be default)

Reminder

Data does not *do* anything

But we can send data or process data, and *act* accordingly

Break

XML Is Powerful

There are many tools

XSLT: transform XML documents to other XML or text documents (like CSV)

XSD: A schema system for XML documents (rules for what is allowed in an XML document) which you can validate against

However

The time investment for creating these is high

Not as widely used for new development

There are many existing applications that do

XML

- Similar in syntax to html (that you've seen)
  - Tags should be camelCase, starting with a lowercase letter
  - Tags can be anything!
  - Tags can have attributes
    - Attributes and children are sometimes redundant
    - If a child is always singular, consider making it An attribute

```
<band>
    <name>The Smashing Pumpkins</name>
    <genre>Alternative rock</genre>
</band>
```

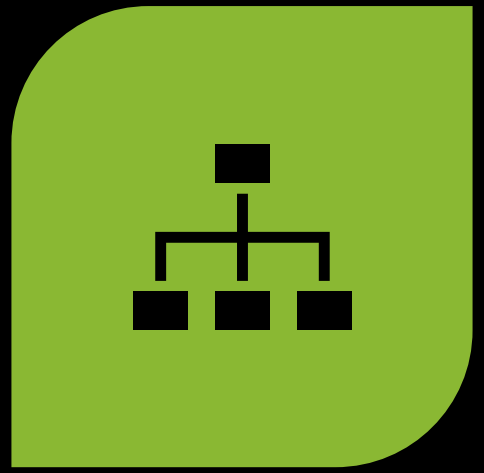
Can have multiple names!

```
<band name="The Smashing Pumpkins">
        <genre>Alternative rock</genre>
</band>
```

JSON vs. XML

- JSON has been and is in many cases, replacing XML
- XML is
  - Probably not going away
  - XML has more support for schemas
- JSON is
  - Fewer characters
  - Faster to write
  - Can have arrays
  - Is much faster for JavaScript to decode/parse
    - Node, client-side JavaScript
    - Preferred for fetch()/XMLHTTPRequest()

XSD
XML Schema Definition



A SCHEMA DESCRIBES THE STRUCTURE

AN XML SCHEMA DESCRIBES THE STRUCTURE OF AN XML DOCUMENT

Let's Make An XML Document

Describe a Restaurant Menu (Lab)

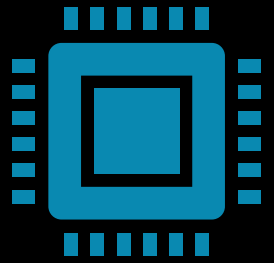
Break

XML Namespaces

There can be conflicts in element names

The meaning of a tag might mean something else

- Homonyms – band vs. band
- Context : phone vs. phone



Like other technologies we have seen

Use namespaces!

```xml
<band>
    <name>The Smashing Pumpkins</name>
    <genre>Alternative rock</genre>
</band>
```

```xml
<band>
    <name>Ktunaxa</name>
    <population>1536</population>
    <regions>
        <region>
            <country>United States</country>
            <province>Idaho</province>
            <province>Montana</province>
        </region>
        <region>
            <country>Canada</country>
            <province>British Columbia</province>
        </region>
    </regions>
</band>
```

XML Namespaces

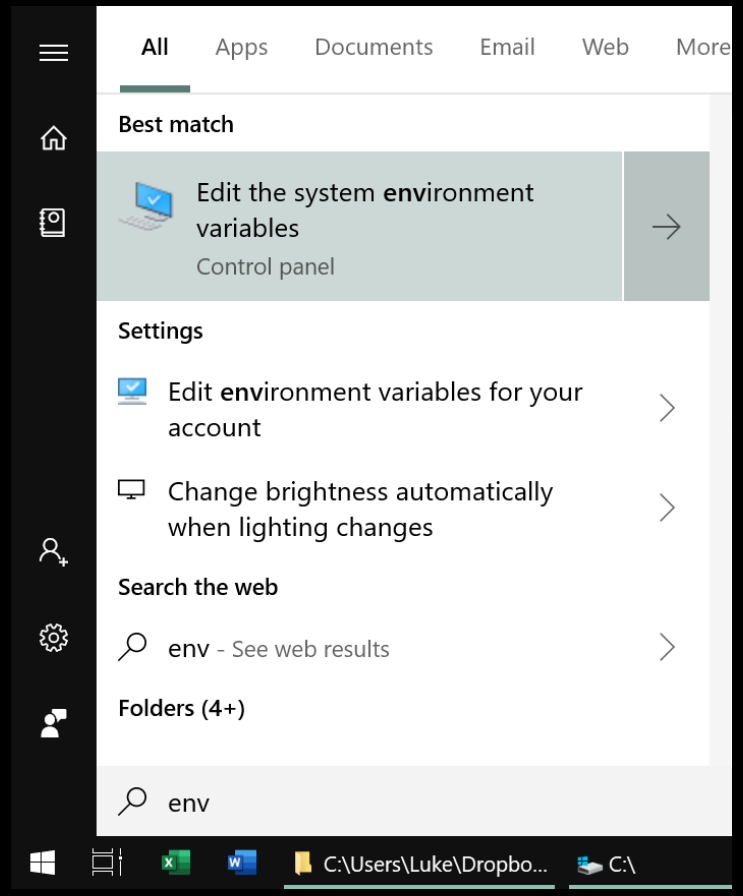
- To use a namespace
  - We specify a namespace attribute of a parent element, (usually on root)
  - we prefix our tags with the namespace
  - Name spaces are given a URI
    - If you are "creating" - should be a domain, you control/own
    - **No technical limitation**, but the idea is that the URI uniquely defines the namespace in the document.

```
<bands xmlns:mb="https://www.music.com/bands">
    <mb:band>
        <mb:name>The Smashing Pumpkins</mb:name>
        <mb:genre>Alternative rock</mb:genre>
    </mb:band>
<mb:bands>
```

XML Schema

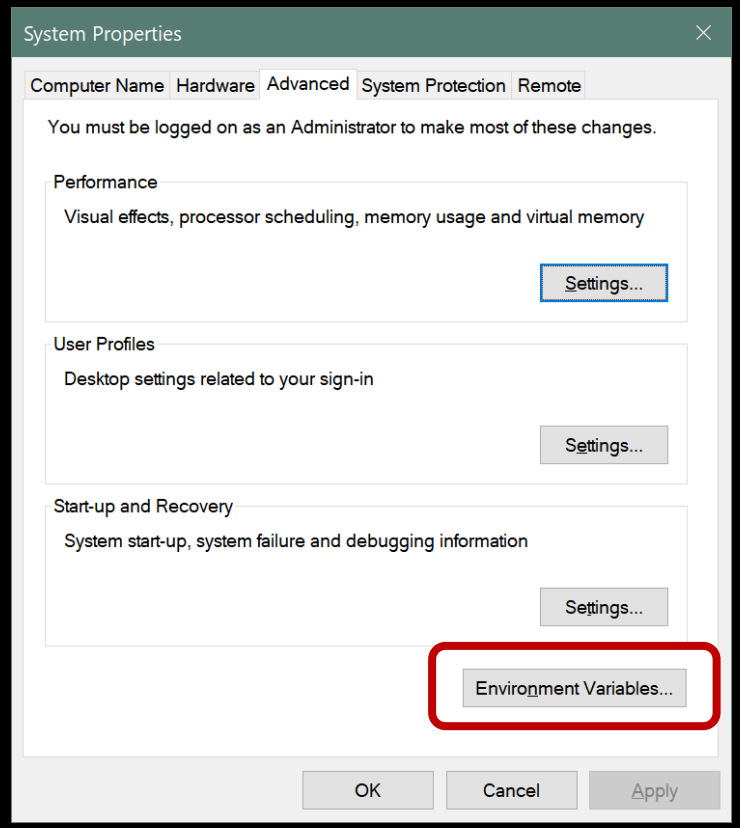
An XML Schema describes the structure of an XML document

VS Code

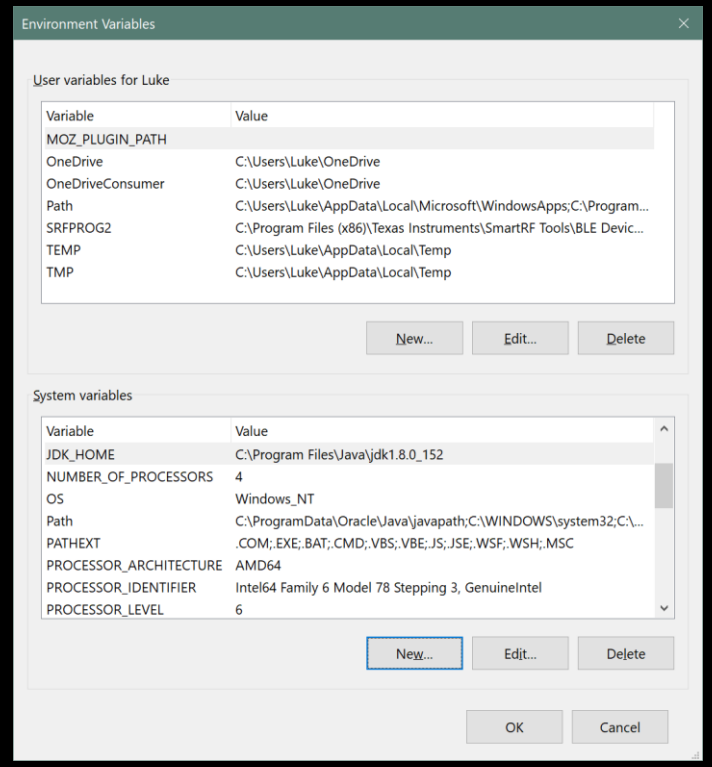


- We want a decent XML extension
- We will install XML, by red hat
- Requires JDK
  - Should be installed!
  - Find it: should be: C:\Program Files\Java
    - Copy the jdk path that *contains* bin
  - Make sure you have JDK_HOME path set in environment variables
    - Go Start->Type: "Environment", "Edit the system environment variables"

VS Code



- Click environment variables
  - In system variables (bottom)
  - Look for JDK_HOME
  - If not there add it
    - Variable name is JDK_HOME
    - Value: the path to the jdk folder, eg:
      - C:\Program Files\Java\jdk1.8.0_152



Schema Definition

There are two main ways to create a schema for an XML format

1. Document Type Definition
   - Uses a unique syntax, used for SGML DTDs
   - No data typing
   - Limited enumeration constraints
2. XML Schema Definition
   - Uses XML syntax
   - Can define data types
   - Supports namespaces

Why

- So all ends of communication know what can/is included
  - Data stored consistently
- Reduce errors with data formats
  - E.g. Dates

- We can ensure the data is valid and doesn't break the rules

XSD Tags

- A schema <schema> tag is the root tag of any XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="https://www.vehicles.com/vehicles"
    xmlns="https://www.vehicles.com/vehicles"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
        …
</xs:schema>
```

XSD <schema>

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

The 'xs' elements of this document are part of the XMLSchema namespace, schema elements should be prefixed with 'xs'

```
elementFormDefault="qualified"
attributeFormDefault="unqualified"
```

Elements in valid documents need to prefixed

Attributes in valid documents need *not* be prefixed

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:element name="vehicles">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="vehicle" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="year" type="xs:int"></xs:element>
                     <xs:element name="make" type="xs:string"></xs:element>
                  </xs:sequence>
                  <xs:attribute name="model" type="xs:string"></xs:attribute>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

XSD Schema Elements

- Element *https://www.w3schools.com/xml/el_element.asp*
  <xs:element>
  - Represents a singular element in a valid document
    - No children OR attributes
  - Attributes
    - name: tag name
    - type: XSD type, simpleType or complexType
      - xs:string
      - xs:decimal
      - xs:integer
      - xs:boolean
      - xs:date
      - xs:time
    - maxOccurs/minOccurs: the maximum/minimum number of times the element should be included here
    - default: value if none specified

```
<xs:element name="vehicle" maxOccurs="unbounded">
      …
</xs:element>
```

XSD Schema Elements

- Attributes *https://www.w3schools.com/xml/schema_simple_attributes.asp*

  <xs:attribute>
  - Represents a singular attribute on a tag
  - Similar structure to <xs:element>
  - Attributes
    - name: attribute name
    - type: XSD type, simpleType or complexType
      - xs:string
      - xs:decimal
      - xs:integer
      - xs:boolean
      - xs:date
      - xs:time
    - use: "required" – attribute is require

```
<xs:attribute name="model" type="xs:string"></xs:attribute>
```

XSD Schema Elements

- Restrictions *https://www.w3schools.com/xml/schema_facets.asp*
- <xs:restriction>
- Restricts allowed values
  - Specifies base type: *base* attribute
  - Have child elements with restriction types and parameters
    - Ranges:

      <xs:minInclusive value="0"/>

      <xs:maxInclusive value="365"/>
    - Enumerations:

      <xs:enumeration value="Ford"/>

      <xs:enumeration value="Toyota"/>
    - Patterns (similar to regex)

      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>

```
<xs:element name="year" type="xs:int">
   <xs:simpleType>
      <xs:restriction base="xs:integer">
         <xs:minInclusive value="1940" />
         <xs:maxInclusive value="2050" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

```
<xs:element name="make" type="xs:string">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:enumeration value="Audi" />
         <xs:enumeration value="Ford" />
         <xs:enumeration value="Toyota" />
         <xs:enumeration value="Golf" />
         <xs:enumeration value="BMW" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

```
<xs:element name="postalCode" type="xs:string">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:pattern value="[a-zA-Z][0-9][a-zA-Z] [0-9][a-zA-Z][0-9]"></xs:pattern>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Complex vs. Simple Types

Remember these are definitions of allowed tags	Complex Types	Simple Types
What is allowed in a valid document Describe allowed tags	Can contain other elements Can't have restrictions	Represent a single element/value Can have restrictions Cannot have attributes

Reminder

Simple Type

```
<xs:element name="age">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0" />
            <xs:maxInclusive value="100" />
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Complex Type

```
<xs:element name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- Multiple child elements must be placed inside an "order indicator" tag
  - Order indicator tags are:
    - sequence : elements must be in the specified order
    - choice : choice of the specified elements
    - all: elements can appear in any order
      - minOccurs and maxOccurs can restrict number of times

Example
all *order indicator with occurrence indicators*

```
<xs:element name="person">
   <xs:complexType>
      <xs:all>
         <xs:element name="firstname" type="xs:string" minOccurs="1" maxOccurs="1" />
         <xs:element name="lastname" type="xs:string" minOccurs="1" maxOccurs="1" />
         <xs:element name="childname" type="xs:string" minOccurs="0" maxOccurs="20" />
      </xs:all>
   </xs:complexType>
</xs:element>
```

Break

X22
Day 2


SPIDERS ARE THE
ONLY WEB DEVELOPERS
THAT ARE HAPPY TO FIND BUG

Heritage college 2020
Lucas stephenson

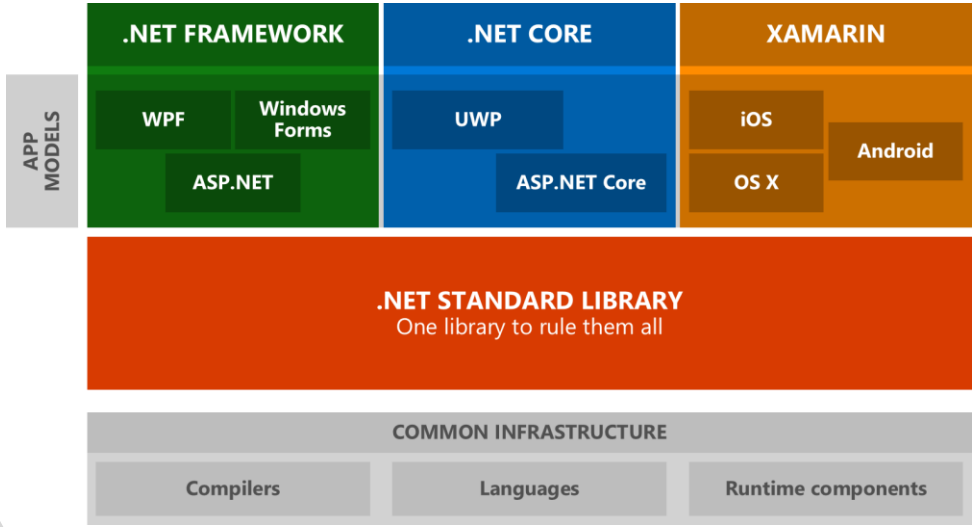
lstephenson@cegep-heritage.qc.ca

Advanced *Web Applications* and *Web Services*

Intro to .NET Core

.NET Framework Implementations:

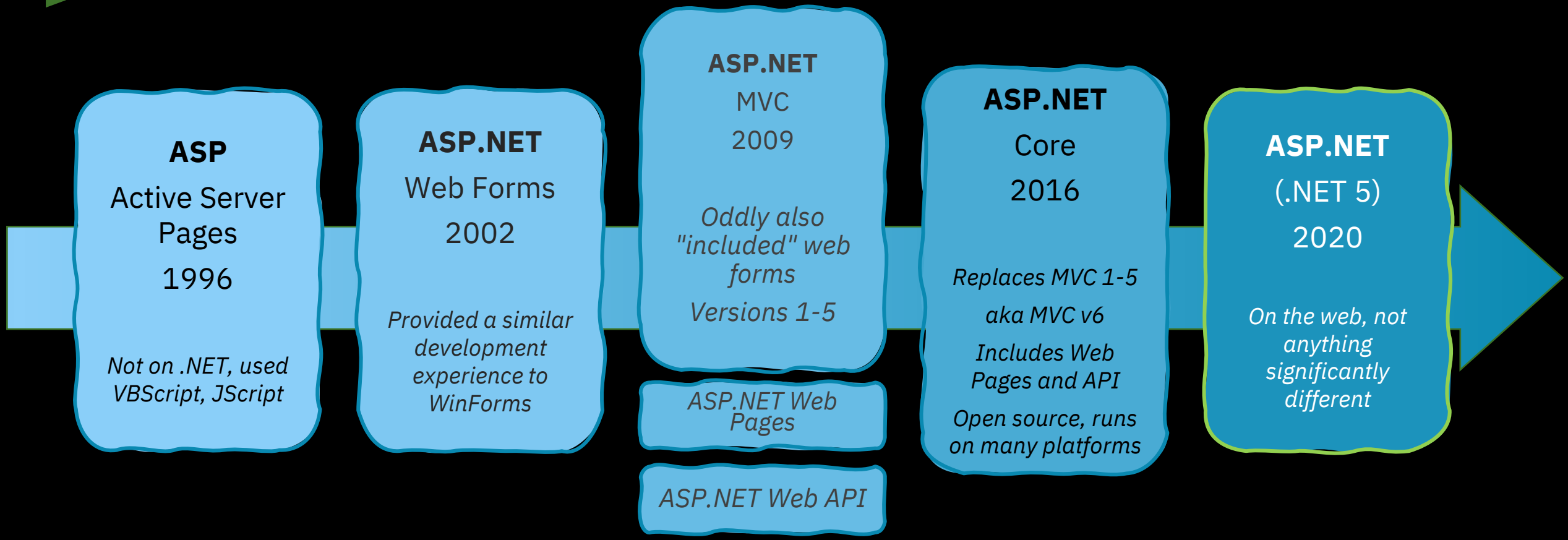
- .NET Framework
  - Original .NET runtime (~2001)
  - **Windows only**
  - Most evolved, best support in Visual Studio
- Xamarin/Mono
  - Open source version of .NET framework (meant to be feature compatible)
  - Started by Ximian, Novell
  - **Runs on many platforms**, windows, linux, macOS, iPhone, Android, etc.
  - Not really supported in Visual Studio
  - Acquired by Microsoft
- .NET Core
  - Started by Microsoft
  - Reuse and Rewrite and open source parts of .NET framework that don't rely on windows



.NET 5

- No "Core" or "Framework"
  - Released scheduled for November
  - No official support until .NET 6
  - **Unify .NET Core and .NET Framework**

.NET for the Web



.NET Core

- (right now) Supports fewer project types than .NET framework, **but more platforms!**
- Open source!
  - You can see the source code for all of .NET core
    - And .NET 5
  - .NET Core apps will run (on windows) with am updated version of .NET framework

What for?

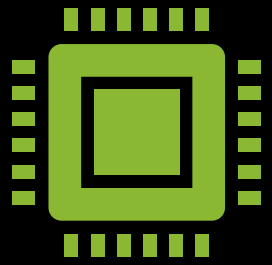
- Meant for server (back-end) applications
  - Web Apps
  - APIs
    - REST
    - GraphQL
    - Custom
  - Leverage other .NET tools:
    - Like Machine learning: ML.NET

    https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet
  - Faster than .NET framework MVC
    - By a lot

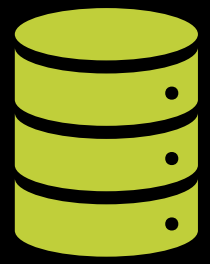
Support for SPA

React, Angular, Vue, etc.

