Adv Web Dev Arch - Lecture 3

Amir Amintabar, PhD

Outline

- 1- Review
- 2-Why we need Web APIs
  - Example:
  - Challenges of launching an online movie store
- 3- Anatomy of Web APIs
- 4- Postman
- 5- Intro to RESTful **architecture** style
  - The five key principles of RESTful service-enabled applications
  - Microservices
  - Example of an educational application with Microservice Architecture

Review: Labs

- Lab0: The four review quizzes were based on week1 content: COMP4537-1- review HTML CSS JS Hosting.pptx

- Lab1: Were based on the week 2 of lecture material: COMP4537-2 Architecture JSON LocalStorage.pptx

Review: Labs

- You should have a basic understanding of SQL-based database, JavaScript, HTML and CSS *based on your prior coursework*:

Web Development 1
COMP1537: https://www.bcit.ca/outlines/20211087560/

- Create JavaScript classes and objects.
- Create JavaScript code that handles various types of events on th
- Apply the standard three-tiered web architecture (data, applicati
  app with that architecture.
- Utilize various storage mechanisms on both the client-side and se
  requirements.
- Design and implement a Progressive Web Application (PWA).
- Apply core JavaScript concepts such as functions as objects, vario
  control constructs, and how JSON is utilized.

Web Development 2
COMP2537: https://www.bcit.ca/outlines/20211087580/

Utilize techniques to dynamically serve web content such as image

Perform DOM manipulation via jQuery on the server-side.

Utilize a routing framework on the server-side.

Use asynchronous programming on the client and server

Relational Database Design and SQL
COMP1630: https://www.bcit.ca/outlines/20211086178/

- Use SQL - DDL to implement a relational database.
- Use SQL for data manipulation such as the basic Select statement.
- Use SQL for advanced manipulation such as Group BY, Having, Correlated subqueries.
- Create stored procedures and triggers.
- Discuss techniques for transaction management and concurrency control.

earning Resources

Review:

- We face a choice: do we pause and potentially bore those who are already familiar with the content from previous courses, or do we move forward and assume that students without the background will catch up? How can we find a compromise here?

2021 COMP1537, https://www.bcit.ca/outlines/20211087560/

Review: JSON

- Last we learned JSON is a way of **converting an object into a string**, so that we can serially store it or send it to other machines on the net. Then to pars it back to the same object

- // Object to String

```
let myJSON = JSON.stringify(myObj);
```

- // String to Object
- `let myObj = JSON.parse(myJSON);`

- Q1: what are the JavaScript data types?
- Q2: what would be the result of typeof (null) ?

Review: Web Storage

- We also learned about web storages
- LocalStorage and sessionStorage
- They are more **secure** compared* to cookies
- local storage is **permanent**, but session storage **expires**
- All pages, from same origin, can store and access the same data **within same browser** on your computer of course
- Web storage only handles string key/value pairs.
- **Q:** what could be the variable value in the code snippet below?
- `let value = sessionStorage.getItem("key");`
- A: it returns string…. or null
- * secure compared to cookie, not fully secure.

- **Q:** Now that we can only store strings in localStorage, is there a way to store an object in localStorage?

- A: yes, via JSON

Review: JavaScript (from COMP4537-1- review HTML CSS JS Hosting.pptx)

- Let
- const
- var
- Hoisting
- Set time out

Review: JavaScript (from COMP4537-1- review HTML CSS JS Hosting.pptx)

- Q: what does

typeof []

return?

- **var**: Declares a **function-scoped** variable.
- **let**: Declares a **block-scoped**, variable.
- **const**: Declares a **block-scoped** const. const variables cannot be *re-assigned* after assigned once.

Scope of Declarations by var, let, const:

```
{
  var a = 10;
}
console.log(a);
```

```
{
  let a = 10;
}
console.log(a);
```

```
{
  const a = 10;
}
console.log(a);
```

What does console.log(a) print on console ?

10	**Uncaught ReferenceError: a is not defined**	**Uncaught ReferenceError: a is not defined**
Function scoped	Block Scoped	Block Scoped

Q: What will console print in each of these scenarios ?

A

```
console.log (n)
var n = 150
```

B

```
var n = 150
console.log (n)
```

C

```
console.log (n)
let n = 150
```

variable hoisting (only for variables declared by “var”)

- The JavaScript engine treats all variable declarations using “var” as if they are declared at the top of a functional scope (if declared inside a function, if not as if they are declared at top of the script)

```
<script>

    console.log(myVar);// will not issue error
    var myVar;
    myVar = 140;
</script>
```

JS engine moves declarations using **var** to top

Will not issue error, even though myVar was declared next line
It will simply print undefined

```
<script>

    console.log(myLet);// will issue error
    let myLet;
    myLet = 140;
</script>
```

Will issue error and terminate execution.
JS engine will NOT hoist variables declared by “let”

- Q: When declaring a variable which one you should choose first? const, let or var?

- Q: What if you use a variable without declaring it in JavaScript? (btw worst practice)

A: first const, then let and never var.

A: becomes global scope

=== vs ==

- Q1: Whats the diff between the two?
- The == (or !=) operator performs an automatic type conversion if needed.
- The === (or !==) operator will not perform any type conversion. It first compares if both sides have the same type, then compares the values

- Q2: Can a function return another function in js?

- Q3: Can a function accept another function as arguments in js?

Order of execution … Event Queue

- What would be the output of this script?

```
<script>
    console.log(1);
    setTimeout(function () {
        console.log(2);
    }, 1000);
    console.log(3);
</script>
```

1
3
2

- What if we replace 1000 with 0 ?

1
3
2

Study: https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

Code of conduct reminder

- Students' work will not be checked before assignment submission if the request is very general.

- Statements such as "I did not read the lecture notes or lab description etc." are not accepted as valid excuses for exemption from meeting lab requirements

- … If students miss a class/lecture/lab, it is their responsibility to determine what was missed … https://www.bcit.ca/outlines/20233046882/

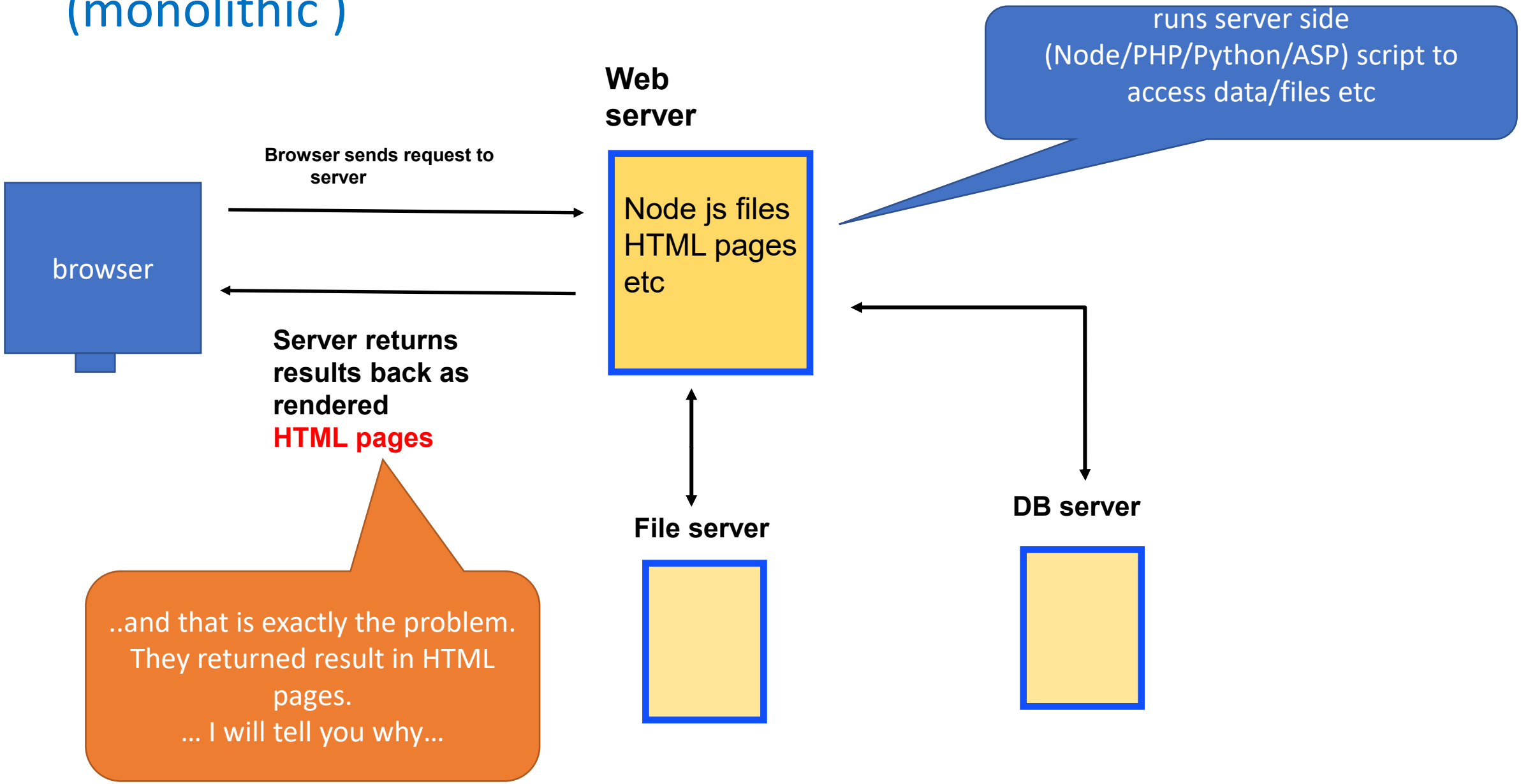
Why do we need API servers?

- Let's start by asking a question:
  Q: What does a web server return when browsers send a request like these?
- Examples:
- www.who.int/index.**html**
- https://wordpress.com/index.**php**

A: they return **HTML pages** and their JavaScript and CSS

Press ctrl+u to see it for yourself

The architecture of small web applications (monolithic)



Imagine

You want to launch an online Movie store

What are the main challenges?

You need a database of all existing movies.
You have two choices:
1- Create your own database which takes years!
2- or getting from websites like IMDB.com which already contain what you need.
… 100,000 move titles

Online movie store

challenge 1, HTML scraping

… so you chose using IMDb's data.

however

IMDB does not offer you any data format other than HTML which contains CSS, Tags etc.

But

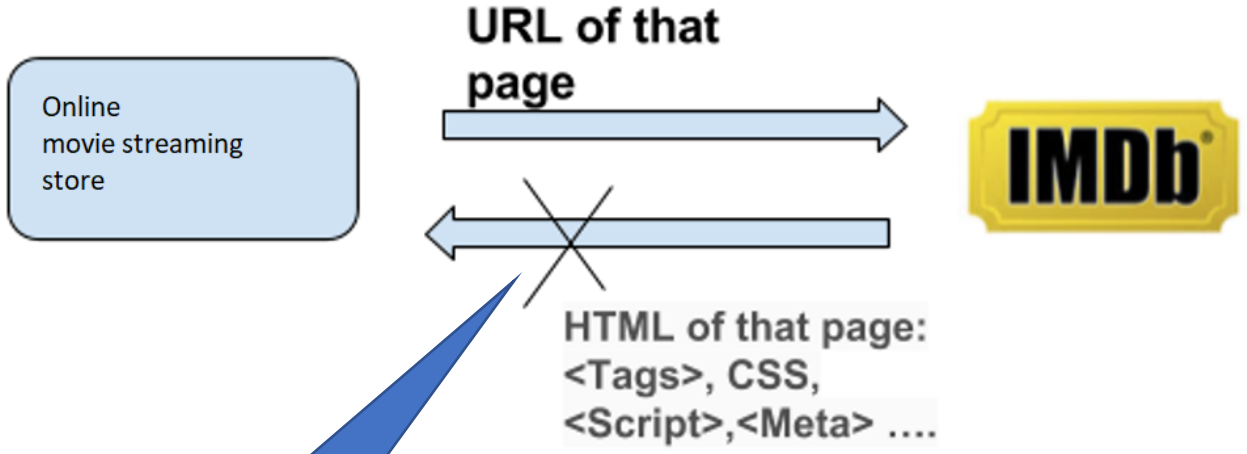
you don't need all of that.. So you need to extract what you need (**Screen Scraping**!!!) you need to search for the Actor names, Date was published etc inside the HTML!...

Imagine that you need to extract movie titles by HTML scaping of IMDB web pages!

challenge 2, what if UI of IMDB pages change?!