Runs on a variety of platforms...



Has a built-in web server "Kestrel"*



Having it built-in, makes deployment easier

Don't need to explicitly install one

Like express on NodeJS

We can configure the server entirely from our application

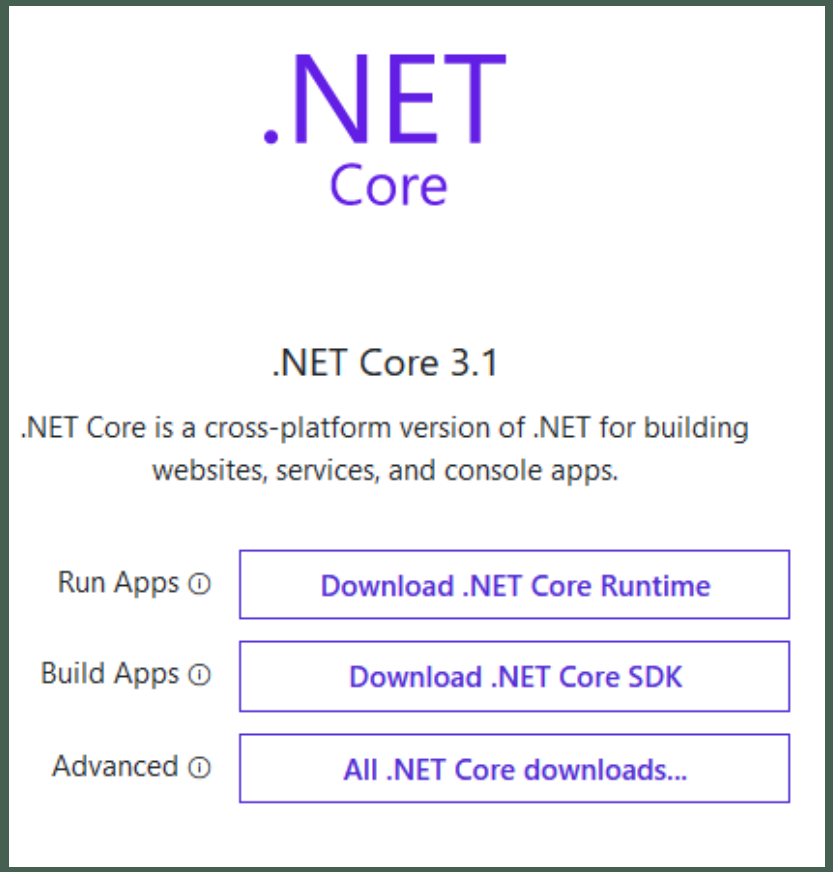
**Can run Kestral directly, or inside another web server (apache, IIS, etc, or a Container service, like Kubernetes...*

NuGET
new-get

- Like NPM, but for .NET apps
  - .NET core is organized as NuGET packages
  - ASP.NET features are provided as nuget packages
  - Makes it easier to deploy to headless platforms (Cloud services)

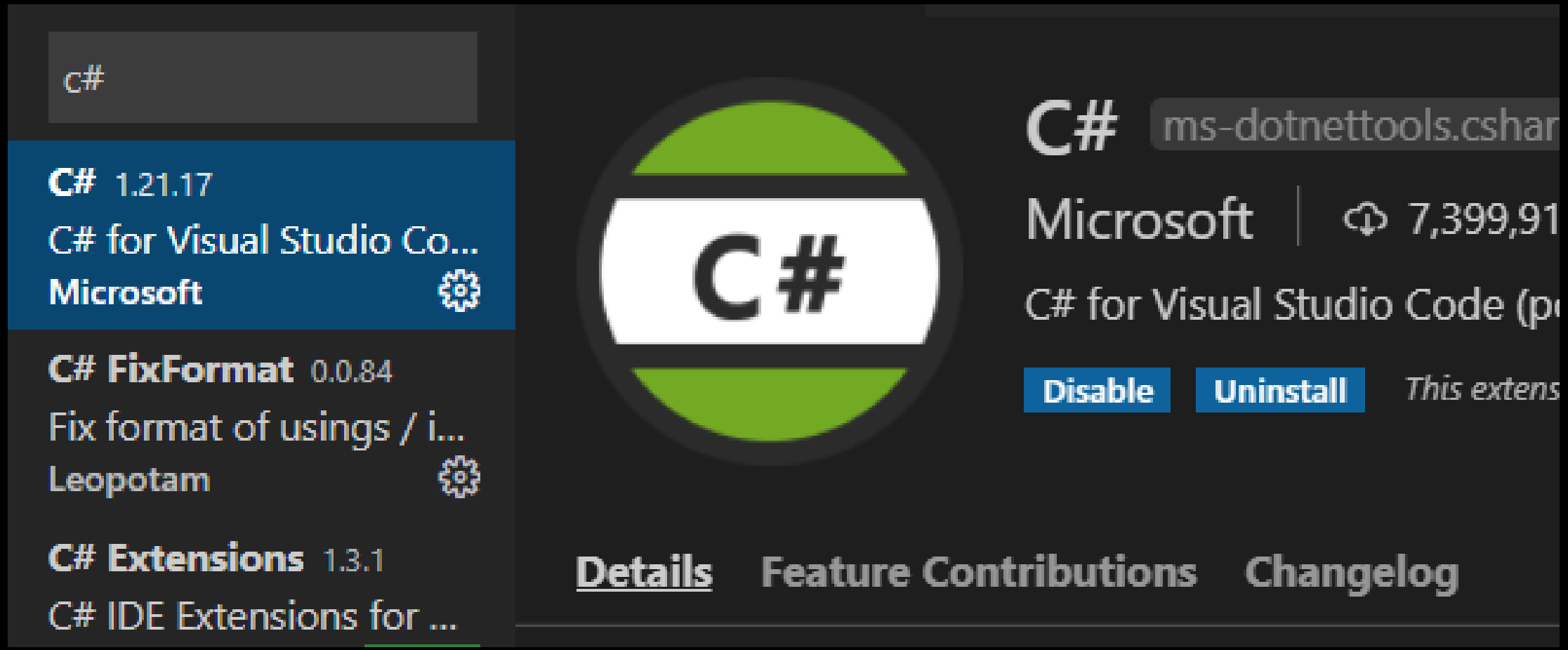
Setup .NET Core 3.1

- https://dotnet.microsoft.com/download



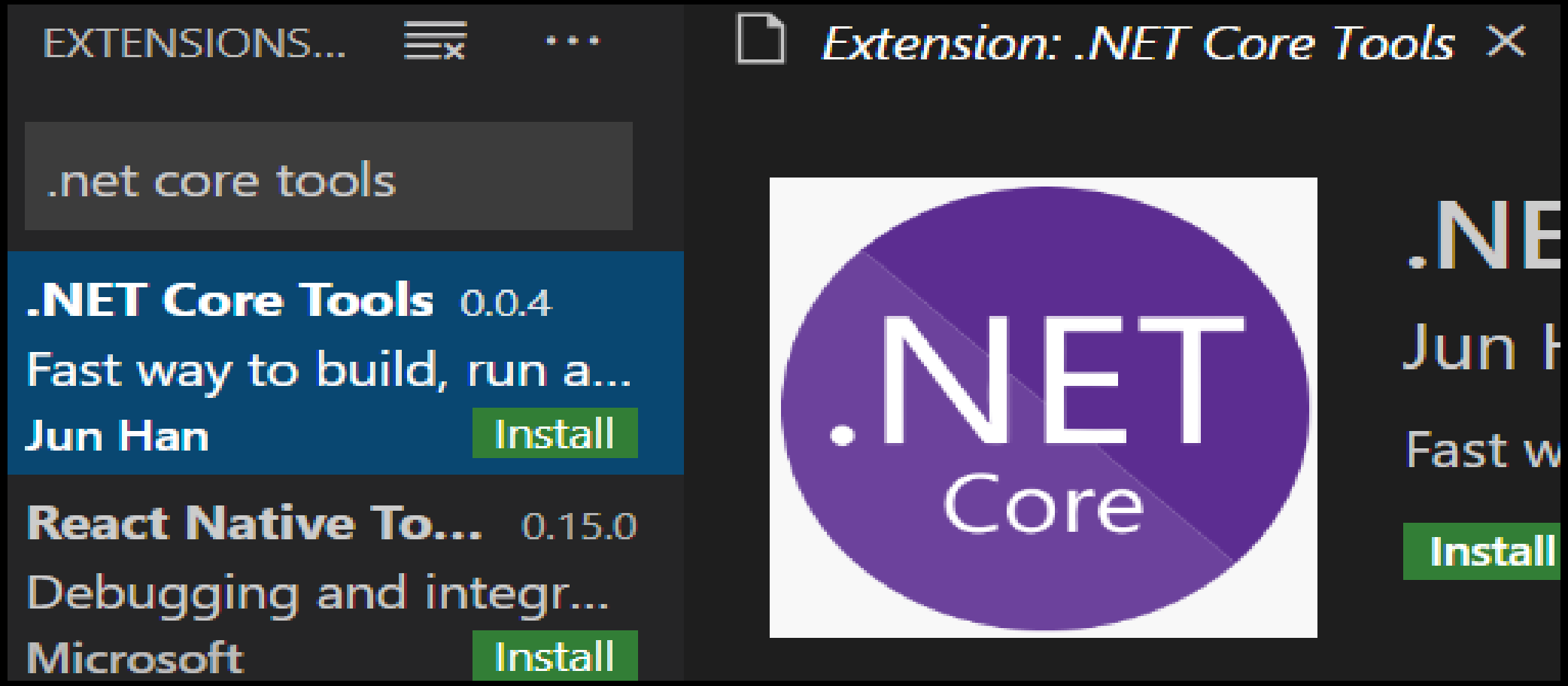
Add the *C#* Extension to Code

Support for C# syntax highlighting and more



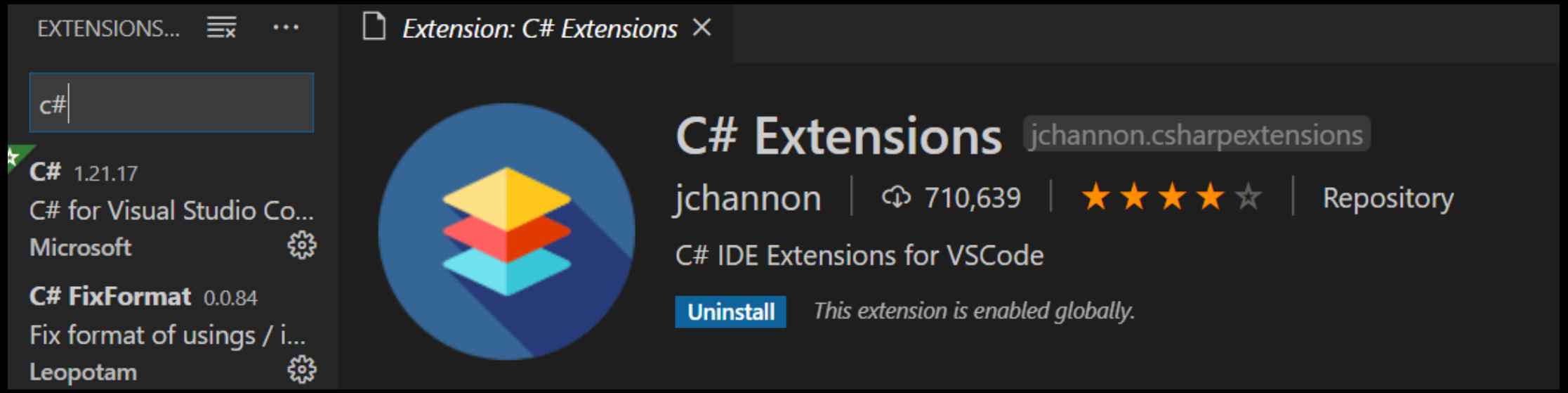
Add the *.NET Core Tools* Extension to Code

Easier access to build/run



Add the *C# Extensions* Extension to Code

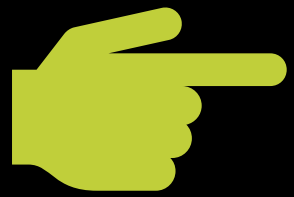
Faster creation of C# classes



See the available templates

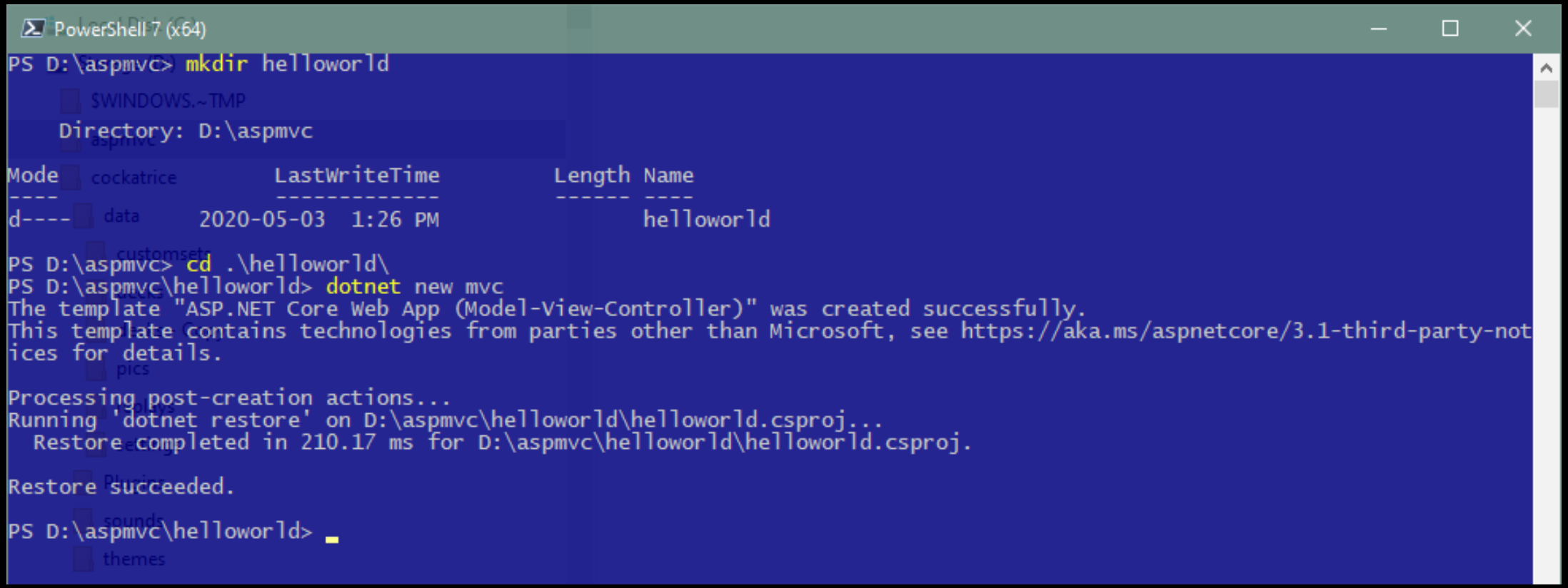
Powershell

```
dotnet new -l
```



You can Install new templates

https://github.com/dotnet/templating/wiki/Available-templates-for-dotnet-new



Choose a folder for your projects

- Open Powershell in that folder
- Make a new folder
  - "helloworld"
  - Change to helloworld
  - Run `dotnet new mvc`

The start

- command-line application
  - Execute the web server (kestrel) which runs
    - The .NET Core MVC application
  - By default application's entry point is
    - *Startup* class in "*Program.cs*" `public static void main`

Break

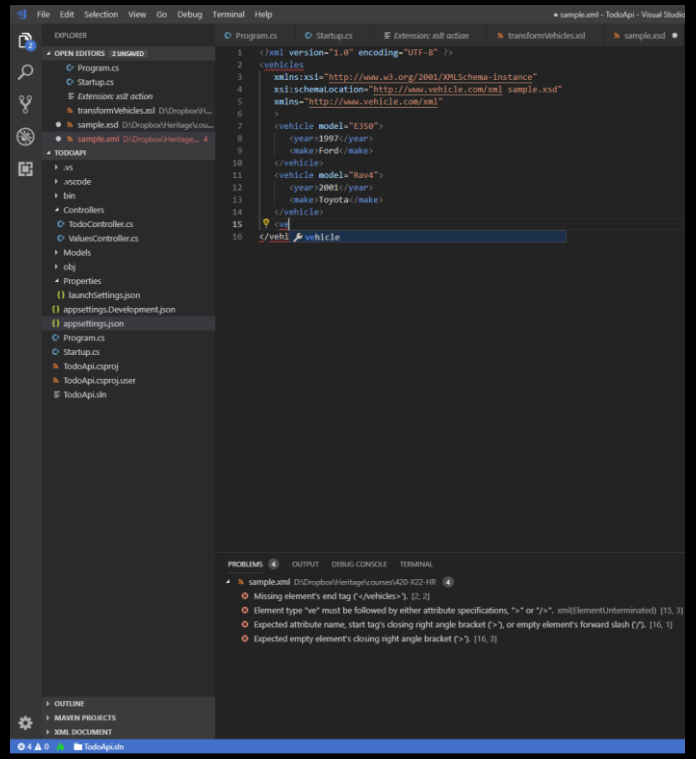
Applying XSD

Add `noNamespaceSchemaLocation` to document/parent to validate against xsd

- You can use children if it only covers a section of your document

```
<vehicles xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="vehicles.xsd">
```

Applying XSD



- Benefit from auto-complete


SPIDERS ARE THE
ONLY WEB DEVELOPERS
THAT ARE HAPPY TO FIND BUGS
imgflip.com

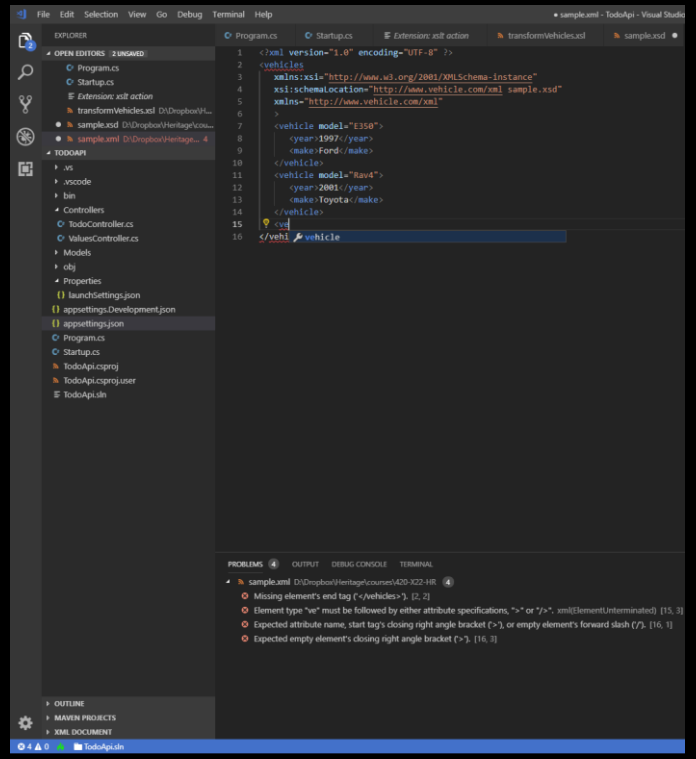
Applying XSD

Add `noNamespaceSchemaLocation` to document/parent to validate against xsd

- You can use children if it only covers a section of your document

```
<vehicles xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="vehicles.xsd">
```