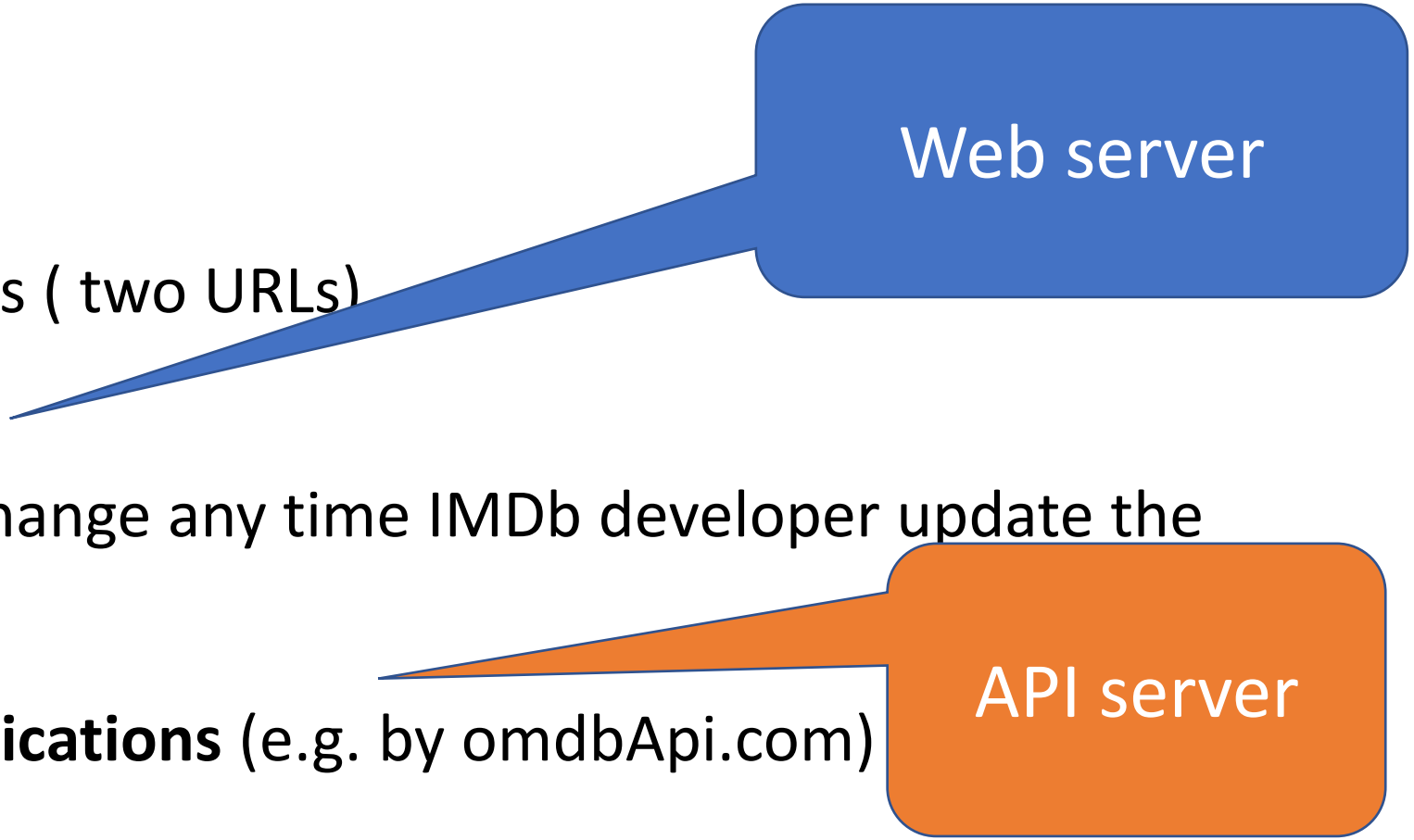
IMDb **decided to change** the look of their pages! Thus the HTML pages change.

As a result your code which was written for that particular HTML will no longer work!



What if they change the design of their pages and your HTML scraper can no longer find movie title in same location at the HTML file!

Solution



IMDb provides two services (two URLs)

1- For **browsers and users**.

Returns HTML pages

Returning result can change any time IMDb developer update the website

2- For **data concumer applications** (e.g. by omdbApi.com)

Returns Json/XML

The rule of thumb is OMDb does not change what this url return

Calling API

http://www.omdbapi.com/?t=star+wars&y=2015&apikey=1668f32d


```
{
  "Title":"Star Wars: The Force Awakens",
  "Year":"2015",
  "Rated":"PG-13",
  "Released":"18 Dec 2015",
  "Runtime":"136 min",
  "Genre":"Action, Adventure, Fantasy",
  "Director":"J.J. Abrams",
  "Writer":"Lawrence Kasdan, J.J. Abrams, Michael Arndt, C
  "Actors":"Harrison Ford, Mark Hamill, Carrie Fisher, Ada
  "Plot":"Three decades after the Empire's defeat, a new t
  "Language":"English",
```


What API server returns (JSON)

vs

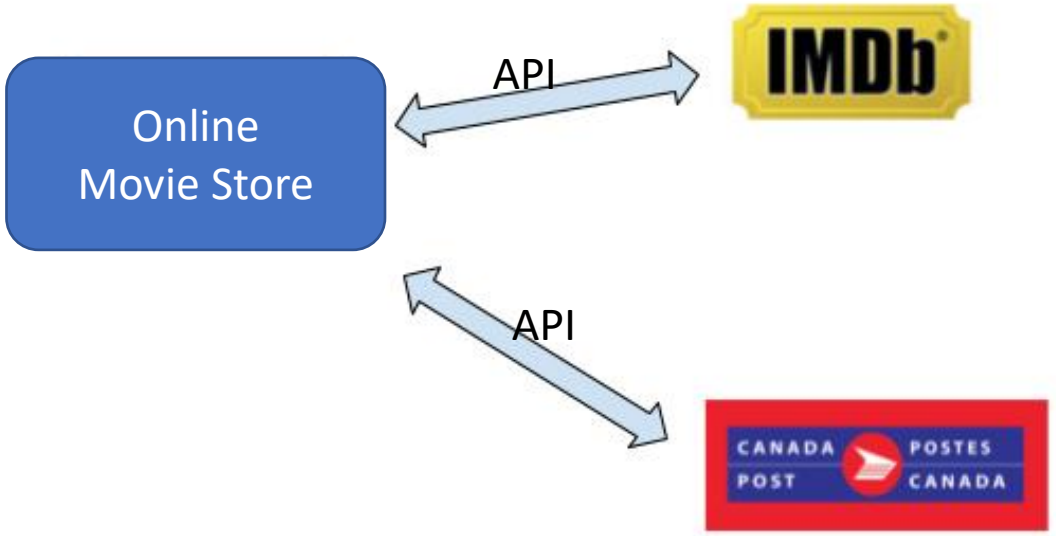
Browsers: Rendering HTML pages

https://www.imdb.com/title/tt2488496

```
<script>
    if (typeof uex == 'function') {
      uex("ld", "LoadIcons", {wb: 1});
    }
</script>

        <meta property="pageId" content="tt2488496" />
        <meta property="pageType" content="title" />
        <meta property="subpageType" content="main" />


        <link rel='image_src' href="https://images-na.ssl-images-amazon.com/images
        <meta property='og:image' content="https://images-na.ssl-images-amazon.com

        <meta property='og:type' content="video.movie" />
    <meta property='fb:app_id' content='115109575169727' />

    <meta property='og:title' content="Star Wars: The Force Awakens (2015)" />
    <meta property='og:site_name' content='IMDb' />
    <meta name="title" content="Star Wars: The Force Awakens (2015) - IMDb" />
        <meta name="description" content="Directed by J.J. Abrams.  With Daisy Ric
threat arises in the militant First Order. Stormtrooper defector Finn and spare pa
        <meta property="og:description" content="Directed by J.J. Abrams.  With Da
new threat arises in the militant First Order. Stormtrooper defector Finn and spar
        <meta name="keywords" content="Reviews, Showtimes, DVDs, Photos, Message B
        <meta name="request_id" content="1WZ0HAN3CMHMGBP4DX2C" />

<script>
    if (typeof uet == 'function') {
      uet("bb", "LoadCSS", {wb: 1});
    }
</script>
  <script>(function(t){ (t.events = t.events || {})["csm head pre css"] = new Date
```

What web server returns (HTML page)

Now with the help of API you can even let your users to track their orders inside your website by partnering with Fedex or Canada post using their APIs.



In case someone needed physical DVD

API

Application Programming Interface

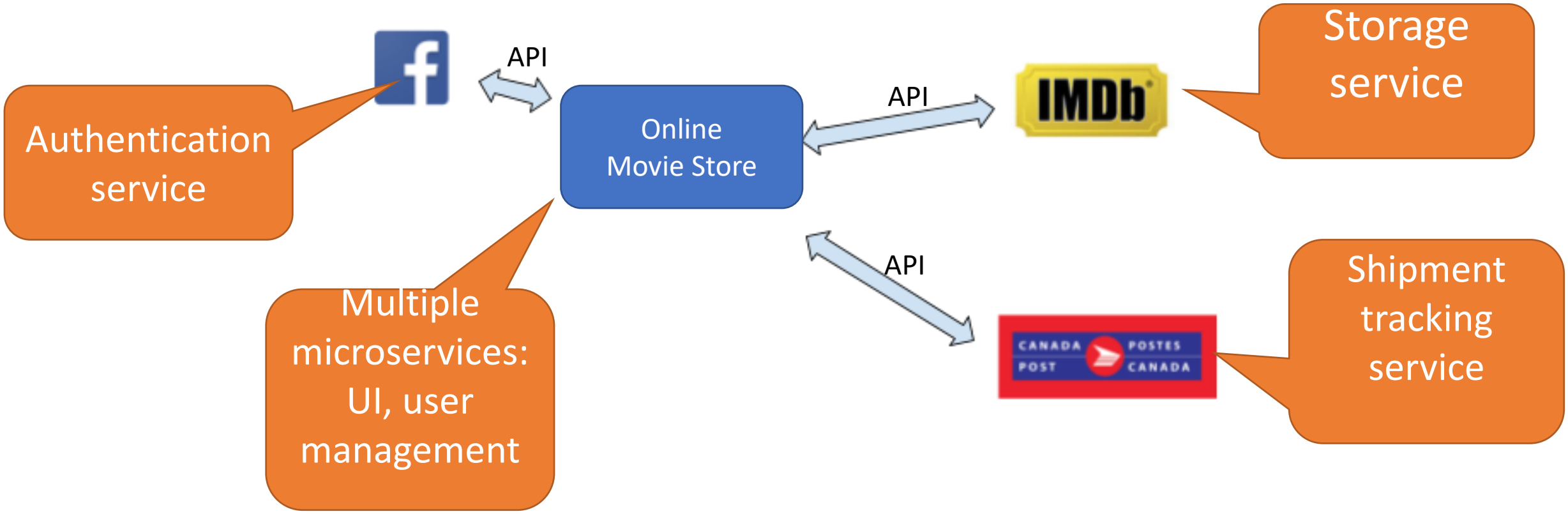
- A user interface to access partners' data and services
- In web development, Web API often refers to the way by which we retrieve information from an online service.

API documentation :

- list of URLs, query parameters etc on how to make a request from the web API server,
- It also inform us what sort of response will be received for each query (json, text file, xml etc)

API and Microservices

You can even let your visitors buy Movie using their facebook account by integrating your online Movie store to facebook



Anatomy of web APIs

Anatomy of Web APIs

1- Headers: The additional details provided for communication between client and server. Some of the common headers are:

Request:
- *API-key*: the subscription key the client. This way the server knows who is making the request.
- *count*: the number of objects you want the
- pageNo: the page number etc

Response:
- *status*: the status of request or HTTP code.
- *content-type*: type of resource sent by server (e.g. html/img/...

2- Data: (also called body or message) contains info you want to send to the server.

Reference: https://www.freecodecamp.org/news/rest-api-tutorial-rest-client-rest-service-and-api-calls-explained-with-code-examples/

Anatomy of Web APIs

3- Endpoint: it is the URL where the API Server is listening to.

4- HTTP Methods: determines the method of communication between client and server? Web API servers implements multiple 'methods' for different types of request, the following are most popular:

- **GET**: **Get** resource from the server.
- **POST**: **Create** resource to the server.
- **PUT**: **Update** existing resource on the server.
- **DELETE**: **Delete** existing resource from the server.

The two most common HTTP methods are: GET and POST.