Applying XSD



- Benefit from auto-complete

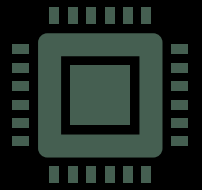
XSLT

eXtensible Stylesheet Language Transformations

Allows the transformation of an XML document, to another document

Source document *must be* XML

Target document can be any text format



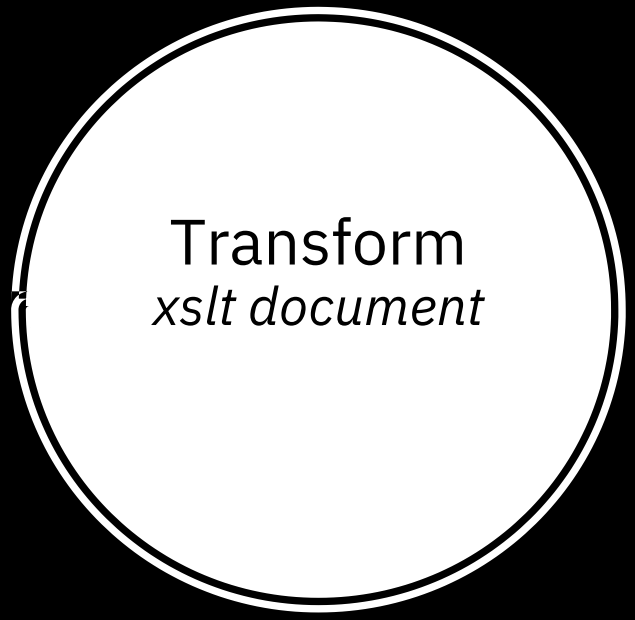
XSLT is a programming (logic) language

With XML syntax

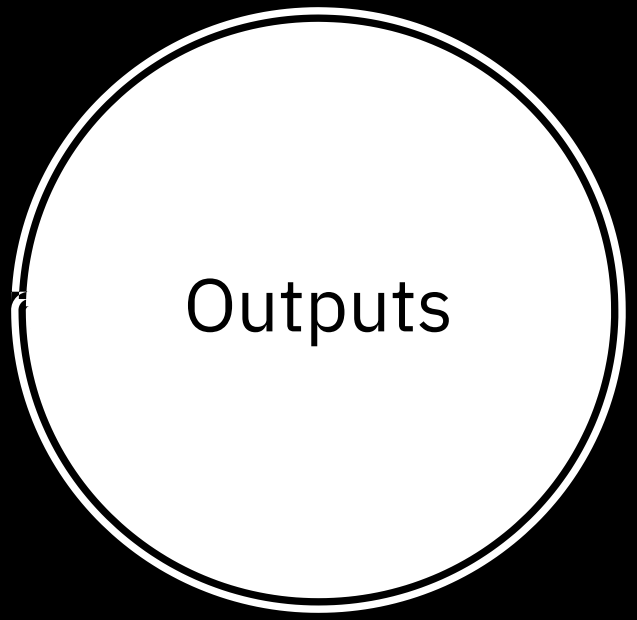
Given

xml document

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="vehiclesTransform.xsl"?>
<vehicles xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.vehicle.com/xml vehicles.xsd"
   xmlns="http://www.vehicle.com/xml">
   <vehicle model="Taurus">
      <year>2020</year>
      <make>Ford</make>
   </vehicle>
   <vehicle model="Taurus">
      <year>2020</year>
      <make>Mercedes</make>
   </vehicle>
</vehicles>
```




```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:v="http://www.vehicle.com/xml">
<xsl:template match="/">
<html>
<body>
<h2>Cars in Stock</h2>
<xsl:for-each select="v:vehicles/v:vehicle">
   <div>
   <div>Year: <xsl:value-of select="v:year"/></div>
      <div>Model: <xsl:value-of select="@model"/></div>
      <div>Make: <xsl:value-of select="v:make"/></div>
    </div>
    <hr />
</xsl:for-each>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```




```
<html xmlns:v="http://www.vehicle.com/xml">
   <head></head>
   <body>
      <h2>Cars in Stock</h2>
      <div>
         <div>Year: 1997</div>
         <div>Model: E350</div>
         <div>Make: Ford</div>
      </div>
      <hr />
      <div>
         <div>Year: 2001</div>
         <div>Model: Rav4</div>
         <div>Make: Toyota</div>
      </div>
      <hr />
      <div>
         <div>Year: 1940</div>
         <div>Model: Jeep</div>
         <div>Make: Audi</div>
      </div>
      <hr />
   </body>
</html>
```

The Transform

- Root element is

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:v="http://www.vehicle.com/xml">
```

Specify the XSL Transform

- In the source XML file

```
<?xml-stylesheet type="text/xsl" href="vehiclesTransform.xsl"?>
<vehicles xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
   xsi:schemaLocation="http://www.vehicle.com/xml vehicles.xsd"
   xmlns="http://www.vehicle.com/xml">
```

<xsl: template>

- Allows you to match on an XML Path (using XPath)
  - To match the whole document we can just specify match="/"
  - Inside is the template that will receive the data

```
<xsl:template match="/">
...
</xsl:template>
```

Output

- <xsl:value-of />
  - Specify which through XPath *https://www.w3schools.com/xml/xpath_intro.asp*
    - XPath works similarly to urls/file system paths, but is used to indicate elements of an XML document

```
<xsl:value-of select="vehicles/vehicle/year"/>
```

    - Will return the first "year" value, that is a child of vehicle, that is a child of vehicles
    - Note: this *must* be inside your template!
  - For an *attribute* (model is an attribute on vehicle)

```
<xsl:value-of select="vehicles/vehicle/@model"/>
```

<xsl:for-each>

- Allows looping over children
- The following will loop over each vehicle in the source

```
<xsl:for-each select="v:vehicles/v:vehicle">
```

- Note: it also sets the relative XPath to `v:vehicles/v:vehicle`
  - Therefore, to output year (inside for-each):

```
<xsl:value-of select="v:year"/>
```

<xsl:sort>

https://www.w3schools.com/xml/ref_xsl_el_sort.asp

- We can sort the items in a for-each

```
<xsl:for-each select="v:vehicles/v:vehicle">
    <xsl:sort select="v:year"/>

</xsl:for-each>
```

```
<xsl:for-each select="v:vehicles/v:vehicle">
    <xsl:sort select="v:year" order="descending" />

   </xsl:for-each>
```

<xsl:if>

- Allows conditional elements

```
<xsl:for-each select="v:vehicles/v:vehicle">
   <xsl:if test="v:year &gt; 2000">
      <div>Year: <xsl:value-of select="v:year"/></div>
   </xsl:if>
</xsl:for-each>
```

- We can't use greater than or less than symbols directly in XML, instead:
  - >
  - <
  - '=' is ok

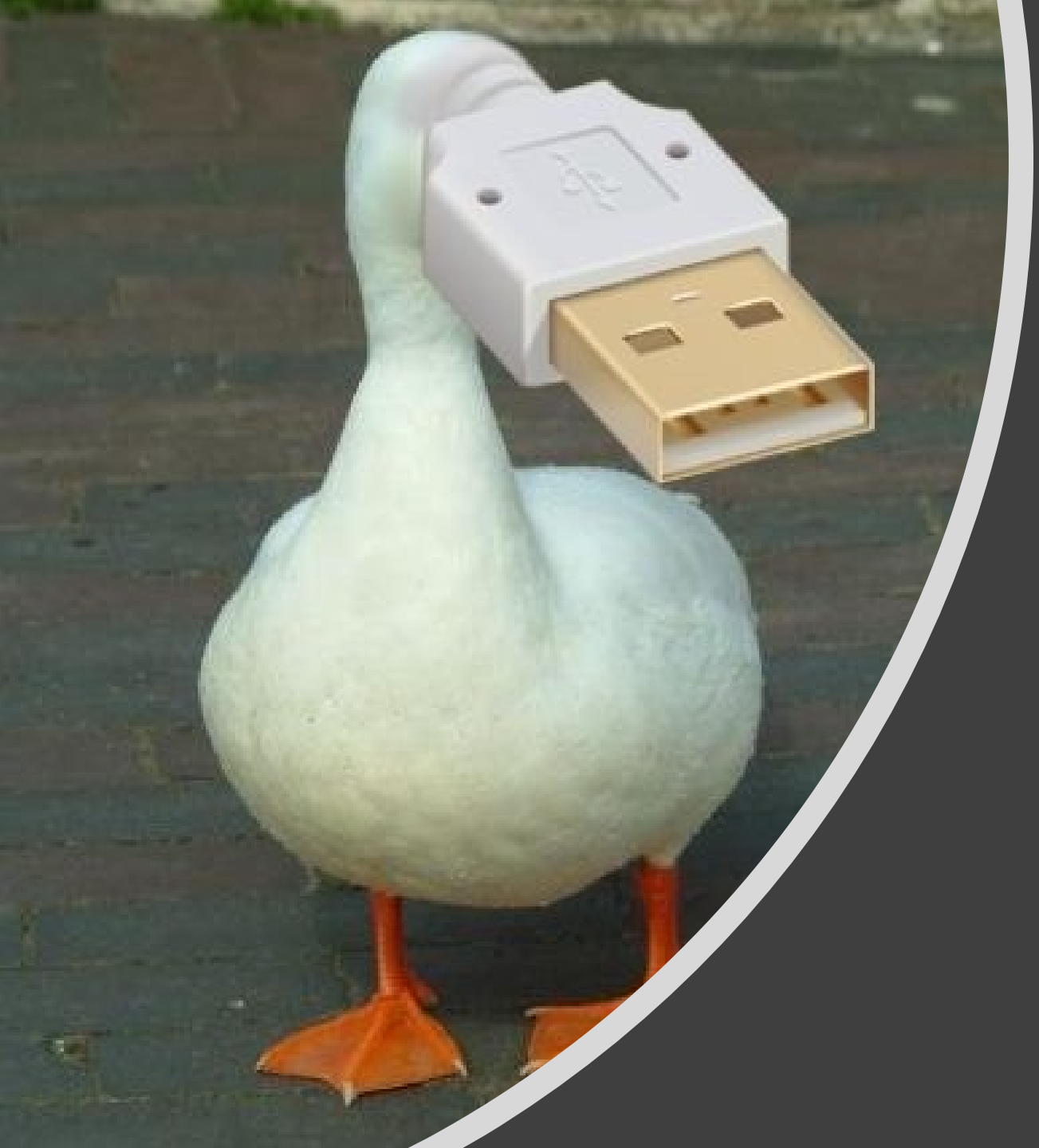
<xsl:choose>

- To make if/else logic
  - Contains
    - <xsl:when>
    - <xsl:otherwise>

```
<xsl:for-each select="v:vehicles/v:vehicle">
   <xsl:sort select="v:year" order="descending" />
   <div>
      <xsl:choose>
         <xsl:when test="v:year &gt; 2000">
            New
         </xsl:when>
         <xsl:otherwise>
            Old
         </xsl:otherwise>
      </xsl:choose>
      <div>
         Year:
         <xsl:value-of select="v:year"/>
      </div>
      <div>
         Model:
         <xsl:value-of select="@model"/>
      </div>
      <div>
         Make:
         <xsl:value-of select="v:make"/>
      </div>
   </div>
   <hr />
</xsl:for-each>
```

If Interested

- Transforms can be used like view/template engines
- You can also create non-xml documents
  - https://www.w3schools.com/xml/xsl_elementref.asp



New (maybe) Prerequisites

- MariaDB – install via XAMPP
  - https://www.apachefriends.org/index.html
- PostMan
  - https://www.postman.com/downloads/

Advanced *Web Applications* and *Web Services*

Day 3

Heritage college

Lucas stephenson
lstephenson@cegep-heritage.qc.ca

HTTP Methods

- Defined methods or actions in http
  - There is no enforcement of their meaning
    - Developers can potentially use them all, however they want
      - GET and POST are most common
  - Most web browsers only support GET/POST for user browsing
    - But fetch/XMLHttpRequest can specify a method to use

HTTP Methods

GET

- Asks for the specified resource
- Requests using GET should only *retrieve* data.

POST

- Submit data to the specified resource
- Can change state the server.

HTTP Methods

PUT

- Asks to replace the data at the resource with the request data.

HEAD

- Requests the same as GET, but without the response data.

DELETE

- Requests the deletion of the specified resource.

HTTP Methods

OPTIONS

- Is used to describe (http method) options for the URL.

PATCH

- Requests a partial modification to the resource.

HTTP Methods

Two More

- *But don't use them for web-dev*

CONNECT

- Used to for http proxies
- asks to connect to another server through the http proxy

TRACE

- Commonly implemented as a "ping" – a message loopback request

APIs

- API
  - A layer or interface that allows applications to "talk" to another application
    - No people are involved
- Web API
  - A layer that works across the internet, *most often* using the HTTP protocol
    - No webpage/site/browser/person needs to be directly involved
  - Web API and Web Service are essentially synonymous

Styles of Web API
Remote Procedure Call (RPC)

- This is the most natural way (*for most people*)
- Client asks server to do something
  - If needed, server responds with results/data
- Standards exists, but aren't extensively adhered to these days
  - https://www.jsonrpc.org/
- Simple Object Access Protocol (SOAP)
  - Was popular in the early to mid 2000s
- XML-RPC
  - Is slightly older, and like SOAP, less featureful, more streamlined
- This is what was used until REST became popular

Sample SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        <ns1:RequestHeader
            soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
            soapenv:mustUnderstand="0"
            xmlns:ns1="https://www.google.com/apis/ads/publisher/v201905">
            <ns1:networkCode>123456</ns1:networkCode>
            <ns1:applicationName>DfpApi-Java-2.1.0-dfp_test</ns1:applicationName>
        </ns1:RequestHeader>
    </soapenv:Header>
    <soapenv:Body>
        <getAdUnitsByStatement xmlns="https://www.google.com/apis/ads/publisher/v201905">
            <filterStatement>
                <query>WHERE parentId IS NULL LIMIT 500</query>
            </filterStatement>
        </getAdUnitsByStatement>
    </soapenv:Body>
</soapenv:Envelope>
```

Client-Side Requirements

- The popularity of these schemes fell off (most likely)
  - Because AJAX became popular
  - Goal became to minimize the **client-side work**

Styles of Web API *REpresentational State Transfer(REST)*

Large focus on data and entities

Less on immediate functionality

Hypermedia as the Engine of Application State (HATEOS)

Responses should include links to related resources

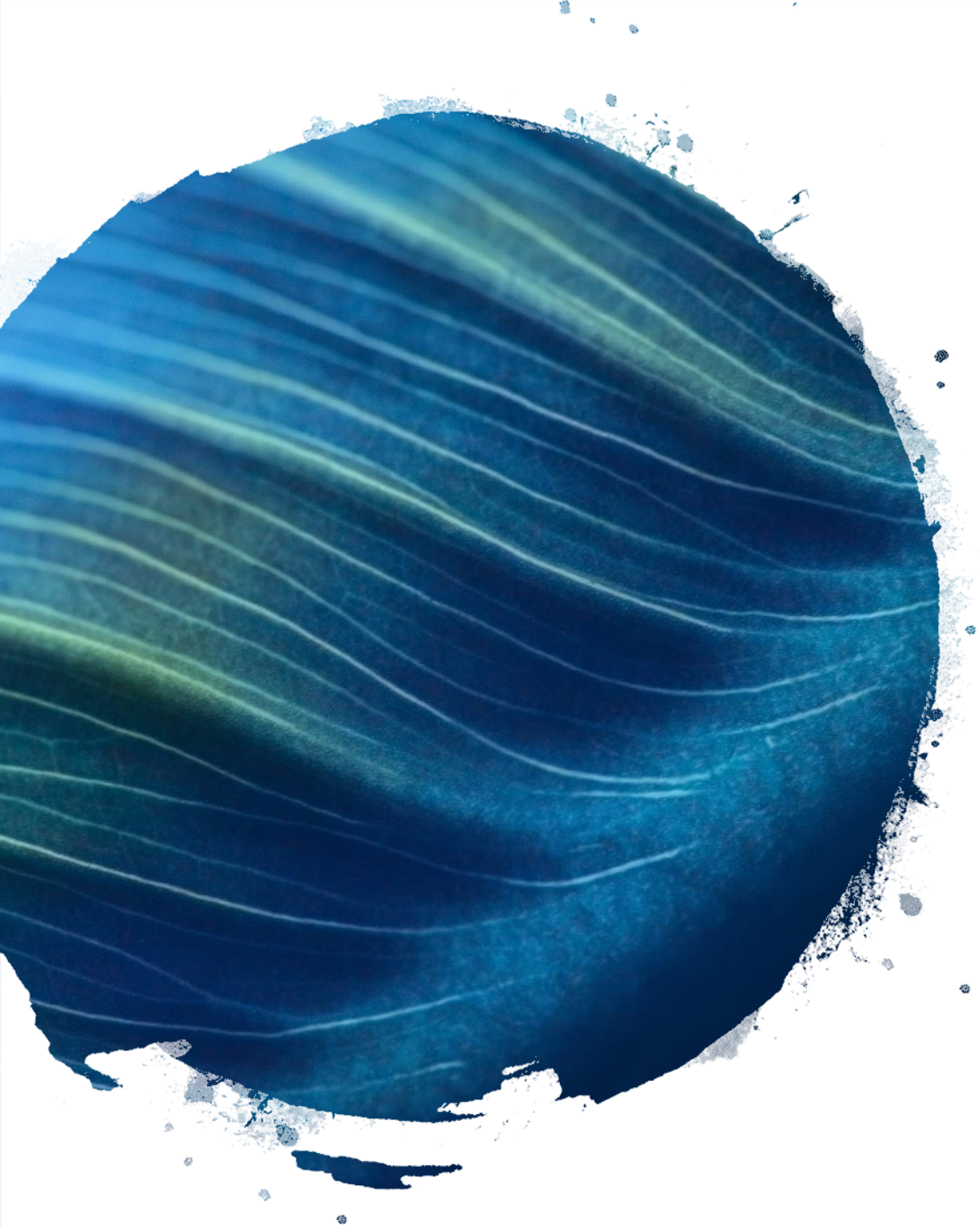
•(like links on a standard html page)

Intended to allow *navigating* the API without the documentation possible

A nice feature, but not fully utilized most of time

Purists will say it is not RESTful if it doesn't HATEOAS

“Folks everywhere are building RESTish APIs which are basically just RPC + HTTP verbs + Pretty URLs.”



CRUD
Create, Read, Update, Delete

When many developers (and employers) refer to REST they are talking about mapping CRUD to HTTP verbs.

https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

There are common ways to map HTTP verbs to REST services

Most Common Mapping

METHOD	ACTION	GENERAL RESPONSE (X/)	SPECIFIC RESPONSE (X/ID)
POST	Create	• Creation of new item, return of item, or link to item • 201 Created	
GET	Read	• Return list of resource • 200 OK	Return single item 200 OK
PUT	Update/Replace		204 No Content 404 Not found if id doesn't exist
PATCH	Update/Modify		204 No Content 404 Not found if id doesn't exist
DELETE	Delete		204 No Content 404 Not found if id doesn't exist

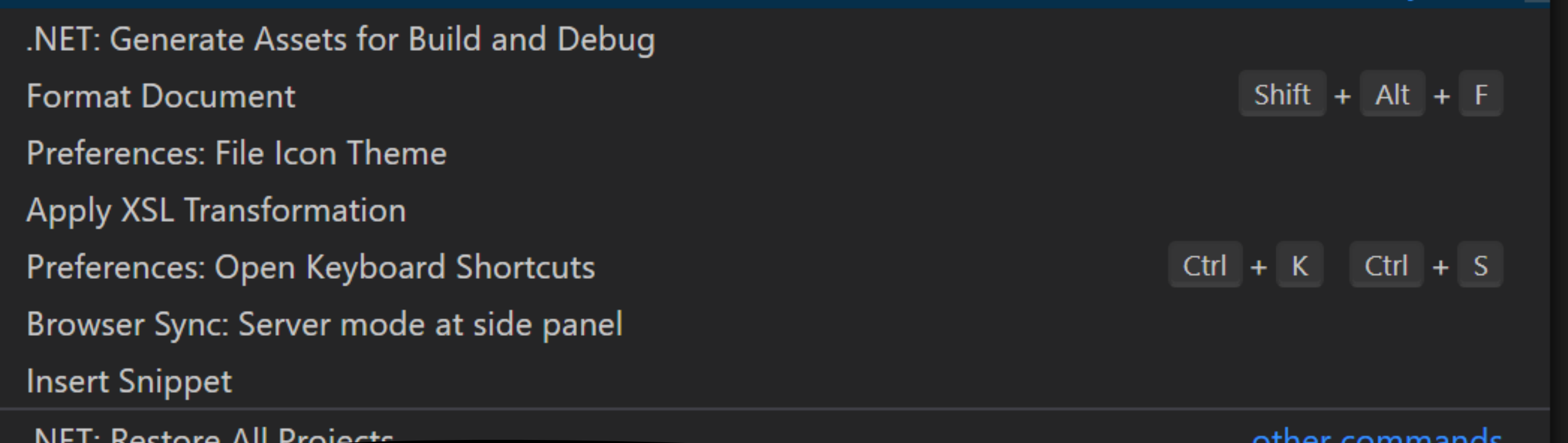
Break

Todo API

- We are going to use mariadb and .NET core 3.1 to make a Todo Webservice
  - This is a web **service** that will allow tracking of tasks we need to do
- We will create a *client* later

Setup

- In your project folder
  `dotnet new webapi -o TodoApi`
  → Creates a new .NET Core project based on the "webapi" template, in the "TodoAPI" subfolder
  `cd .\TodoApi\`
  → Switches the working directory
  `dotnet add package Microsoft.EntityFrameworkCore.SqlServer`
  → Install support for sql server, and some other DB Context functionality
  `dotnet add package Pomelo.EntityFrameworkCore.MySql`
  → Adds a nuget package to support saving .NET Entity Framework objects to mysql (and mariadb) databases
  `dotnet add package Microsoft.EntityFrameworkCore.InMemory`
  → Adds a nuget package to support saving .NET Entity Framework objects to memory