Reference: https://www.freecodecamp.org/news/rest-api-tutorial-rest-client-rest-service-and-api-calls-explained-with-code-examples/

GET vs POST method

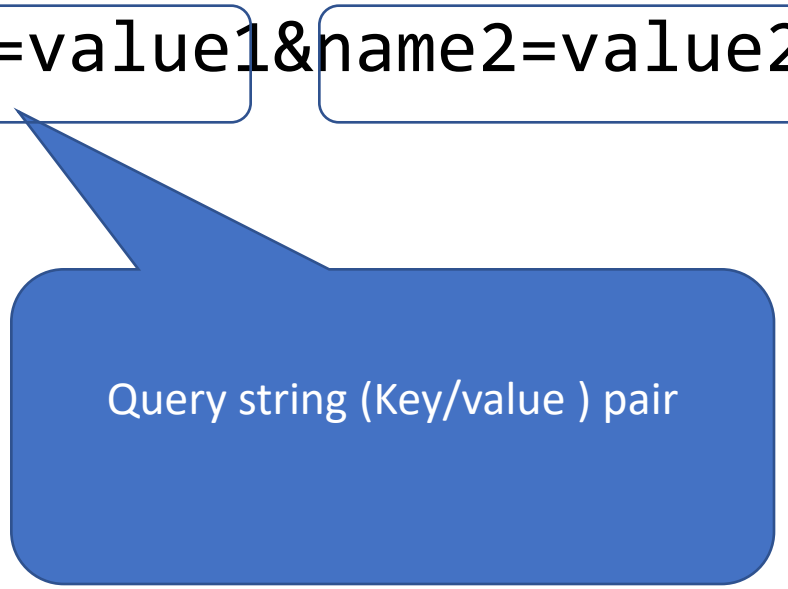
The two most common HTTP methods are: GET and POST.

GET

- GET is used to request data from a specified resource.

- the query string (name/value pairs) is sent in the URL of a GET request

- `/test/demo_form.php?name1=value1&name2=value2`



GET

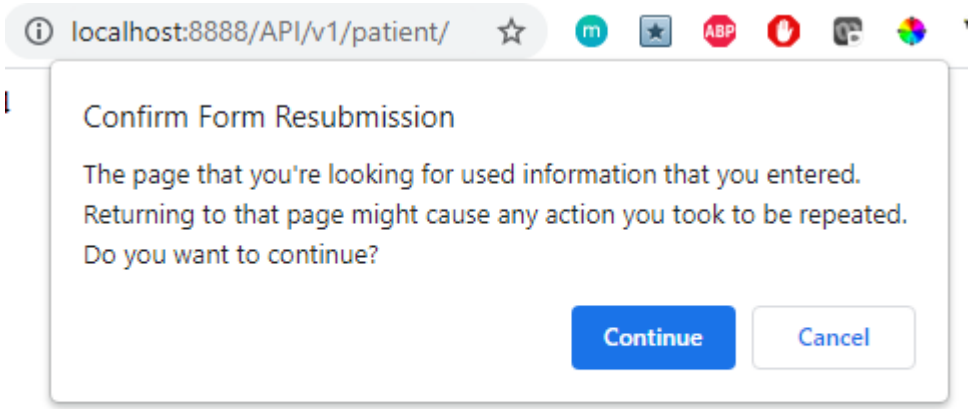
- GET requests can be **cached**
  - GET requests are re-executed but may not be re-submitted to server if the HTML response is already stored in the browser cache.
- GET requests **remain** in the browser **history**
- GET requests can be **bookmarked**
- GET requests should never be used when dealing with sensitive data
  - **Q**: Why?
- GET requests have **length restrictions** on data size
- GET requests **should** only be used **to request data** (not modify)

POST

- POST is used to send data to a server to create/update a resource

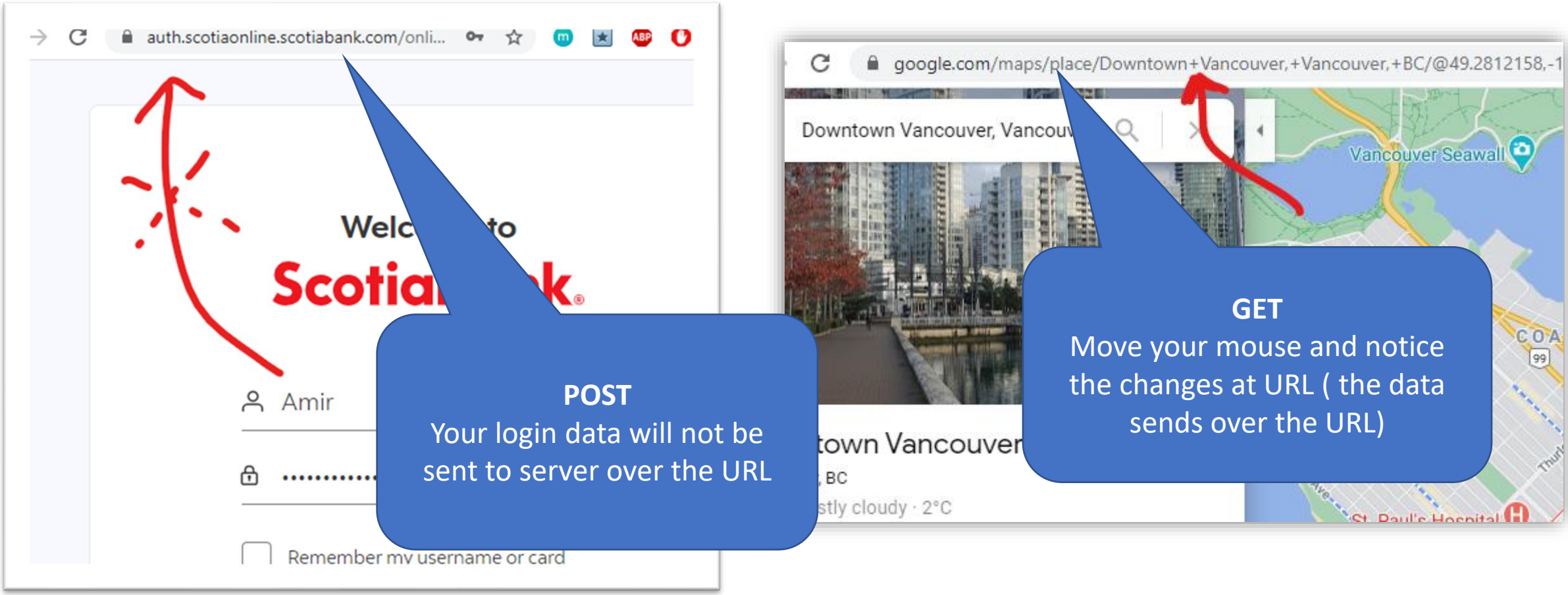
- The data sent to the server with POST is stored in the request body of the HTTP request (not part of the URL)

- POST requests are **never cached**
- POST requests **cannot be bookmarked**
- POST requests **do not remain** in the browser **history**
- POST requests have **no restrictions** on data length
- If refreshing leads to resending a post request (using form), the browser warns the user



In class activity

- Which of the following web API request should be handled by GET which one should be handled by POST?
- 1- Google map search
- 2- Login to your online banking system



In class activity

- Q: which method is secure, GET or POST?
- A: none! The data is not encrypted by default when sent over GET or POST, you need to encrypt data (or use secure protocols such as https etc)

	GET	POST
Pressing BACK button or browser/Reload (refreshing page)	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security (none are secure!)	GET is less safe compared to POST because data sent is part of the URL Never use GET when sending passwords	POST is a **little** safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Ref:w3school

Part 4

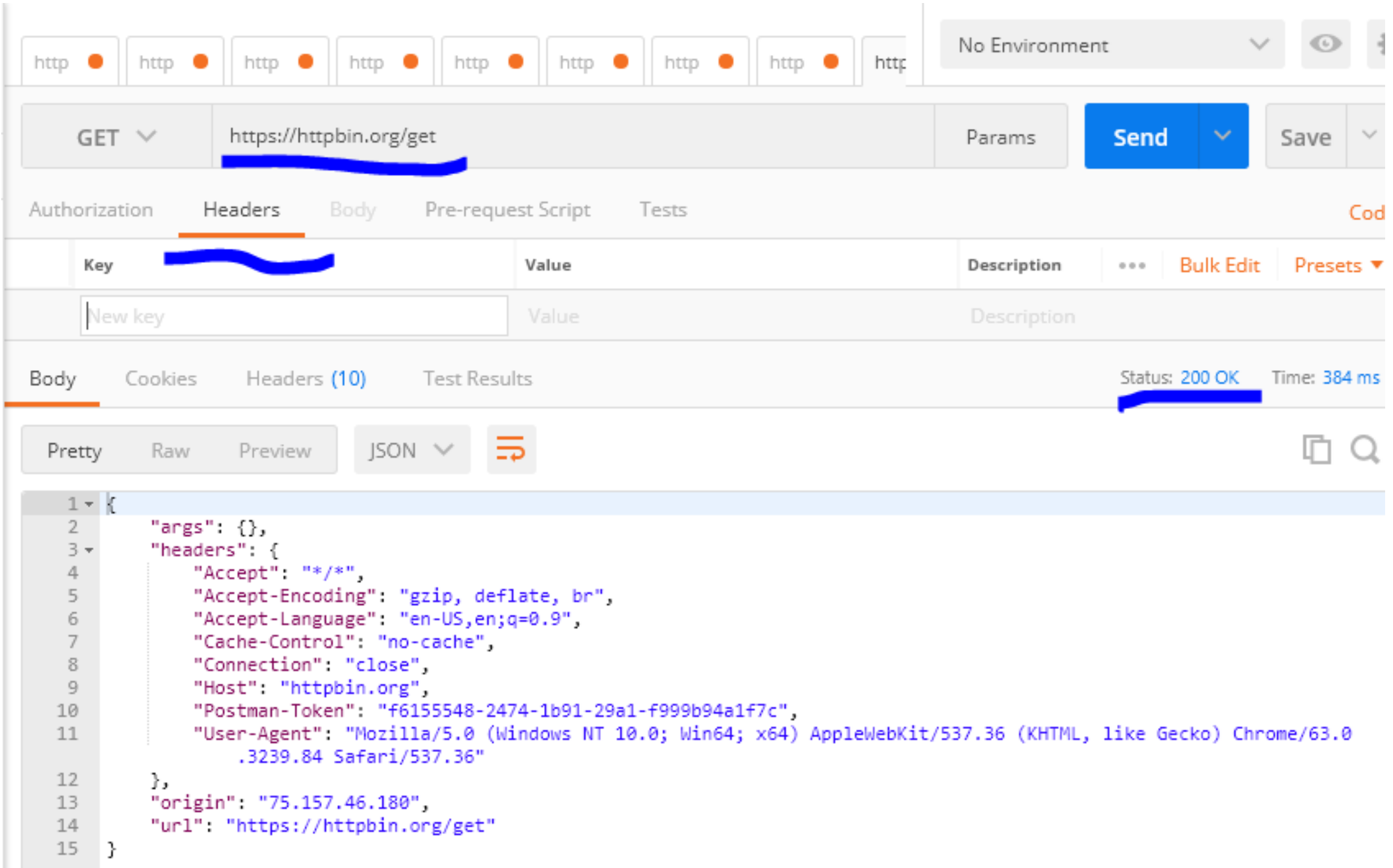
How to test and try an API?