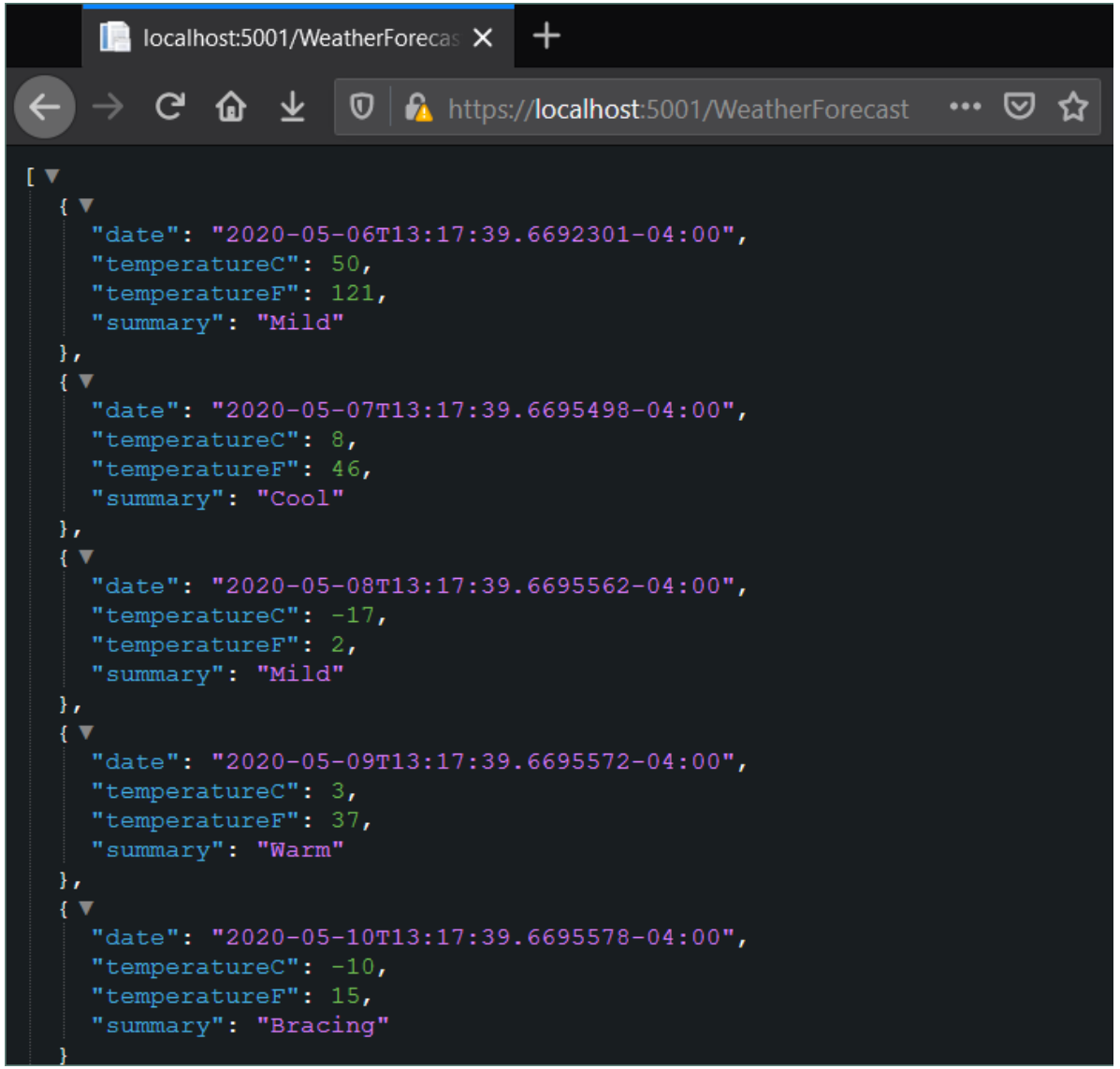


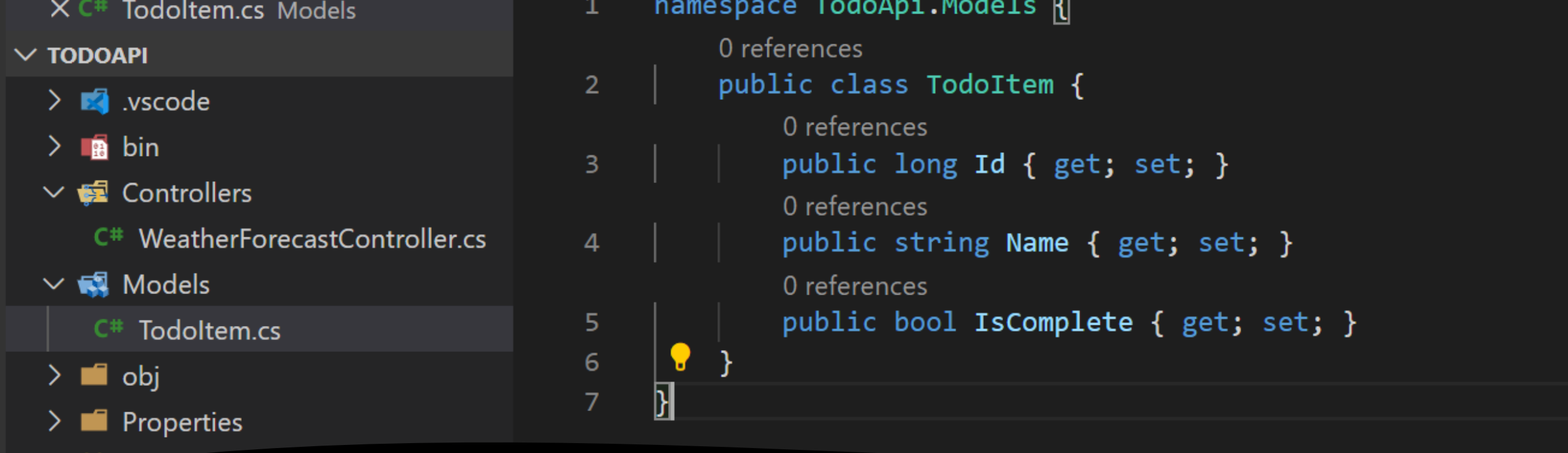
Load in Code

- Need to generate assets
  - Can use pop-up, or use the command palette (CTRL+SHIFT+P)

Run it!

- CTRL+F5, or Run→Start without Debugging
- Goto:
  - https://localhost:5001/WeatherForecast
- Should get some JSON out





Organize!

- The "WeatherForcast" model code data structure is not in a model folder
  - Create a "Models" folder hold our models
  - Add a TodoItem class

Database Context

- A .NET database context is magic, that stores/retrieves data from an underlying storage engine
  - We installed support for mariadb, as well as in-memory
- Add a class "TodoContext" to the models folder

```
using Microsoft.EntityFrameworkCore;

namespace TodoApi.Models {
    public class TodoContext : DbContext {
        public TodoContext (DbContextOptions<TodoContext> options)
               : base (options) { }

        public DbSet<TodoItem> TodoItems { get; set; }
    }
}
```

Magic: EntityFramework

- Microsoft EntityFramework is an "ORM" framework
  - Object Relational Mapping
  - Allows mapping .NET objects to other formats, namely relational types
    - Example: a .NET class to a RDBMS table, or vice versa
- Learn more about the .NET core EF here:
  - https://docs.microsoft.com/en-us/ef/core/

"register" the "Database Context"

- Open Startup.cs
- Remove unused "usings"
- Add *usings* for EntityFrameworkCore and the todo API models namespaces

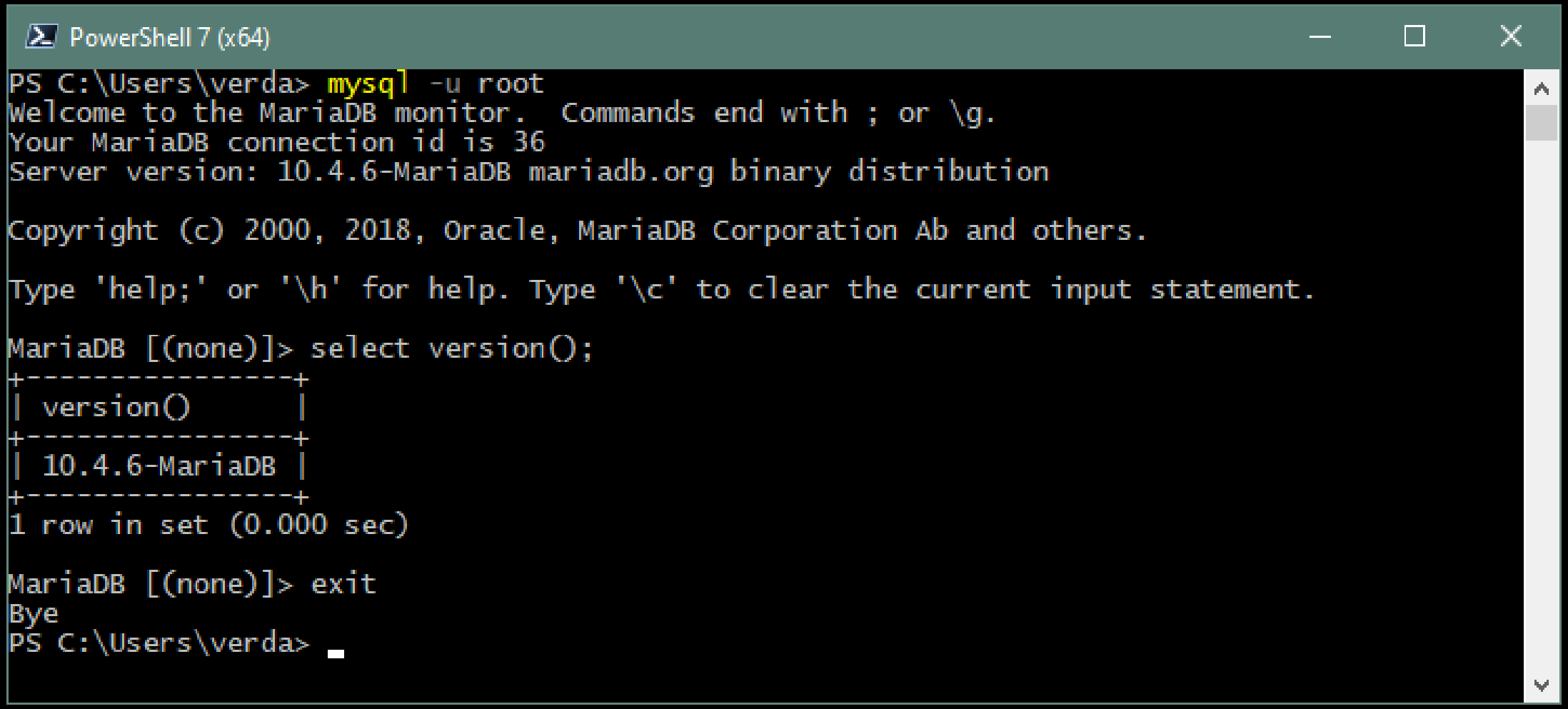
```
using System;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Pomelo.EntityFrameworkCore.MySql.Infrastructure;
using TodoApi.Models;
```

"register" the "Database Context"

- In the "ConfigureServices" method, we want to configure the db context
- *We could an in-memory context here*

```
public void ConfigureServices (IServiceCollection services) {
    services.AddControllers ();
    services.AddDbContextPool<TodoContext> (options => options
        // replace with your connection string
        .UseMySql ("Server=localhost;Database=todo;User=root;", mySqlOptions => mySqlOptions
            // replace with your Server Version and Type
            .ServerVersion (new Version (10, 4, 6), ServerType.MariaDb)
        ));


}
```



MariaDB Version

- To check which version of MariaDB we have installed, we need to query "VERSION()"
  - Can do it with Heidi, or the mysql console app

Generate a Controller

- Use magic
- Add nuget packages
  - Microsoft.VisualStudio.Web.CodeGeneration.Design
  - Microsoft.EntityFrameworkCore.Design
- Install a "tool" that can generate (our controller) code
  - `dotnet tool install --global dotnet-aspnet-codegenerator`
- Run the tool
  - `dotnet aspnet-codegenerator controller -name TodoItemsController -async -api -m TodoItem -dc TodoContext -outDir Controllers`

View the Generated Controller code

- It generated methods that will handle
  - api/TodoItems: **GET** – gets a list api/TodoItems
  - api/TodoItems /*id* **GET**: gets a specific TodoItems
  - api/TodoItems /*id* **PUT**: updates/replaces a specific TodoItems
  - api/TodoItems: **POST** – adds an api/TodoItems
  - api/TodoItems /*id* **DELETE**: removes a specific TodoItems

Change the POST

- return the new TodoItem
  - It already passes the name of the method (GetTodoItem) and runs it
  - Instead, lets make it a bit more robust by passing the specific name of the type, using C#'s the nameof expression

```
[HttpPost]
public async Task<ActionResult<TodoItem>> PostTodoItem (TodoItem todoItem)
{
    _context.TodoItems.Add (todoItem);
    await _context.SaveChangesAsync ();

    return CreatedAtAction (nameof (GetTodoItem),
        new { id = todoItem.Id },
        todoItem);
}
```

Database doesn't exist!

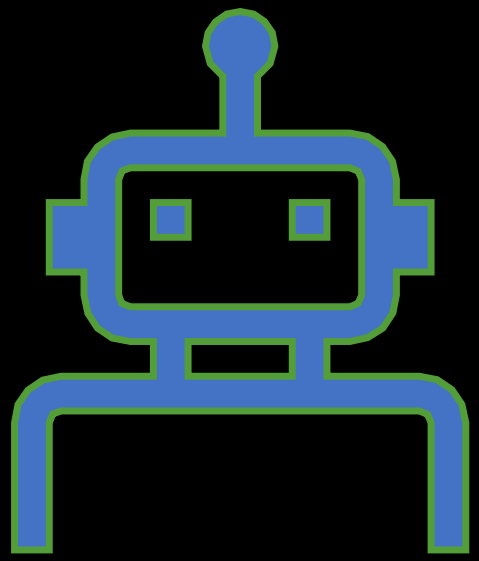
1. We need to create an empty database
   - Create a database, the name you specified in *Startup.cs*
   - *Recommend Heidi!*
2. Create the TodoItem table
   1. This can be done automatically! (Magic)
   2. Create an Entity Framework Migration (in the Terminal)
   3. Run an Entity Framework database update

Commands

```
dotnet ef migrations add dbInit
dotnet ef database update
```

Name of the "migration"
doesn't matter what it is, but no spaces

Try It



- Run it,
  - Use postman and Heidi!

Lab

- Add a deadline (DateTime) field!
- And another field (you choose)
    1. Add to the class (TodoItem)
    2. Create a new migration, update the database
    3. Test

Lab

- Add a deadline (DateTime) field!
- And another field (you choose)
    1. Add to the class (TodoItem)
    2. Create a new migration, update the database
    3. Test

Prerequisites

You will need a recent version of node for the ToDo Client

- https://nodejs.org/en/download/

- There is a node webpack starter project on Teams
  - Files\Examples\ToDoClient.zip

Advanced *Web Applications* and *Web Services*

Day 4

- Heritage college
- Lucas stephenson

lstephenson@cegep-heritage.qc.ca

2020

Important Dates

May 7th	: Today
May 12th	: Midterm
May 13th	: Assignment Instructions
May 18th	: No class (Victoria Day)
May 25th	: Last day (presentations and quiz)

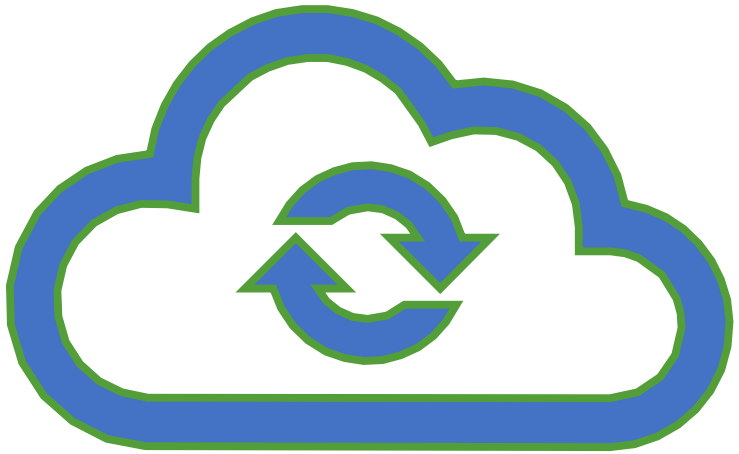
Web Service Architecture

2 Main Components

Server – anything that provides a service

Client – anything that consumes a service

Sounds simple



Services

- We have discussed styles of web service/api
  - RPC
  - REST

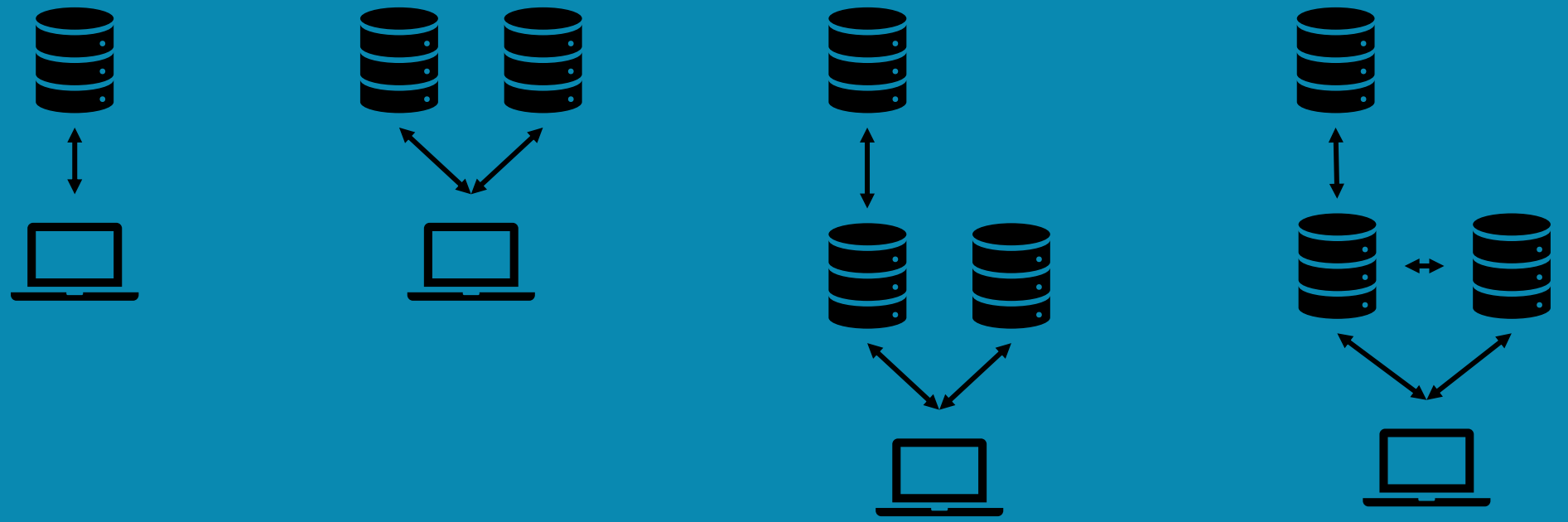
Designs

Client

Server

- Most commonly, on the web clients are isolated from each other, for privacy and security concerns
- This is not always true, peer-2-peer applications are the exception
  - Tor, bittorrent, etc.
- **HTTP is client-server**

Designs

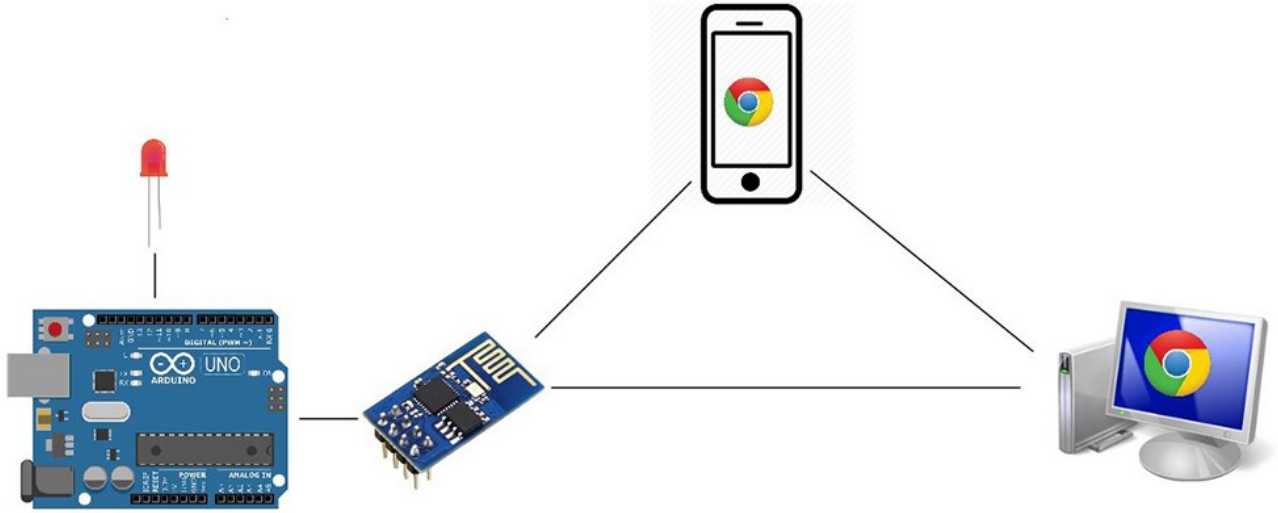


Servers

- Servers can be written in any language, on any platform
  - iot (internet of things) is heavily based on everything providing data
    - Clients can also be servers
  - Important to remember that servers provide 1 or more service