We can use postman

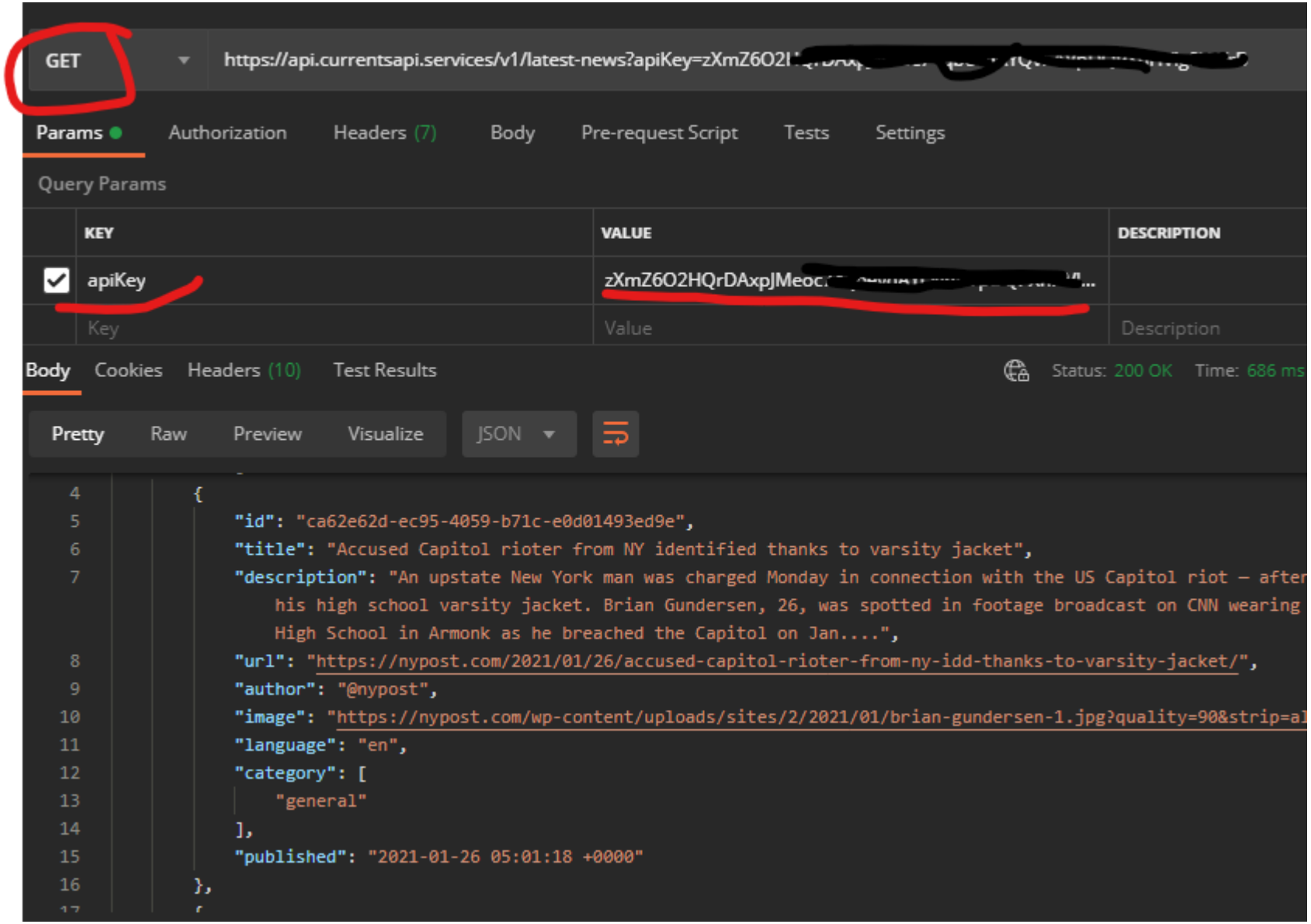
Download: https://www.getpostman.com/

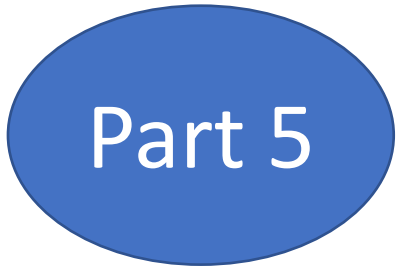
In class activity
Now use postman to try https://httpbin.org/get API

try these
https://github.com/public-apis/public-apis
Or even larger set at:
https://rapidapi.com/marketplace


http
No Environment
GET
https://httpbin.org/get
Params
Send
Save
Authorization
Headers
Body
Pre-request Script
Tests
Key
Value
Description
Bulk Edit
Presets
New key
Value
Description
Body
Cookies
Headers (10)
Test Results
Status: 200 OK
Time: 384 ms
Pretty
Raw
Preview
JSON
{
"args": {},
"headers": {
"Accept": "*/*",
"Accept-Encoding": "gzip, deflate, br",
"Accept-Language": "en-US,en;q=0.9",
"Cache-Control": "no-cache",
"Connection": "close",
"Host": "httpbin.org",
"Postman-Token": "f6155548-2474-1b91-29a1-f999b94a1f7c",
"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0
.3239.84 Safari/537.36"
},
"origin": "75.157.46.180",
"url": "https://httpbin.org/get"
}

Example 2: https://currentsapi.services/en/docs/





What is RESTful API?

- It's service follows RESTful Architectural style

RESTful Architecture

- **REST: Representational** State **Transfer** architectural design
- Is an architectural style; we will see
- how this **stateless** model helps **applications to scale easily** and
- how it separates data preparation and data consumption.

- Microservices (as a way of implementing the backend of RESTful API servers)
- Architects have started introducing the concept of microservices, aiming to reduce the complexity in designing by improving modularity
- systems by splitting the core components into small and independent pieces called Microservice that simply do a single task.

- Relation between Microservices and RESTful API:
  - The Microservices communicate via RESTful API calls

- Microservices – These are small and lightweight services that execute a single functionality. The Microservices Architecture framework has a number of advantages that allows developers to not only enhance productivity but also speed up the entire deployment process.

- The components making up an application build using the Microservices Architecture aren’t directly dependent on each other. As such, they don’t necessitate to be built using the same programming language.

- Therefore, developers working with the Microservices Architecture are free to pick up a technology stack of choice. It makes developing the application simpler and quicker.

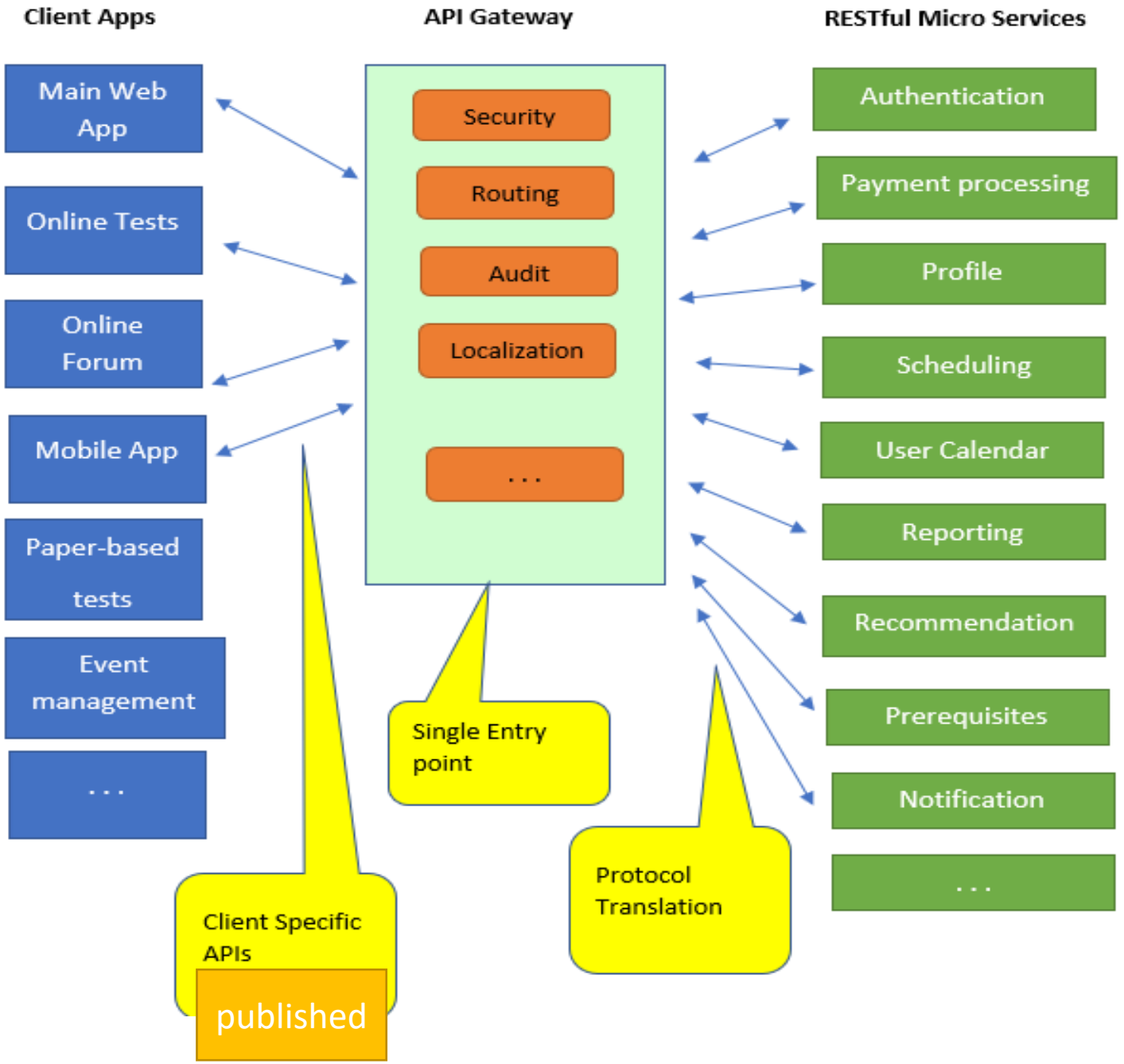
Example: Micro-service architecture diagram

Q: Guess application this diagram represent?
Online store? CRM?

RESTful API

Remarks:

1. Services communicate via API calls
2. Every request goes through API gateway
3. Gateway is responsible for routing, authentication, Auditing etc
4. Client specific APIs are the one published to developers
5. Each microservice could be developed in a different technology stack and could be hosted in a server geographically separated from other services.



RESTful is not an architectural pattern!

- REST is not an architectural pattern like Microservices or MVC.
- Instead, it's an architectural style or set of constraints for API communication between systems.
- The communication between components in an architectural patterns usually happen via **API** calls (Q: How else the communication can happen?)

RESTful API Servers development principals

- **1. Client–Server**
- **2. Stateless**
- **3. Cacheable**
- **4. Uniform Interface (core constraint)**
- **5. Layered System**

- **Q:** what's the benefit to follow these principles when implementing (coding) our API server?

RESTful API Servers development principals

- **1. Client–Server**
- **Principle:** Separation of concerns — the client handles the user interface/experience, and the server manages data and business logic.
- **Benefit:** Improves portability (UI can change without backend changes) and scalability (servers optimized for logic, clients optimizes for UI).
- **Example: GitHub** — Web app, mobile app, and CLI clients all consume the same server APIs.

- **2. Stateless**
- Each request contains all the info needed; server doesn't store session state.
- Example: If your API server returns pages of a book, then to keep it *stateless* (and RESTful), the client must always specify the exact page. For instance, if you has just asked for page 50, to get content of page 51 , you cannot serve such request from client like **"give me the next page."** The server should respond **"I don't keep track of previous requests — you must tell me which page you want."**

- **3. Cacheable**
- The API server you implement must define if a response is cacheable or not (via HTTP headers like Cache-Control, ETag, Expires).
  the **client or intermediaries** (like browsers, CDNs, reverse proxies) can cache the response and reuse it.

- **4. Uniform Interface (core constraint)**
- Resource Identification via URIs → Each resource has a unique URL (e.g., /users/123).
- Manipulation through Representations → Resources are manipulated via JSON/XML representations.
- Self-Descriptive Messages → Requests and responses include metadata (e.g., HTTP verbs, headers).
- HATEOAS (Hypermedia as the Engine of Application State) → Responses include links for discoverability.(Q? I don't know what is means!!!)
- **Example**: GitHub REST API — /repos/{owner}/{repo}/issues returns issues plus links to related actions (close, comment, assign).

- **Benefit**: Simplifies interactions and makes APIs predictable.
- **Example**: GitHub REST API
- To get a user: GET /users/octocat
- To get that user's repos: GET /users/octocat/repos
- To get a specific repo: GET /repos/octocat/hello-world
- 👉 You don't need to read special documentation for every endpoint — the pattern is consistent and predictable.
- Nouns (resources) are always in the URL.
- Verbs (actions) are always HTTP methods (GET, POST, PUT, DELETE)

- **5. Layered System**
- **Principle:** The system can be composed of multiple layers (e.g., client, proxy, gateway, server). Clients don’t know if they are talking to the origin server or an intermediary.
- **Benefit:** Adds flexibility, scalability, and security. Enables CDNs, load balancers, and gateways without changing the client.
- **Example: Amazon Web Services (AWS API Gateway)** — sits between clients and microservices, handling routing, throttling, and security.

Benefits to making our API server RESTful?

- Client–Server → separation & evolvability (GitHub)
- Stateless → scalability & reliability (Netflix)
- Cacheable → performance & efficiency (Twitter)
- Uniform Interface → simplicity & predictability (GitHub API)
- Layered System → flexibility & security (AWS API Gateway)
- Code-on-Demand → extensibility (Web apps delivering JavaScript)

Resources

- https://www.w3schools.com/
- RESTful Web API Design with Node.js 10, Third Edition, Valentin Bojinov
- chatGPT