Clients

- In *our* case, typically a web browser
  - But anything connected to the internet
    - Phones
    - Desktop
    - Fridge
    - TV
    - Arduino
    - Anything…
  - Does not need a UI



Cross-Origin Resource Sharing

- CORS
  - Without CORS
    - It is possible for malicious sites to capture data
      - Mainly through cookies
      - If you visit a website, and that website uses AJAX to call https://www.mybankwithallmymoney.com
      - And you are logged in with cookies
      - The response would contain your login cookies
    - Therefore,
      - user-agents that support cookies should not allow in-page requests (AJAX) to off-site addresses

Restrictions

- When the browser AJAX methods are called to a remote resource
  - *A negotiation happens*
  - The browser asks the server
    - If requests from the origin (page/site that you're on) are allowed
      - *https://www.mybankwithallmymoney.com*
      - Says "no" or "yes"
    - If yes – we can request stuff from *https://www.mybankwithallmymoney.com*
    - If no – we can't
    - Older servers, or unconfigured servers don't understand the question
      - Newer web browsers: refuse the send the request

JSONP

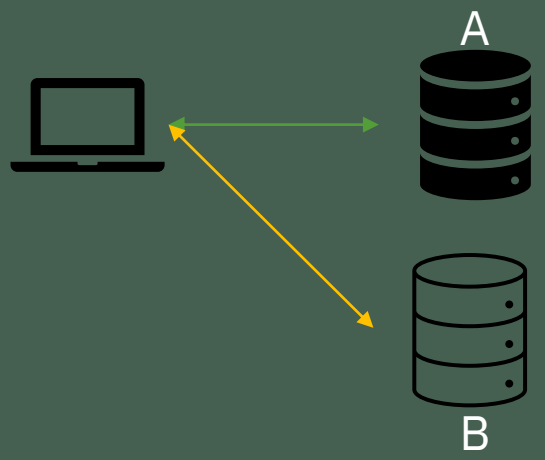
CORS Solution

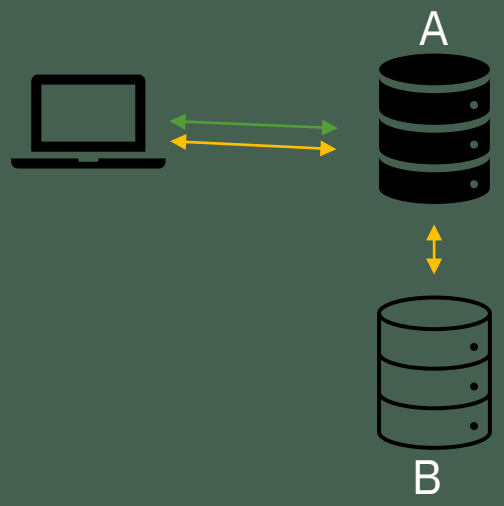
- JSON-Padding
- Scripts that are loaded via the <script> tag are always fetched
- To work around CORS, some implemented their APIs to be loaded via script tags
  - Wrap the (JSON) data in a function, which is called when the .js is fetched
- Scripts were called by dynamically adding <script> tags

- This requires that the service output JSONP rather than JSON
- The client then can run the returned function

Another Solution

- Direct all requests through the origin web server
  - This does not allow a user's remote cookies to be shared with the currently site
  - Since the remote website's cookies are never involved





Problem

User visits web app A

1. The webapp includes a client (JavaScript) fetch to a webservice B
2. Webapp A's code can capture cookies, and session data (via JavaScript) from client's prior connexions to B

Result

To protect users; **browsers** do not allow fetch from different origins, unless the webservice explicitly allows it.

Workaround

Requests for the web service B are routed through A, A doesn't have access to the client's session data with B

Result

Fetch generates extra traffic, must develop server to use and do pass through

If a service does not have any personal or private data, it can set a CORS policy, to allow full access from client fetches

Real Solution

- Web Services (the server) should specify if they allow CORS
  - This is done with http headers
  - User agents ask first, if the server responds with saying it is "ok", the client will send the actual request.

Example using a go Between Service

```
app.get("/getToilets", cors(), async (req, res) => {
    const response = await fetch(
        "http://...download/publicwashrooms.json"
    );
    const data = await response.json();

    res.json(data);
});
```

Other Benefits of Going *Through*

Protect apiKeys

Client never sees them

Cacheing

Store recent results in a database

If same request comes in, send the copy without sending additional remote request

- Probably set an age limit on your record

Break

Create A Web Client

- TodoItem Client
  - Unless we host it on the same server, in order to access our api.
    - need to pass our requests through another web server (blegh!)

    - Or -

    - Implement permissive CORS policy in our web services
  - In Startup.cs, ConfigureServices() add a CORS service, giving full rights from any origin

```
services.AddCors (opt => {
    opt.AddPolicy ("AllowAll",
    builder => builder.AllowAnyOrigin ().AllowAnyMethod ().AllowAnyHeader ());
});
```

Create a Web Client

Allow listing (GET), adding (POST), completing (PATCH) and deleting (DELETE)

Using JavaScript fetch

```
fetch('https://localhost:5001/api/todoitems/');
```

- Reminder: fetch uses GET by default
  - We do want to send an HTTP flag, so the response is not cached
  - A second parameter takes an object with named properties

```
{
    method: 'POST', // *GET, POST, PUT, DELETE, etc.
    mode: 'cors', // no-cors, *cors, same-origin
    cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached
    credentials: 'same-origin', // include, *same-origin, omit
    headers: {
      'Content-Type': 'application/json'
    },
    redirect: 'follow', // manual, *follow, error
    referrerPolicy: 'no-referrer', // no-referrer, *no-referrer-when-downgrade,
          origin, origin-when-cross-origin, same-origin, strict-origin,
          strict-origin-when-cross-origin, unsafe-url
    body: JSON.stringify(data) // body data type must match "Content-Type" header
}
```

Fetch Options

List - GET

```
fetch('https://localhost:5001/api/todoitems/',
{
   cache: 'no-cache',
   method: 'GET',
});
```

Add - POST

```
fetch('https://localhost:5001/api/todoitems', {
   method: 'POST',
   headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
   },
   body: JSON.stringify(data),
});
```

Remove - DELETE

```
fetch('https://localhost:5001/api/todoitems/' +
 id, {
   method: 'DELETE',
   headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
   },
});
```

Edit – PATCH*

```
fetch('https://localhost:5001/api/todoitems/' + id, {
   method: 'PATCH',
   headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
   },
   body: JSON.stringify({
      id: id,
      isComplete: true,
   }),
});
```

*Note that we do not have a PATCH implementation in our web service

Implement PATCH

set IsComplete: true or false

```
[HttpPatch("{id}")]
public async Task<IActionResult> PatchTodoItem(long id, TodoItem it
em)
{
    if (id != item.Id)
    {
        return BadRequest();
    }

    var todoItem = await _context.TodoItems.FindAsync(id);
    if (todoItem == null)
    {
        return NotFound();
    }

    todoItem.IsComplete = item.IsComplete;
    _context.Update(todoItem);
    await _context.SaveChangesAsync();
    return NoContent();
}
```

Handlebars Cheatsheet

https://devdocs.io/handlebars/

Advanced *Web Applications* and *Web Services*

Day 6

Heritage college

Lucas stephenson

lstephenson@cegep-heritage.qc.ca

Important Dates

May 12th : Midterm
May 13th : Assignment Instructions
May 25th : Last day (presentations and quiz)

Todo Client

MAKE A GIT PROJECT

ADD TO GITHUB

ADD SORT BY DEADLINE

Create Remote Repo and local

Put both projects in the same parent directory

Where the repo will be

Generate gitignore files for both projects

API: dotnet new gitignore

Client: use gitignore.io

Add to repo

Commit

Push!

Sort

- Modify the get handler in your controller

```
IQueryable<TodoItem> todoItemsQuery =
    (from tdTable in _context.TodoItems orderby tdTable.Deadline select tdTable);

return await todoItemsQuery.AsNoTracking ().ToListAsync ();
```

Working with Maps

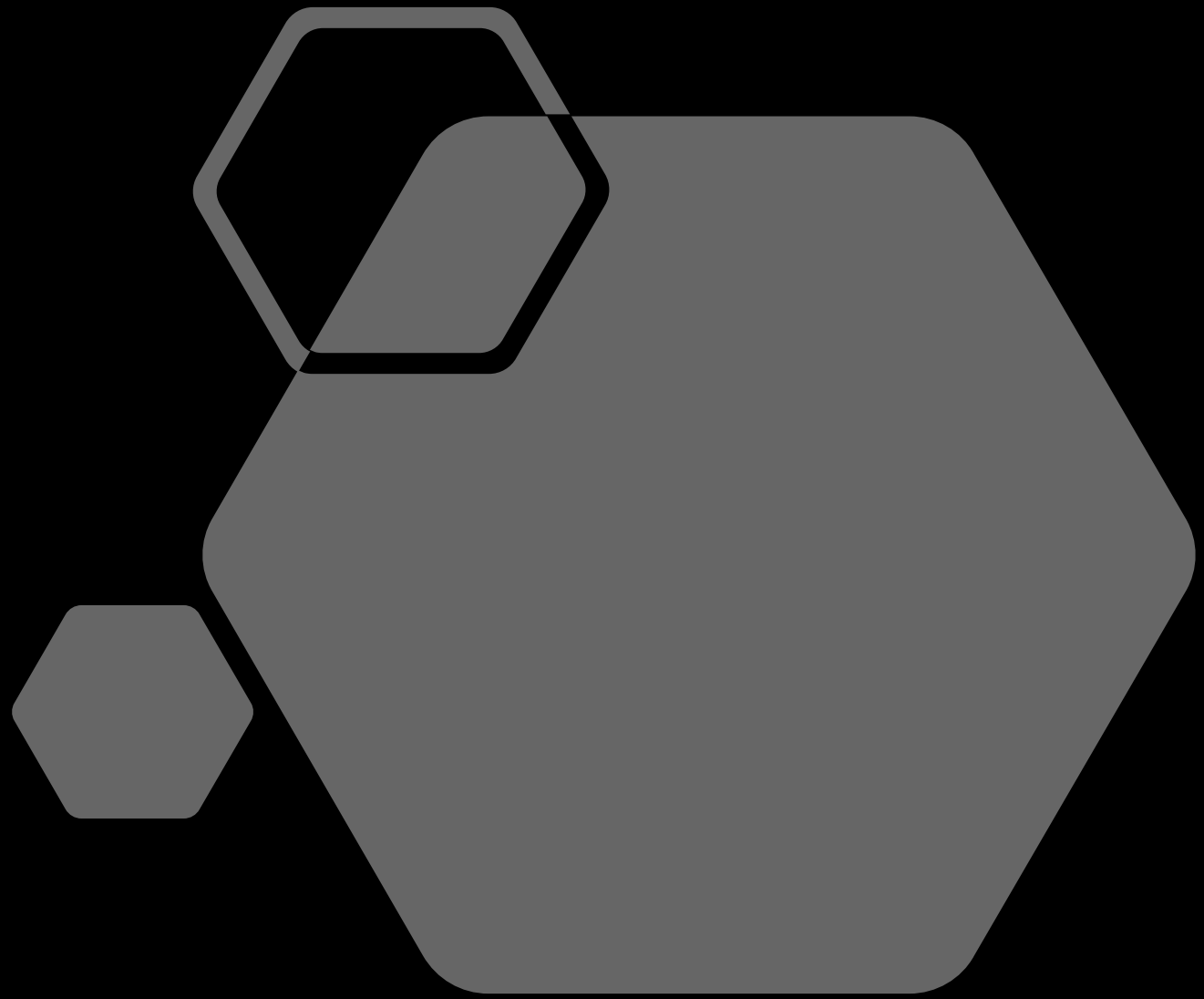
Many apps are maps based

Reuse what others have done

- Don't have the resources to create our own maps

Google Maps

- Is the most popular
  - 2 years ago (June 2018) they changed policy
    - 200$ /month fee
    - Steep afterwards
    - Need to give them billing info (credit card) for ***any*** use
    - We will **not** use



Alternative

- There are a few alternatives
  - Become a bit more technically complex
  - Some involve using separate tile sources and UI (aka slippy map)

Maps

We will use mapbox

Get account https://www.mapbox.com/

See access token @ https://account.mapbox.com/

Design template

They used to be supported by the Leaflet.js library

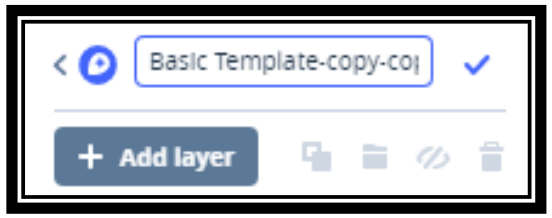
Have their own now

WebGL based

- Render OpenGL in HTML element

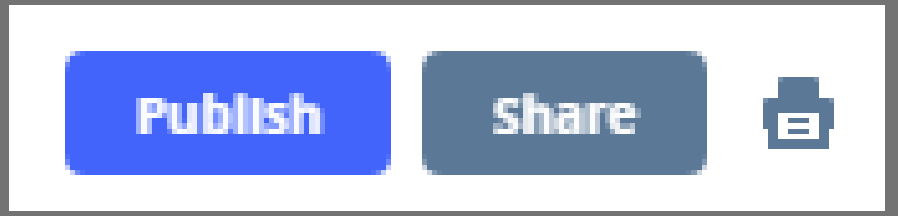
Styles

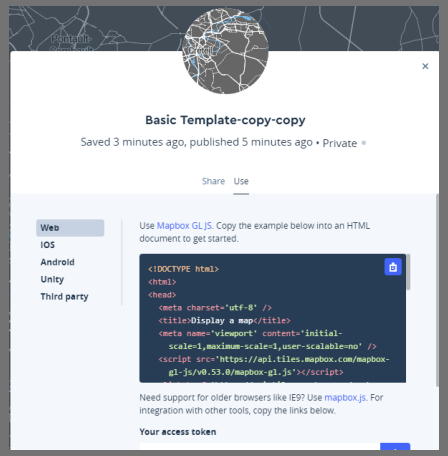
- Visit https://studio.mapbox.com/
- Make a new Style
  - Anything you want
  - Give it a name
  - Publish when done



Share...

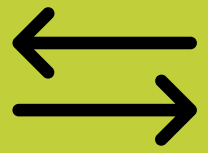
- Select share
- Use
- We will copy all the source for now





Make a new Project

Paste what you copied into index.html



Move the JavaScript to a *file js/map.js*

Put the map initialization in a *DOMContentLoaded* event

Move from pasted code, into map.js

```
mapboxgl.accessToken ='token';

document.addEventListener('DOMContentLoaded', () => {
    const map = new mapboxgl.Map({
        container: 'map',
        style: 'mapbox://styles/username/styletoken',
        center: [-82.2059, 42.3847],
        zoom: 10.7
    });
});
```

Set Initial Zoom and Location

CENTER: [-75.765, 45.4553]

ZOOM: 14.5

Add Some CSS

css/style.css

- Remove CSS from index.html
- Add a container div
- Don't forget CSS reference!

```
<body>
    <div class="container">
        <h1>You are Here</h1>

        <div id="map"></div>
    </div>
</body>
```

```
body {
    margin: 0;
    padding: 0;
}

.container {
    display: flex;
    width: 85%;
    margin: auto;
    flex-wrap: wrap;
}

#map {
    width: 100%;
    height: 70vh;
}
```

IP Geolocation

- We want to set the initial center of the map to the user's (estimated) location
- There are two ways to get the client's geolocation
  1. JavaScript geolocation API
  2. Get the user's location using an IP geolocation service

Client-Side Geolocation

- Pros
  - Accuracy (on mobile devices)
  - Easy to use/develop
  - Works with proxy or vpn
- Cons
  - Requires user acceptance
  - Not supported on all platforms
  - Requires HTTPS*
    - Must run on server
    - Over https (not http)

**Some browsers don't observe this, or/or allow for local files, or localhost*

Client-Side Geolocation

- `navigator.geolocation`
  - Can I Use:

    https://caniuse.com/#search=geolocation%20api
  - User can deny
  - Setting can be turned off
  - Mozilla Docs:

    https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API

-
```
if ('geolocation' in navigator) {
   // geolocation available
} else {
   // geolocation not available
}
```

Methods

- Get once

```
navigator.geolocation.getCurrentPosition((pos) => {
   console.log(pos);
});
```

- Get continuously
  - Repeatedly and
  - when changes or more accurate info becomes available

```
const geoWatcher = navigator.geolocation.watchPosition((pos) =>
{
   console.log(pos);
});
// stop watching
navigator.geolocation.clearWatch(geoWatcher);
```

Server-Side Geolocation

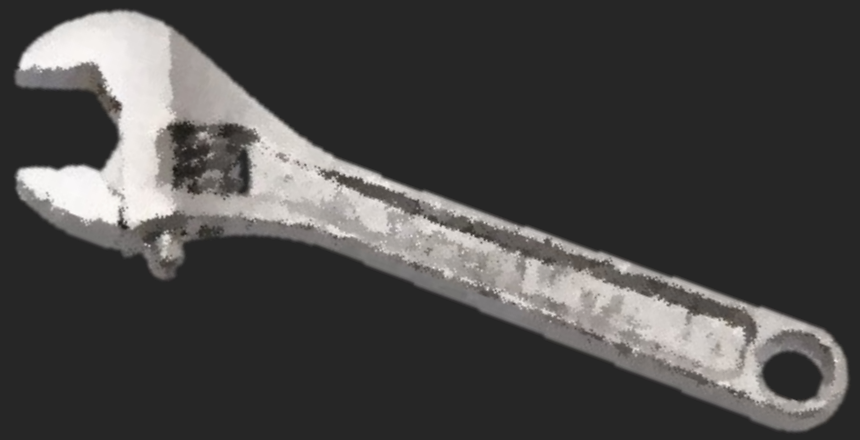
- Cons
  - More difficult to develop
  - Requires 3rd party service
  - Less Accurate
- Pros
  - No need for user approval
  - Broad support (if client supports AJAX, can use)

- 3rd Party Service
  - There are lots
  - We will use
    - https://tools.keycdn.com/geo

Can use server-side as fallback when client-side not available

Both?

Wrench



- More difficult than it sounds
  - If they decline is not the same as client doesn't support
- Can use permission API
  - Problem:
    - Not as supported as geolocation API
      - https://caniuse.com/#search=permissions%20api
    - Must check if navigator.permissions exists
    - navigator.permissions.query() returns a promise

```
navigator.permissions.query({ name: 'geolocation' }).then((res) => {
   console.log(res.state);
});
```

watchPosition Error

- watchPosition()
- can take another parameter
- 'error handler' →
  - runs when there is an error!

```
if ('geolocation' in navigator) {
   const geoWatcher = navigator.geolocation.watchPosition(
      (pos) => {
         setLocation(
            pos.coords.latitude,
            pos.coords.longitude,
            pos.coords.accuracy
         );
      },
      (err) => {
         // client error trying to geolocate
         // err.code and err.message can provide more details
         backupGeoLocation();
      }
   );
} else {
   // client doesn't support geocoding
   backupGeoLocation();
}
```

Consuming the Web Service

- Look at the documentation
  - Find the endpoint
  - Call the API

Wrench

- https://tools.keycdn.com/geo
- Is free, but doesn't support CORS

Make an Express Server

geoservice

01

Make new node project

02

Use express generator

- Grab the starter from teams/files

03

Install axios and cors (save)

- npm i axios cors –s

04

Review code in app.js

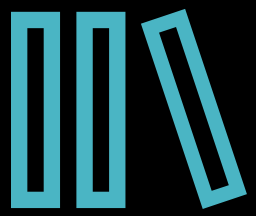
05

Use cors (app.use)

06

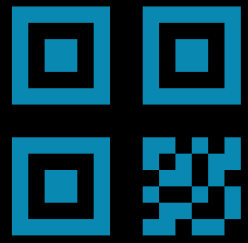
Add axios code

axios



Is one (of many) libraries that can be used to make fetch requests in node (also can be used client side)

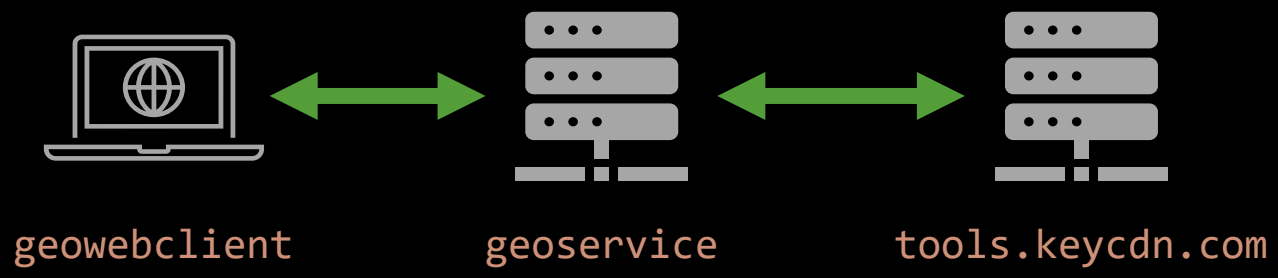
Uses promises



axios.get(url), axios.post(url, data), etc.

GeoService
Handler for geocoding

- Add get handler for /geo
  - "fake localhost requests"
  - Add axios fetch to keycdn with ip from *geowebclient*
  - Review what is returned (data in *result*.data)
  - Return data to our *client*



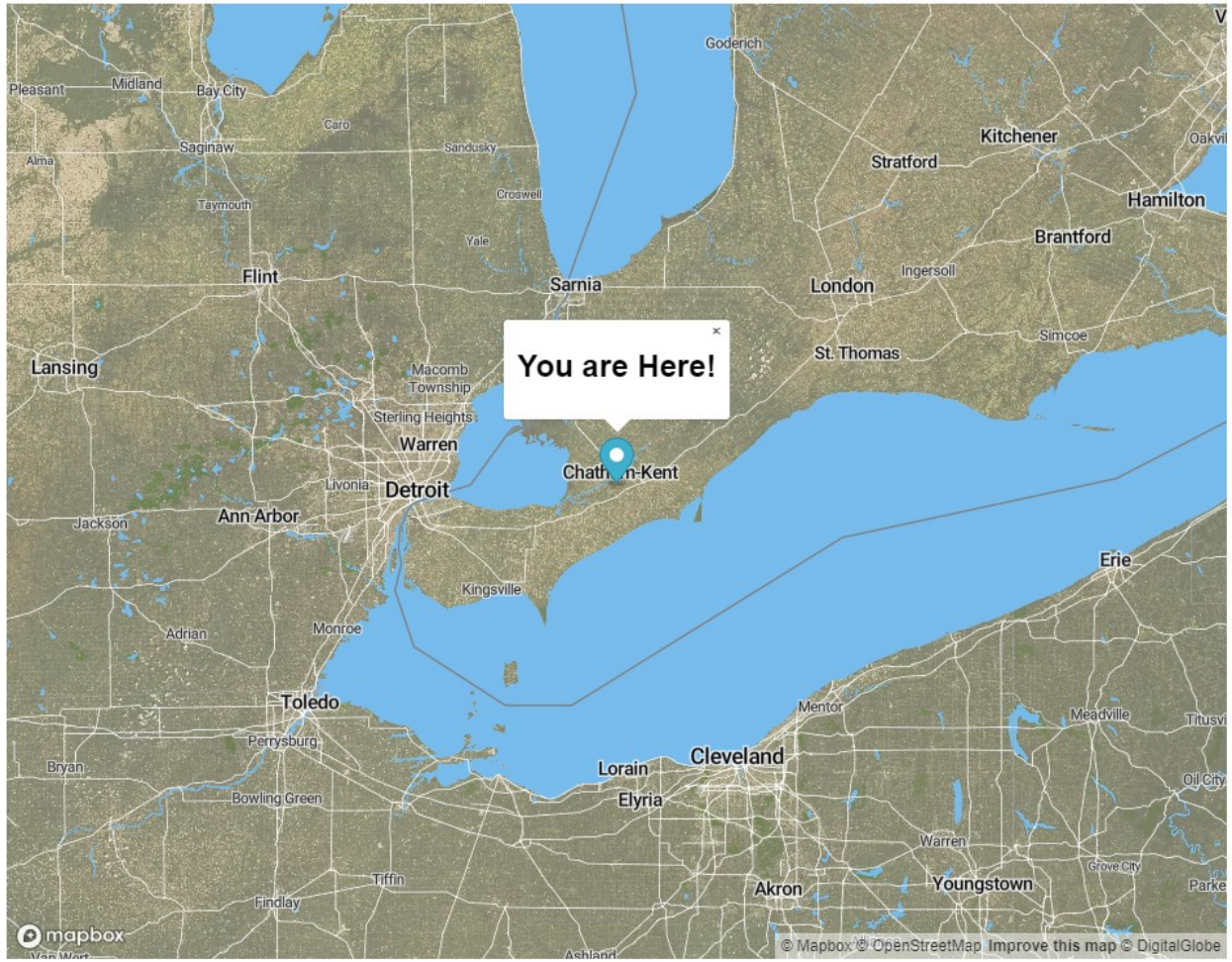
Lab 4: Map
Mapbox Documentation

https://docs.mapbox.com/mapbox-gl-js/api/

Try:

1. Add a marker at user's
2. Add a popup
3. Make it when you click on the title, it re-centers
4. More!

You are Here

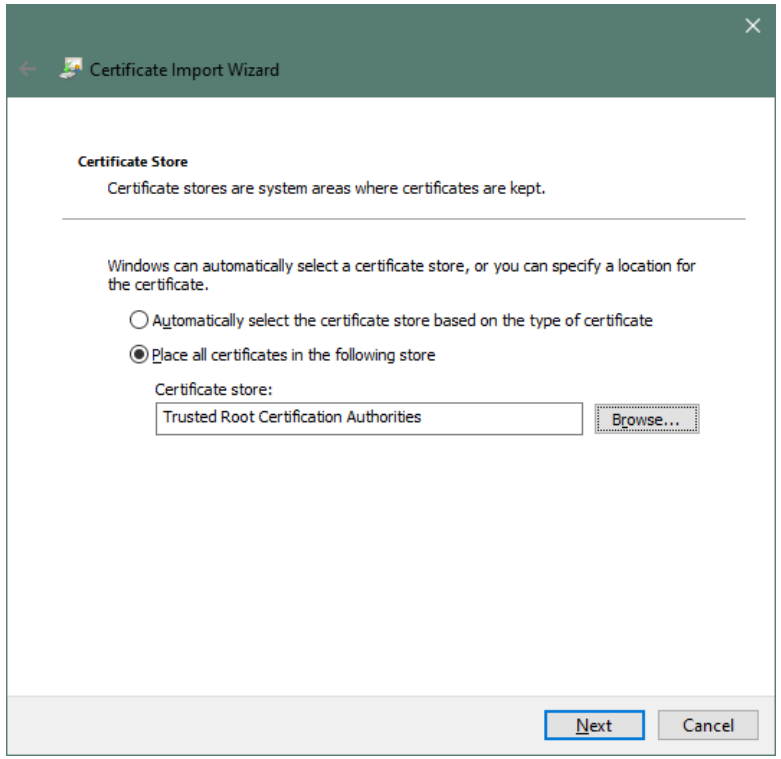

You are Here!
Goderich
Pleasant
Midland
Bay City
Caro
Saginaw
Sandusky
Alma
Croswell
Taymouth
Yale
Flint
Sarnia
Kitchener
Stratford
Hamilton
Brantford
Ingersoll
London
Simcoe
St. Thomas
Lansing
Macomb Township
Sterling Heights
Warren
Detroit
Livonia
Ann Arbor
Jackson
Kingsville
Erie
Adrian
Monroe
Toledo
Perrysburg
Bryan
Bowling Green
Mentor
Meadville
Cleveland
Lorain
Elyria
Oil City
Warren
Tiffin
Findlay
Akron
Youngstown
Grove City
mapbox
© Mapbox © OpenStreetMap Improve this map © DigitalGlobe

Create a Self-signed certificate

- The HTTPS protocol implies "encryption"
  - A secure socket connection: this actually "wraps" around the HTTP protocol
    - SSL: *Secure socket layer*
  - To encrypt/decrypt a set of keys are required
    - This includes a secret key, and a public key
    - Usually packaged into a certificate
    - These can be verified → certificate signatories are (supposed to be trusted)
    - We can generate our own certificates, but no one will trust!
    - We can trust it manually: can be useful for web development

OpenSSL

- May need to install (windows):
  - https://slproweb.com/products/Win32OpenSSL.html
  - Mac: not sure, try reading: https://medium.com/@timmykko/using-openssl-library-with-macos-sierra-7807cfd47892



Generate

Keys:

```
openssl req -x509 -nodes -new -sha256 -days 1024 -newkey rsa:2048 -keyout key.pem -out cert.pem -subj "/C=US/CN=localroot-Root-CA"
```

Certificate:

```
openssl x509 -outform pem -in cert.pem -out RootCA.crt
```

Trust:

This will stop the warning pages...

In windows: double click on the .crt file, or right-click "install"

Install into: *Trusted Root Certification Authorities*

Firefox

- On windows, FireFox needs to be set to allow local certificates
  - Go to: about:config
  - Set `security.enterprise_roots.enabled` to `true`

Advanced *Web Applications* and *Web Services*

Day 7

Heritage college

Lucas stephenson

lstephenson@cegep-heritage.qc.ca

Today

Quiz

Assignment

Catch Up

Quiz

Quiz is available

https://b.socrative.com/login/student/

Room: HRTGX22

https://b.socrative.com/login/student/

I will shut down the quiz tomorrow morning, or after everyone has completed it

Whichever is sooner (if I am awake)

Assignment

Document

- New (slightly changed) document on Teams

Groups

- Highly recommend you work in a group
  - However, you can work alone if needed!
  - Groups of 1-4 allowed
  - Evaluation will consider size of group

Options
Map Based

- Map based application
  - Make a map-based application
  - Get data from at least 1 external webservice
    - Not including IP geocoding
  - E.g. Show parks on a map – list 4 closest ones?

Options

- Web App
  - Interact with at least 1 webservices
    - Multiple endpoints
    - Does something interesting
  - E.g. Create a user webservice
    - Login/create, etc.
    - Let users set a status

Requirements

The application must have a useable client

Should use a local server and storage

The client must work

some interactive elements

E.g. A webservice request is used in response to some data provided by the user

Must use a build-stack: webpack,

Not unnecessarily re-executing web service calls.

E.g. don't geocode the same IP or address over and over

The application must be styled and laid-out thoughtfully (CSS/HTML)

Presentation

DEMONSTRATE THE APPLICATION

HIGHLIGHT INTERESTING PARTS

DISCUSS DIFFICULTIES ENCOUNTERED

INDICATE WHO DID WHAT

CONSIDER THE METHOD OF DELIVERY: REMOTE VIDEO

DISCUSS WHY YOU CHOSE WHAT YOU CHOSE TO INCLUDE

Collaboration Evaluation

- Peer evaluation
  - Can effect shift individual mark up to 20%

Schedule *recommended*

- Planning and communication with me will be evaluated
- Friday, May 15th
  - A document with:
    - Group Name
    - Group Members
    - Brief description of what you want to do
- Meet with me often (via teams) in a group calls.
- Monday, May 25th
  - Video or live presentation by the end of the day on

Evaluation

Evaluation	Grade	% of Grade
Application	**18**	**60%**
Application makes use of web services	7	23%
Application is functional	5	17%
Remote data is combined/used thoughtfully	4	13%
Presentation	**6**	**20%**
Demonstrate application	2	7%
Discuss pain points	2	7%
English	2	7%
Organization	**6**	**20%**
Total	**30**	**100%**

Check	Check that you have done Labs 1,3,4 and I've seen your video
Review	Review what we've done in class
Form	Form a group

Advanced *Web Applications* and *Web Services*

Day 8

Heritage College

Lucas stephenson

lstephenson@cegep-heritage.qc.ca

Fetching XML

- Reminder
  - The concept of doing an HTTP request from JavaScript is often called "AJAX"
    - From a browser, this is accomplisged the xmlHttpRequest object of the fetch API
    - From the server or desktop application → depends on the platform
- Most things you get from AJAX will (hopefully) be in JSON
  - But could get any kind of data!

Fetching XML

- To fetch XML use the same APIs as with JSON

```
let res = await
        fetch('https://www.gatineau.ca/upload/donneesouvertes/BORNE_FONTAINE.xml');
```

- In a browser client you can process into a "document"
  - From the "text"

```
let data = await res
   .text()
   .then((str) => new window.DOMParser().parseFromString(str, 'text/xml'));

locations = data.querySelectorAll('GEOM');

locations.forEach((el) => {
   let val = el.innerHTML.substring(7);
   val = val.substring(0, val.length - 1).split(' ');
   lng = val[0];
   lat = val[1];
   markers.push(new mapboxgl.Marker().setLngLat([lng, lat]).addTo(map));
});
```

Break

Object ↔ Storage

Spectrum

- There are "high-level" and "low-level" tools for interacting with DBMS
  - Database drivers we have been using
    - Write platform specific SQL or other domain specific language:
      - differences between DBMS
      - Lower mobility
  - Mid-Range
    - Independent of underlying DBMS
    - Often defines a new language, or standardizes SQL
      - LINQ, Knex
  - ORM
    - Don't interact with storage engine directly
    - Stores class/objects used in code directly

Object-Relational Mapping

- A layer that allows objects to be stored/retrieved in a storage engine
  - Don't need to explicitly* define storage

ORM
Pros

- Fast to develop
  - Write data structure once (either direction)
- Easier to move between storage engines
- No reliance on domain specific languages (SQL)
- Less maintenance
- Changes can be semi-automatic

ORM *Cons*

- Don't understand the underlying system
  - What if *Magic* fails?
- ORM systems are often just as complex as using the DBMS directly
  - Might as well use database directly → easier to optimize
  - There is no "master" ORM
    - Microsoft Entity Framework, Microsoft Entity Framework Core, Sequalize, Bookshelf, etc.
    - There is less variation between SQL dialects than there is ORMs!
- Efficiency
  - Extra layers decrease performance
- ORM Might not support thew feature(s) you need
  - Many ORMs don't fully support subqueries

ORM

- I don't recommend the use of ORM most of the time
- But can be a relatively quick way to create an API

Node ORMs

Sequelize
https://github.com/sequelize/sequelize

Sequelize is a promise-based Node.js ORM for Postgres, MySQL, MariaDB, SQLite and Microsoft SQL Server. It features solid transaction support, relations, eager and lazy loading, read replication and more. Sequelize follows Semantic Versioning.

Bookshelf
https://github.com/bookshelf/bookshelf

Bookshelf is a JavaScript ORM for Node.js, built on the Knex SQL query builder. It features both Promise-based and traditional callback interfaces, transaction support, eager/nested-eager relation loading, polymorphic associations, and support for one-to-one, one-to-many, and many-to-many relations.

It is designed to work with PostgreSQL, MySQL, and SQLite3.

Waterline
https://github.com/balderdashy/waterline

Waterline is a next-generation storage and retrieval engine, and the default ORM used in the Sails framework.

It provides a uniform API for accessing stuff from different kinds of databases and protocols. That means you write the same code to get and store things like users, whether they live in MySQL, MongoDB, neDB, or Postgres.

Objection
https://github.com/Vincit/objection.js

Objection.js is an ORM for Node.js that aims to stay out of your way and make it as easy as possible to use the full power of SQL and the underlying database engine while still making the common stuff easy and enjoyable.

Sequelize

- Most popular node ORM
- Make a new express project
  - Can use forwardservicestarter
    - Rename
- Make sure your database engine is running!
  - *MariaDB* or postgres, mysql, sqlite, mssql

Add Sequelize

- Create a new database ('ormex')
  - You CAN setup the database via sequelize, but why?
- Don't forget to install existing dependencies!
- Add sequelize
  - npm i -s sequelize
- Add specific DBMS support
  - npm i -s mariadb
  - For others: https://sequelize.org/master/manual/getting-started.html

Setup

- Require sequelize as a named import

```
const { Sequelize } = require('sequelize');
```

- Connect to the DB

```
const sequelize = new Sequelize('ormex', 'root', '', {
   dialect: 'mariadb',
});
```

- Write a test get route

```
app.get('/dbtest', async (req, res) => {
   try {
      await sequelize.authenticate();
      console.log('Connection has been established successfully.');
   } catch (error) {
      console.error('Unable to connect to the database:', error);
   }
});
```

Model/Data Structures

Can be defined in 2 ways (for sequelize)

Using the sequelize.define() method

Extending sequalize's Model class

Or they can be generated

Uses sequelize.define

Migrations

- Versioning the database
- Transition to different versions
- Need a command-line tool

```
npm install --save-dev sequelize-cli
```

- Init the sequelize project

```
npx sequelize-cli init --models-path=Model
```

- Generate a model:

```
npx sequelize-cli model:generate --models-path=Model --name User --attributes first:string,last:string,username:string,birthday:dateonly
```

Configure ...config/config.json

```
{
    "development": {
        "username": "root",
        "password": null,
        "database": "ormex",
        "host": "localhost",
        "dialect": "mariadb",
        "pool": {
            "max": 10,
            "min": 0,
            "acquire": 20000,
            "idle": 8000
        }
    }
}
```

Structure

- You can edit the migration to better match your needs
  - DataTypes: https://sequelize.org/master/manual/model-basics.html#data-types
  - Options:
    - defaultValue, allowNull, autoincrement
    - https://sequelize.org/v5/manual/models-definition.html

Apply the migration

`npx sequelize-cli db:migrate`

- This will make the table

Dealing with migrations:

- Undo the last migration

  ```
  npx sequelize-cli db:migrate:undo
  ```

- Revert al migrations

  ```
  npx sequelize-cli db:migrate:undo:all
  ```

- Revert to specific migration

  ```
  npx sequelize-cli db:migrate:undo:all --to XXXXXXXXXXXXX-create-posts.js
  ```

Using the model

- Require
  - The Model folder (with index.js and all Models)

```
const db = require('./Model');
```

  - Make an alias for each model

```
const User = db.User;
```

Working with Models

- create({obj})
  - Inserts a new record, obj, keys are the fields

```
User.create({ first: 'fred' });
```

- findAll()
  - Finds all records - returns
- findAll({where: {column:val})
  - Finds records where column = val
- update({obj}, {where {column: val}})

```
let up = await User.update({ last: 'fredderson' }
, { where: { last: null } });
```

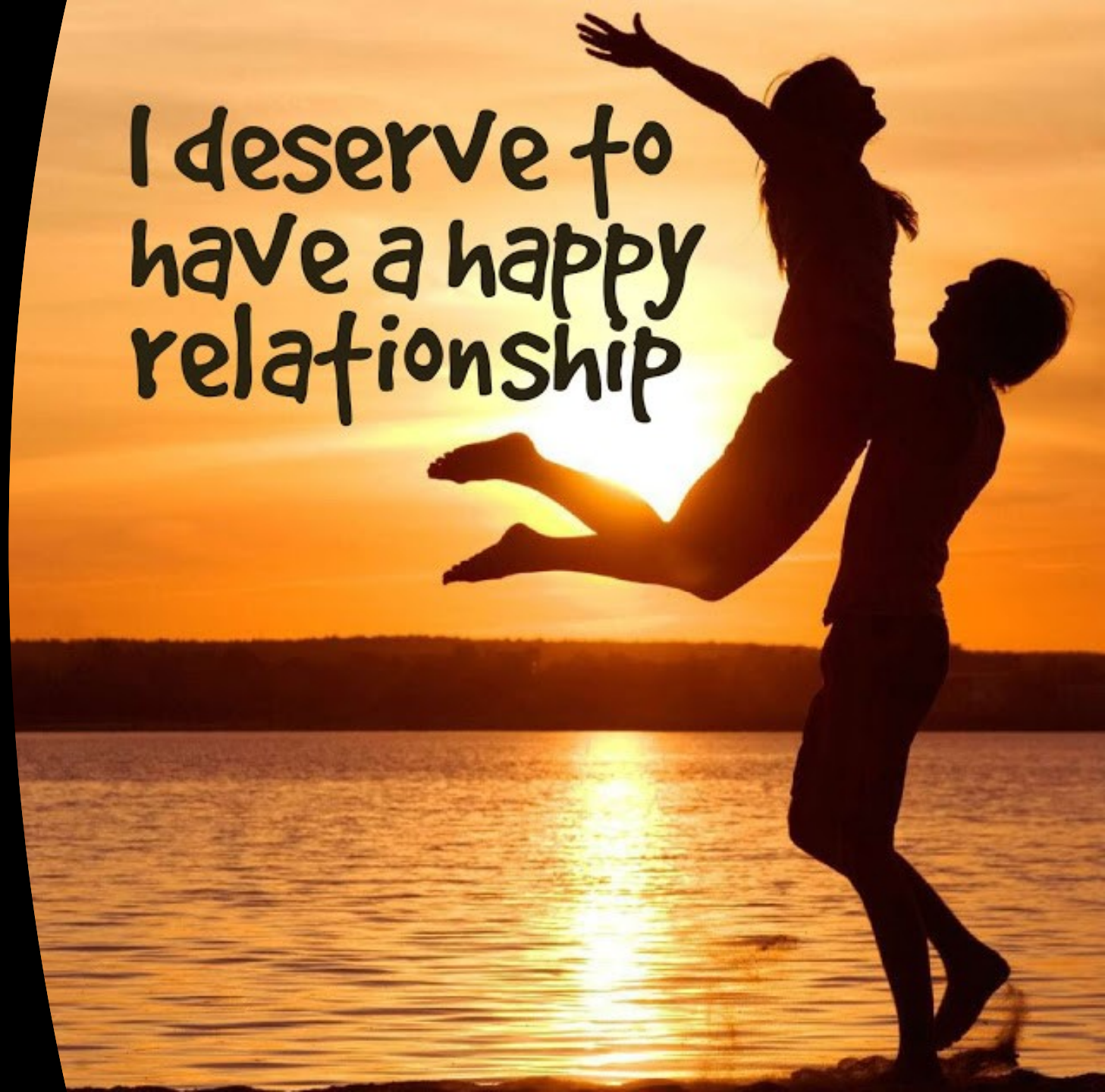
More Basic Queries

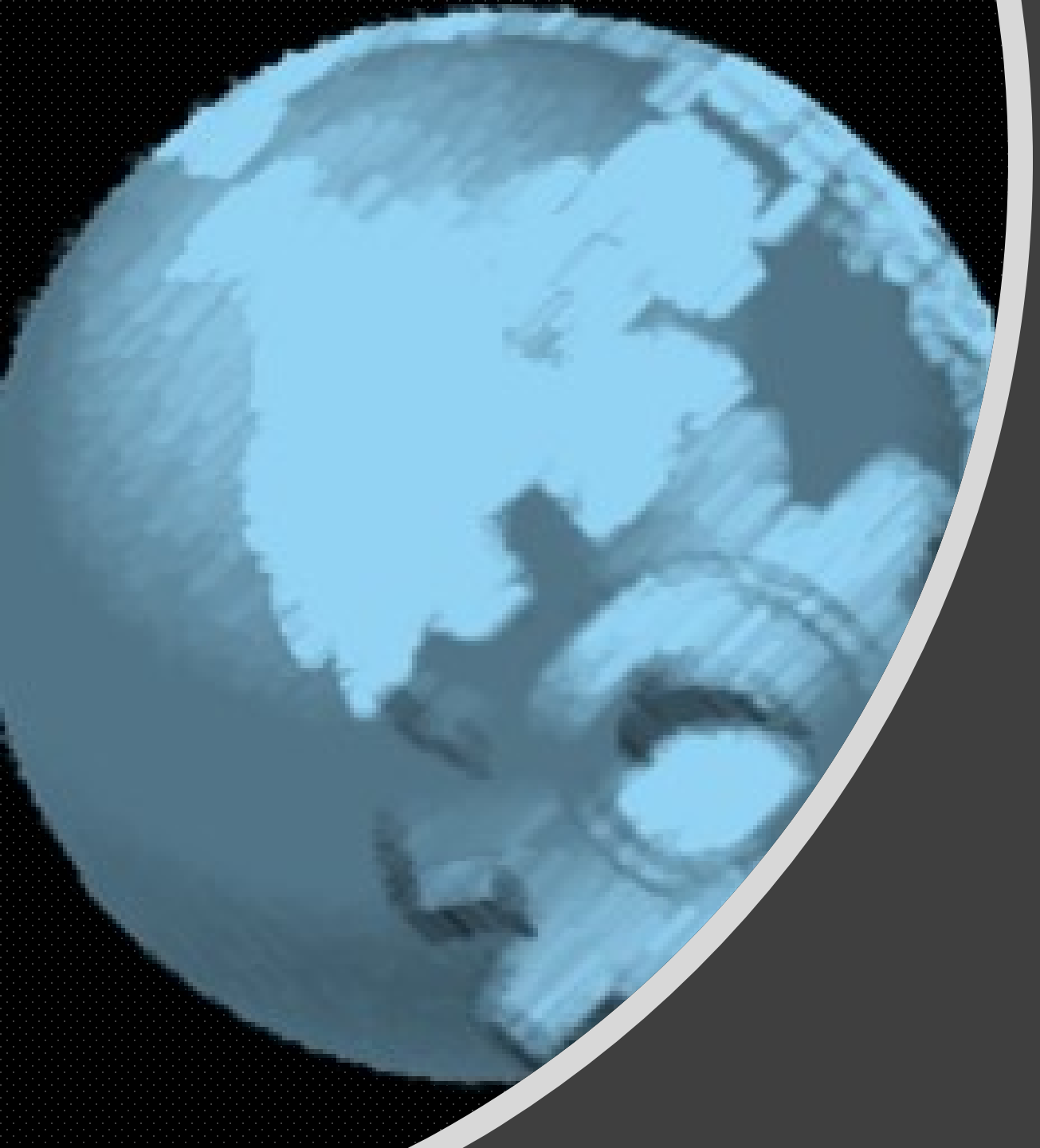
- https://sequelize.org/master/manual/model-querying-basics.html

Associations

Aka : relationships

- https://sequelize.org/master/manual/assocs.html





Advanced *Web Applications* and *Web Services*

HTTP Headers

Day 10

Heritage college

Lucas stephenson

lstephenson@cegep-heritage.qc.ca

HTTP Headers *Request*

- HTTP Headers allow some "meta data" to accompany requests (and responses)
- Main *request* uses are: indicating the
  - content-type (Content-Type, Accept)
  - user agent (User-Agent)
  - Accepted
    - languages
    - character sets
    - Encoding
  - Cookie

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5)
Gecko/20091102 Firefox/3.5.5 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120
Pragma: no-cache
Cache-Control: no-cache
```

HTTP Headers
Response

Examples

- CORS
  - Access-Control-Allow-Origin
  - Access-Control-Allow-Credentials
  - Access-Control-Expose-Headers
  - Access-Control-Max-Age
  - Access-Control-Allow-Methods
  - Access-Control-Allow-Headers
- Content-Type
- Set-Cookie

The Wiki Page Has a Good List

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

See the request headers

Create a new express app

Copy/unzip the express starter (express-mariadb-starter)

Install dependencies

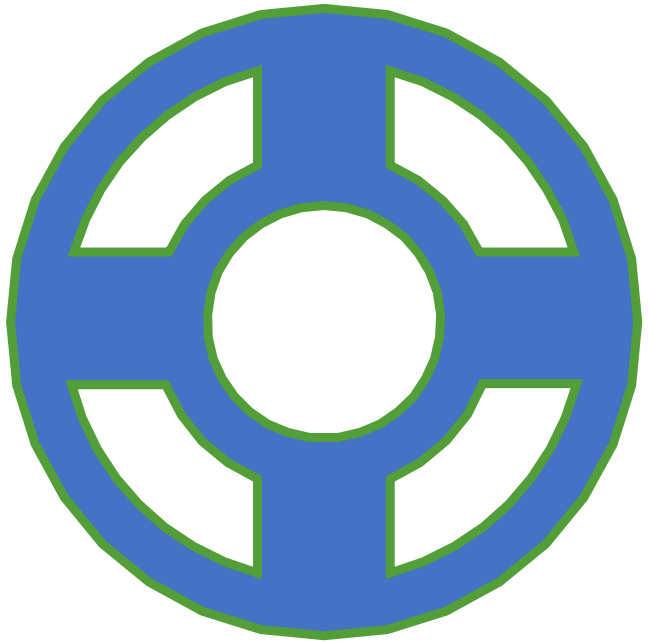
Make an endpoint for this (eg /headers)

Make an html and javascript in /public to fetch the endpoint

/headers endpoint

```
app.get('/headers', async (req, res) => {
   console.log(req.headers);

   res.send("see server");
});
```



Try It

Why do we Care?

Headers are another way to pass information between client/server, besides just URI and cookies

User doesn't see these directly

Good for APIs

We can do authorization using the authorization headers

Authorization Header

- Content is not extensively standardized
  - Usually a "type" followed by auth data
    - We will ignore "type" and use "basic"
  - Auth data Encoded in base64
    - A string -> binary (hex) representation
    - Prevents conflicts
    - Reversable
  - Eg:
    - Basic bob:password

```
Basic Ym9iOnBhc3N3b3Jk
```

Setting HTTP Request Headers

The main way we (manually) create http requests is through AJAX

Forms and links also generate http requests

We can manually set these headers

Authorization header Request

```
const user = "bob";
const pwd = "1234";
const encodedLogin = btoa(`${user}:${pwd}`);
(async () => {
    const response = await fetch("/login", {
        headers: { Authorization: `Basic ${encodedLogin}` }
    });
    const data = await response.json();
    console.log(data);
})();
```

Setup the user table

- Create a new database: header-ex
  - Add a minimal table (user)
  - userId (primary), username,password
- Modify config/dbConn.js
- Modify model/User.js

Using the Authorization Header Data

Need

Need to grab the data from the header

Need

Need to get the data

- Split the word "basic"
- Reverse base64
- In our case, split the user/pass "user:password"

Use

Use the data to verify they have access to the resourse

ExpressJS – Single Request

```
app.get("/getUser", async (req, res) => {
    if (!req.headers.authorization) {
        return res.status(403).json({ error: "credentials missing!" });
    } else {
        let up = req.headers.authorization.split(" ")[1];
        up = Buffer.from(up, "base64").toString();
        up = up.split(":");
        const user = up[0];
        //const pass = up[1];
        let data = { username: user };
        res.json(data);
    }
});
```

ExpressJS – Middleware

```
module.exports = (req, res, next) => {
    const auth = { isAuth: false };
    if (req && req.headers.authorization) {
        let up = req.headers.authorization.split(" ")[1];
        up = Buffer.from(up, "base64").toString();
        up = up.split(":");
        const user = up[0];
        const pass = up[1];
        if (user == "bob" && pass == "password") {
            auth.isAuth = true;
            auth.user = user;
        }
    }
    req.auth = auth;
    next();
};
```

Advanced *Web Applications* and *Web Services*

HTTP Headers

Day 11

Heritage college

Lucas stephenson

lstephenson@cegep-heritage.qc.ca

2020

Important Dates

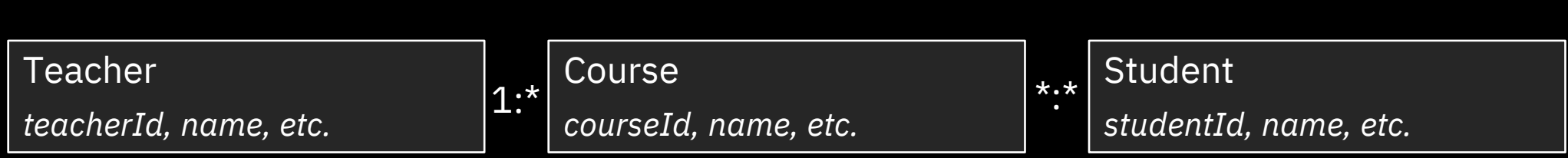
May 25th : Last day (presentations and quiz)

GraphQL

- A technique that brings subqueries/JOINS to REST-like apis
  - If we want to find a list of all student numbers a teacher teaches
  - This is hard to do via REST
    - Make specific endpoints (/teachers/students/ ?)
    - Make lots of individual requests
      - Find the classes a teacher teaches
      - For each class, find the student list
  - Reduces extra requests and data
    - Can improve performance

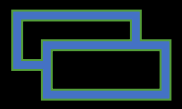
GraphQL Can return (relational) data, in response to relational queries

- Return all courses, with their teachers and students
- Find all the courses with a specific student in it
- Etc.



GraphQL Schemas

- Templates defining relationships
  - **Teacher** has name, etc and **Course***
  - **Course** has name, etc, **Teacher** and **Student***
  - **Student** has name, etc. and **Course***
- **Can be defined using a variety of formats**
  - JSON
  - SDL (GraphQL Schema definition language) (.graphql, or .gql)
  - GraphQL introspection query result (JavaScript)



GQL Files

- Represent the types in the graph
  - Specify their fields and relationships
- E.g.

```
type Teacher {
   id: ID!
   last: String
   first: String
   courses: [Course!]!
}
```

- Does not need to be the same/all fields as tables (but might be)

? GraphQL Allowed *Queries*

- Queries that can have subqueries attached
  - Need to define all possible queries
  - These are what are "run"

```
type Query {
   teacher(id: Int!): Teacher
   teachers: [Teacher!]!
   courses: [Course!]!
   students: [Student!]!
}
```

Field types

- Scalar Types: ID, String, Int, Float, Bool
- Enumeration: enum
- Object types: user defined type (like a table record)

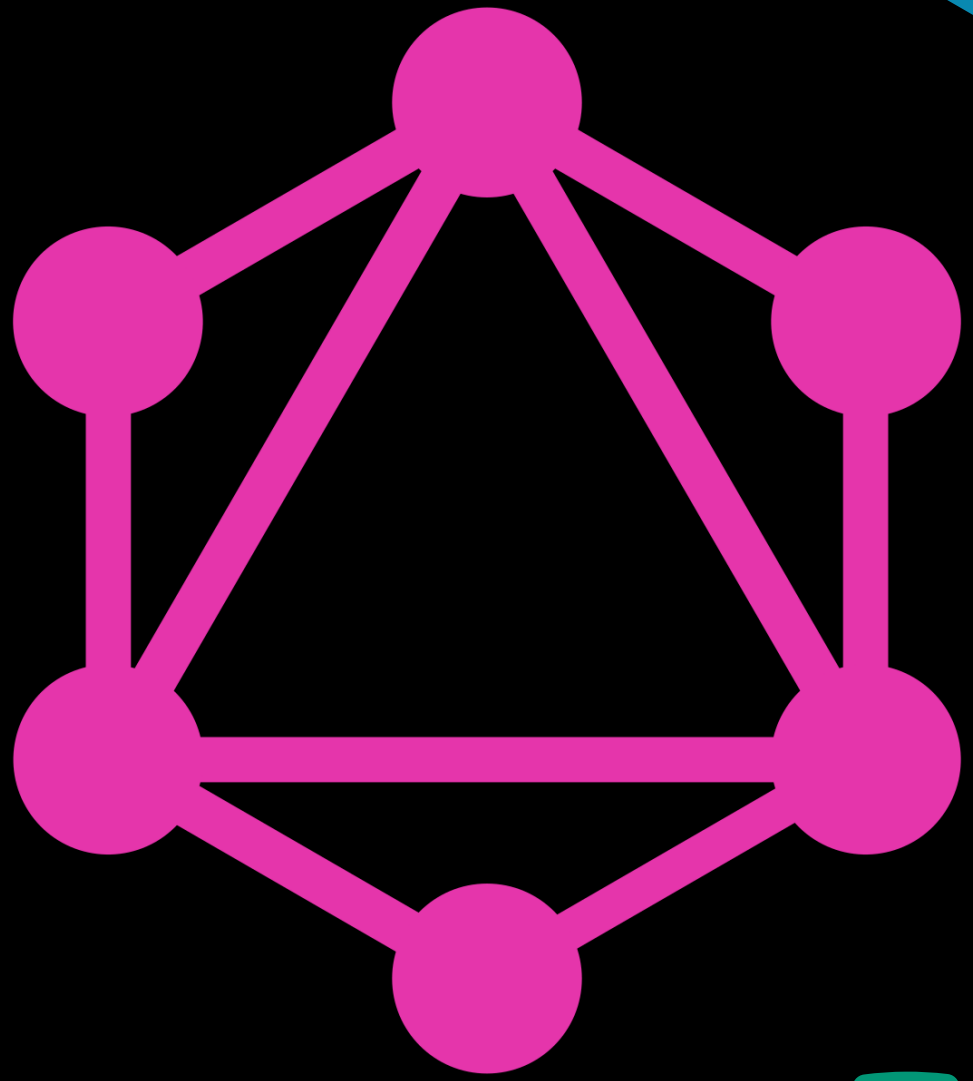
Lists

- Specified with []
  - Many of that type
  - ! – means not null

```
type Teacher {
   id: ID!
   last: String
   first: String
   courses: [Course!]!
}
```

Resolvers

- Resolvers are implementations of queries
  - Aka: The methods to implement the queries specified in the schema
- Can take arguments



Setup

GraphQL & Express

Start

1. Use express-mariadb-starter
    1. Copy/rename *graphql-ex*
    2. Initial install
    3. Add *graphql* and *express-graphql* packages
    4. Change database *(in dbConn):* graphql-ex
    5. Add require to app.js

```
const expressGraphQL = require('express-graphql');
```

Add the database

- Add a database (*graphql-ex)*
  - Add *teacher, course, student, courseStudent* tables
  - Enter some dummy data

GraphQL files

- Are not natively supported by node/express
  - Add a module that reads a schema file and "builds" it.
  - config/gqlSchema.js

```
const fs = require('fs');
const graphql = require('graphql');
module.exports.load = (schema) => {
    fdata = fs.readFileSync(schema, 'utf8');
    return graphql.buildSchema(fdata);
};
```

Define schemas

- Create Teacher, Course and Student Schemas
  - Add a gql file in model: school.gql

```
type Teacher {
   id: ID!
   last: String
   first: String
   courses: [Course!]!
}
```

- Add possible queries

```
type Query {
   teacher(id: Int!): Teacher
   teachers: [Teacher!]!
   courses: [Course!]!
   students: [Student!]!
}
```

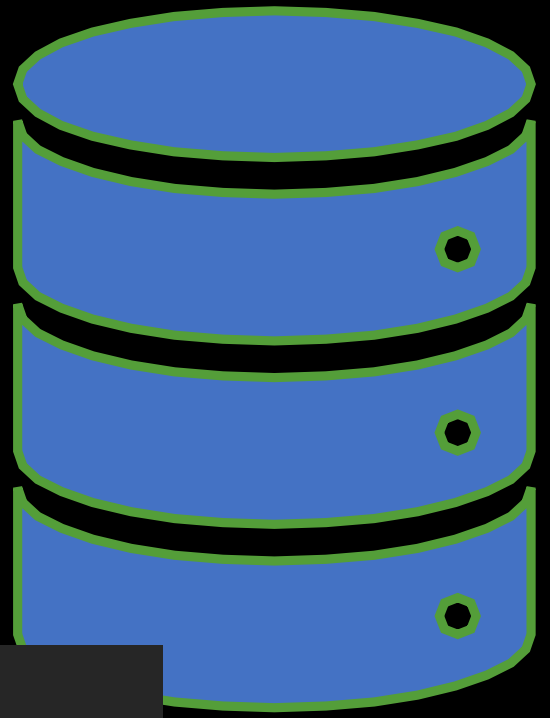
Add the loader and load the schema

- App.js

```
const schemaLoader = require('./config/gqlSchema.js');
const schema = schemaLoader.load('./model/school.gql');
```

Add resolvers

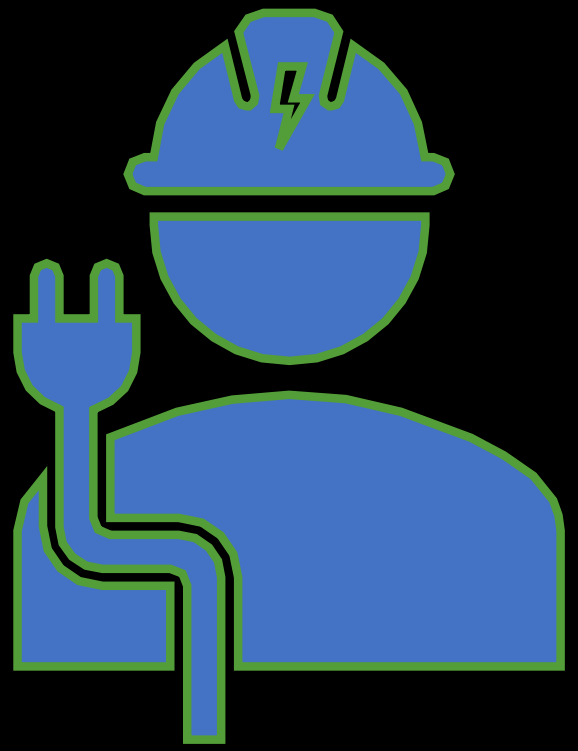
- Create a resolvers,js
  - In a module, create a json object with a "Query" child
    - Child attribute methods for each Query specified in the schema



```
module.exports = {
   Query: {
      teachers: () => {
         return [{ id: 1, first: 'bob' }];
      },
      teacher: (args) => {
…
```

Load resolvers

```
const gqlr = require('./model/resolvers');
```



Create a graphql endpoint on your server

- Graphql-express is middleware
  - Specify the route to apply the middleware to
  - Pass the schema object
  - rootValue: the query resolvers
  - Graphiql: enable query interface

```
app.use(
   '/graphql',
   expressGraphQL({
      schema: schema,
      rootValue: gqlr.Query,
      graphiql: true,
   })
);
```

Run server

- Run it!
  - npm start
  - Visit localhost:9000/graphiql

Lab 8

- Implement the queries specified in the schema
  - Finish the resolvers , using the database

Advanced *Web Applications* and *Web Services*

GraphQL 2

Day 12

Heritage college

Lucas stephenson

lstephenson@cegep-heritage.qc.ca

2020

Important Dates

May 25th : Last day (presentations and quiz)

GraphQL Resolvers

- Resolvers are intended to "resolve" parts of the schema that are (return) ***not*** scalar/value types

```
type Query {
   teacher(id: Int!): Teacher
   teachers: [Teacher!]!
   courses: [Course!]!
   students: [Student!]!
}
```

```
type Student {
   id: ID!
   first: String
   last: String
   courses: [Course!]!
}
```

```
type Teacher {
   id: ID!
   first: String
   last: String
   name: String!
   courses: [Course!]!
}
```

```
type Course {
   id: ID!
   name: String!
   description: String
   teacher: Teacher!
   students: [Student!]!
}
```

Problem

- `buildSchema` does not allow resolvers to be defined for anything besides root Queries (`Query`)
- Instead we need to load our schema and our resolvers module
  - Pass them both to a different method (`makeExecutableSchema`)
  - Which is part of another package (`graphql-tools`)

Add graphql-tools

Install graphql-tools

- `npm i -s graphql-tools`

Organize!

- Create a folder
- Move gqlSchema, resolvers and the schema file (school.gql)
- Require gql-tools in the loader
  - Add a parameter to the module function
    - Resolvers, which will represent our resolvers module

Update schema loader (gqlSchema)

- Require graphql-tools
- Call and return `makeExecutableSchema`

```
const fs = require('fs');
const gqltools = require('graphql-tools');

module.exports = (schema, resolvers) => {
   let fdata = fs.readFileSync(schema, 'utf8');
   return gqltools.makeExecutableSchema({
      typeDefs: fdata,
      resolvers: resolvers,
   });
};
```

Update app.js

- Load the schema loader and the resolver
  - (remove the schema)

```
const schemaLoader = require('./graphql/gqlSchema');
const gqlr = require('./graphql/resolvers');
```

- Load/Build the schema by combining the resolvers object with the schema file (path)

```
const gql = schemaLoader('./graphql/school.gql', gqlr);
```

- Remove rootValue from the middleware arguments
- Magesure our schema points to our executavle schema (`gql`)

```
app.use(
   '/graphql',
   expressGraphQL({
      schema: gql,
      graphiql: true,
   })
);
```

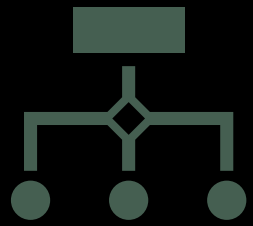
In the model folder

- Make a teacher model
  - Make/move methods to get all teachers and get 1 teacher
  - Remove the courses code from teacher, we will let graphql resolve that
- Make course and student models

In resolvers.js

Require the model files

```
const Teacher = require('../model/teacher');
const Course = require('../model/course');
const Student = require('../model/student');
```



Specify the methods used for non-scalar types

Follows the same structure as the schema

```
const Teacher = require('../model/teacher');
const Course = require('../model/course');
const Student = require('../model/student');

module.exports = {
    Query: {
        teacher: (parent, args) => Teacher.getTeacherById(args),
        teachers: Teacher.getAllTeachers,
        courses: Course.getAllCourses,
        students: Student.getAllStudents,
    },
    Teacher: {
        courses: (teacher) => Course.getCourseByTeacherId(teacher.id),
    },
    Course: {
        students: (course) => Student.getStudentsByCourseId(course.id),
    },
    Student: {
        courses: (student) => Courses.getCoursesByStudentId(student.id),
    }
};
```

Schema

```
type Query {
   teacher(id: Int!): Teacher
   teachers: [Teacher!]!
   courses: [Course!]!
   students: [Student!]!
}
```

```
type Teacher {
   id: ID!
   first: String
   last: String
   name: String!
   courses: [Course!]!
}
```

```
type Course {
   id: ID!
   name: String!
   description: String
   teacher: Teacher!
   students: [Student!]!
}
```

```
type Student {
   id: ID!
   first: String
   last: String
   courses: [Course!]!
}
```

Running a query (from client)

- Simplest is to create graphql query in a string
- Fetch (get) the graphql endpoint with the graphql in the "query" querystring variable

```
let query = '{courses{name}}';
let courseReq = await fetch('/graphql?query=' + query);
let courses = await courseReq.json();
console.log(courses);
```

Advanced *Web Applications* and *Web Services*

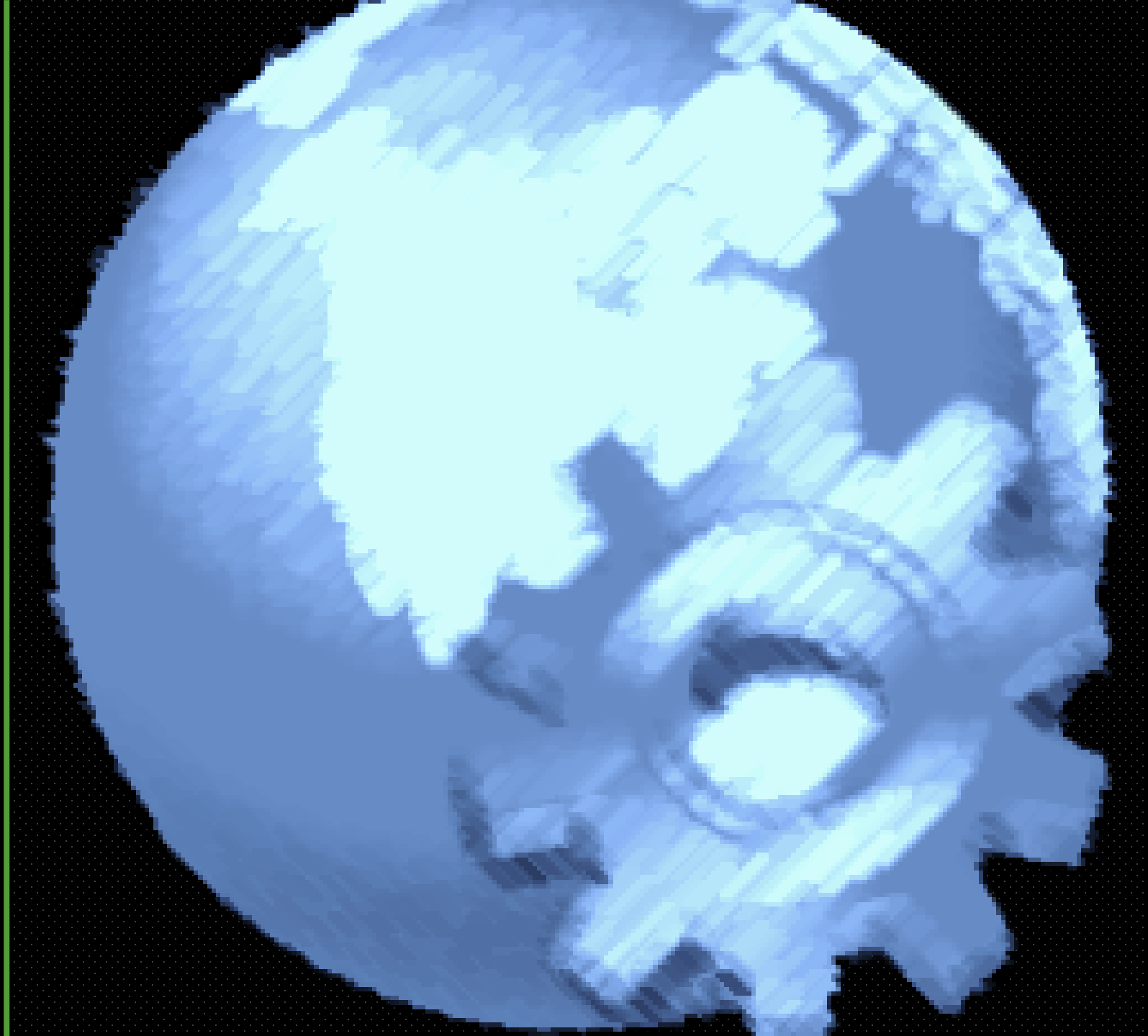
Upload to Cloud Server

Day 13

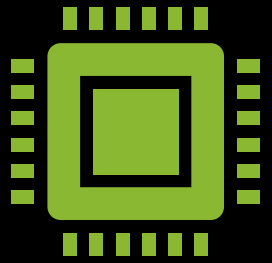
Heritage college

Lucas stephenson

lstephenson@cegep-heritage.qc.ca



Sign-up For an Account



You have limited access to bandwidth for free

Upload node apps

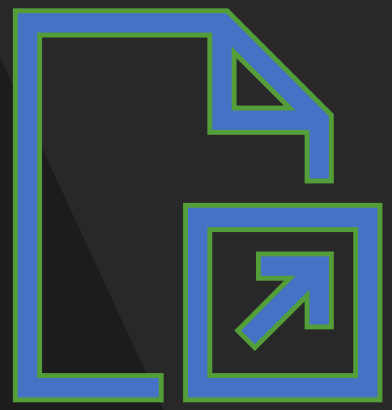
PostgreSQL: small databases

You can add a credit card to get more services

Free or otherwise

Required for MariaDB

Review Code



- Environment Variables

The Heroku CLI

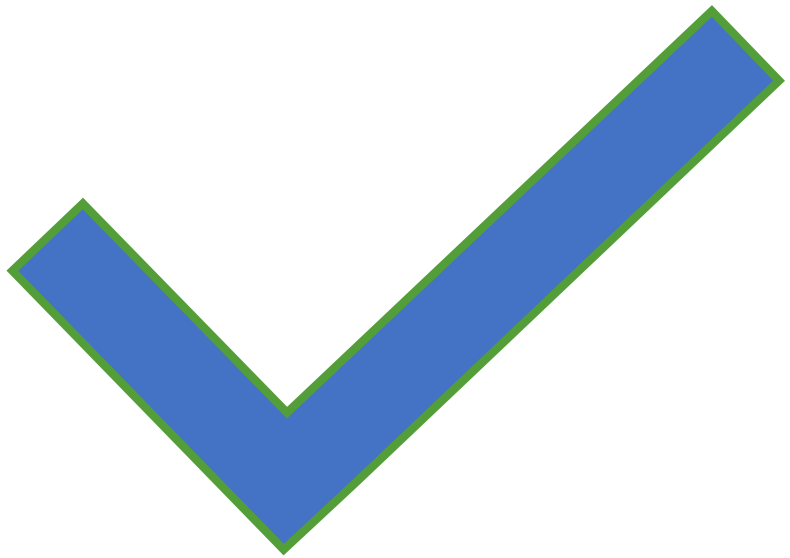
https://devcenter.heroku.com/articles/heroku-cli#download-and-install

- Download/Install

git

- Heroku uses git for code projects
  - Not primarily meant for version control
  - A way to "upload" app
    - Push to Heroku when you want to release a new version
- Create a local git repo for the project

```
git init
git add .
git commit -m "init commit"
```



Heroku CLI

- Use the Heroku cli to "create" a Heroku app

```
heroku create graphql-ex
```

- No spaces in the name, all lowercase, doesn't start with a number
  - You will need to login (first time)
  - It will give you a random name/url and remote repo address
  - Eg: https://thawing-meadow-68388.herokuapp.com/
    - You can rename:
      - max 30 chars
      - Can't be the same as anyone else's

```
heroku apps:rename --app thawing-meadow-68388 ls-heritage-nodejs-mariadb
```

If you just have a node app

- No other external dependencies (like a mariadb server)
  - You will need to change the port, Heroku assigns a random one, available in the constant *process.env.PORT*, when running on heroku
  - You can push your app

```
git push heroku master
```

  - And access it by the URL for your Heroku app

    https://*thawing-meadow-68388*.herokuapp.com/

Setup Maria DB

- Heroku mariadb support is called jawsdb-maria
  - https://devcenter.heroku.com/articles/jawsdb-maria
- Install/Provision to your Heroku account

```
heroku addons:create jawsdb-maria
```

- This is a mariadb server you are setting up (brand new!)
  - Get the address of the new instance by looking at your configuration

```
heroku config
```

  - Or finding the component setting page on Heroku's site
- You can connect your app, or a tool using this information
- Do not share this information!

Get the data from your existing database *(optional)*

- This doesn't work in PowerShell
  - **Use Windows command prompt (cmd)**
  - Non-windows should be able to use their regular terminal
- Output local database to sql file

```
mysqldump -h localhost -u root graphql-ex > backup.sql
```

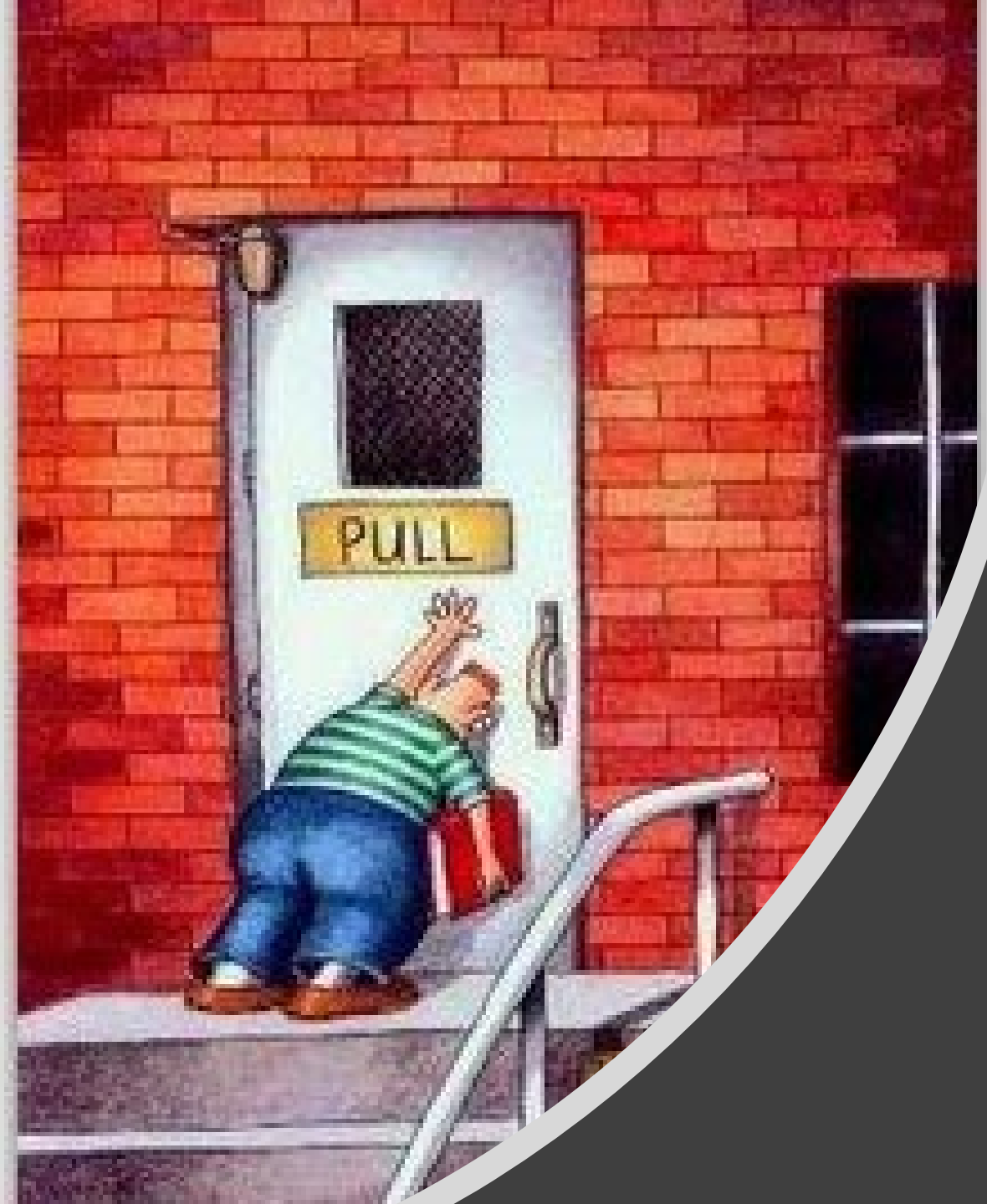
- Upload to remote mariadb
  - Use your connection details (from previous step)
  - (no line break)

```
mysql -h un0jueuv2mam78uv.cbetxkdyhwsb.us-east-1.rds.amazonaws.com -u
dtm4703ipzkg0vt0 -pyieplwduydfjgdvo rt6sijmjd2v0i9c4 < backup.sql
```

No space

Adjust connection details in node

- You will need to adjust connection details to match your new server
  - Edit dbConn.js
  - Optionally make a switch/if for if running on local or heroku



Push App

```
git add .
git commit -m "db config"
git push heroku master
```

Advanced *Web Applications* and *Web Services*

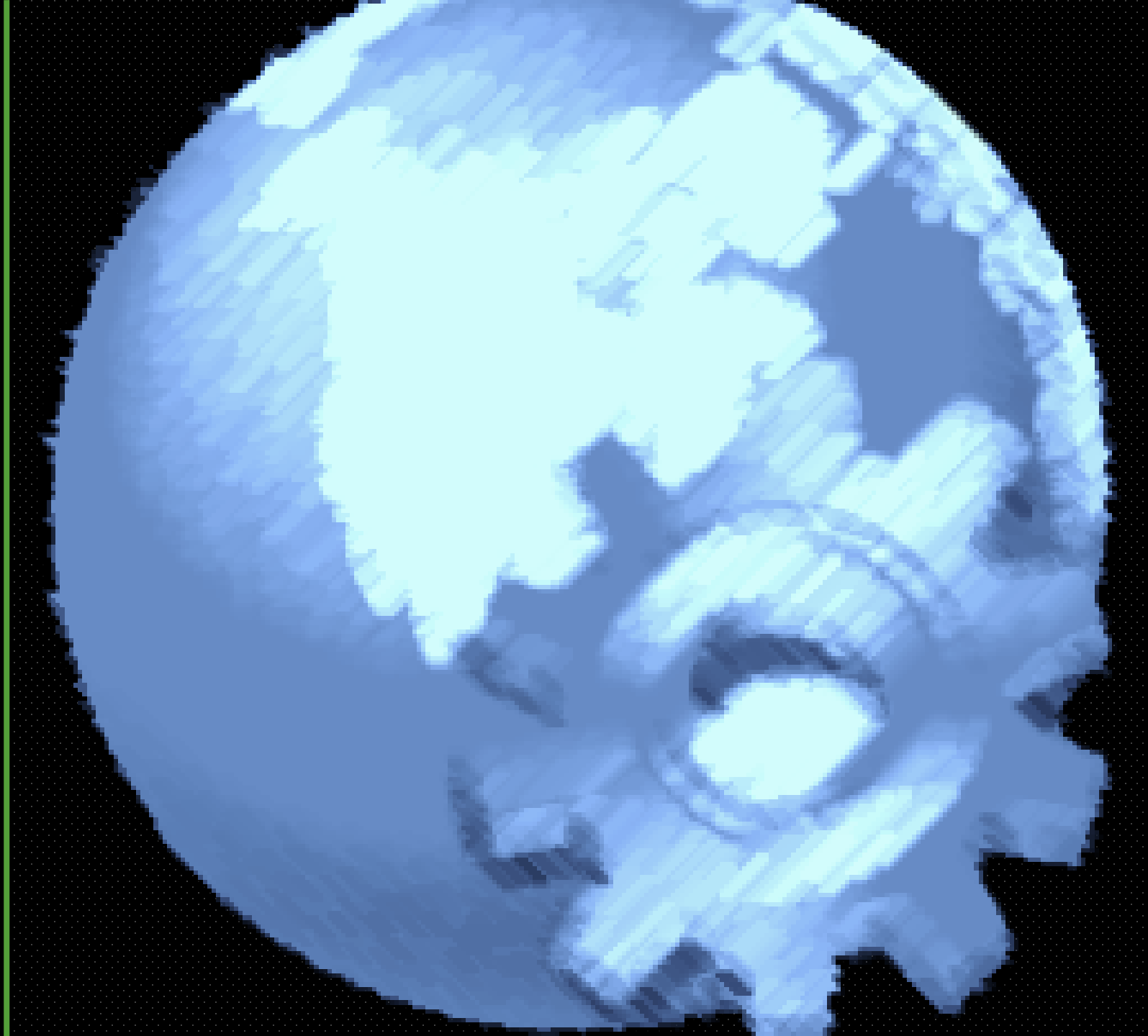
Work and Review

Day 14

Heritage college

Lucas stephenson

lstephenson@cegep-heritage.qc.ca



Final

No Content today

Please tell me (today) if you are

presenting live: I will schedule times today as well

submitting a video: MUST be uploaded to teams

There will be a final quiz/exam on Monday

Available from ~8:30 am, until Tuesday morning or until everyone completes

I will be checking labs

if not posted in the files section under the correct lab, I may not see it

Labs

- 1: XML File
- 3: Database: REST/TodoItems
- 4: TodoItems: Client
- 5: Mapbox Map
- 7: Authorization Header
- 8: GraphQL students and/or courses resolver

Please Review

- Look at the Course Notes (notes tab) to see if there are labs you submitted (or thought you had),
  - Its very possible I forgot/missed some