Fetch and REST API

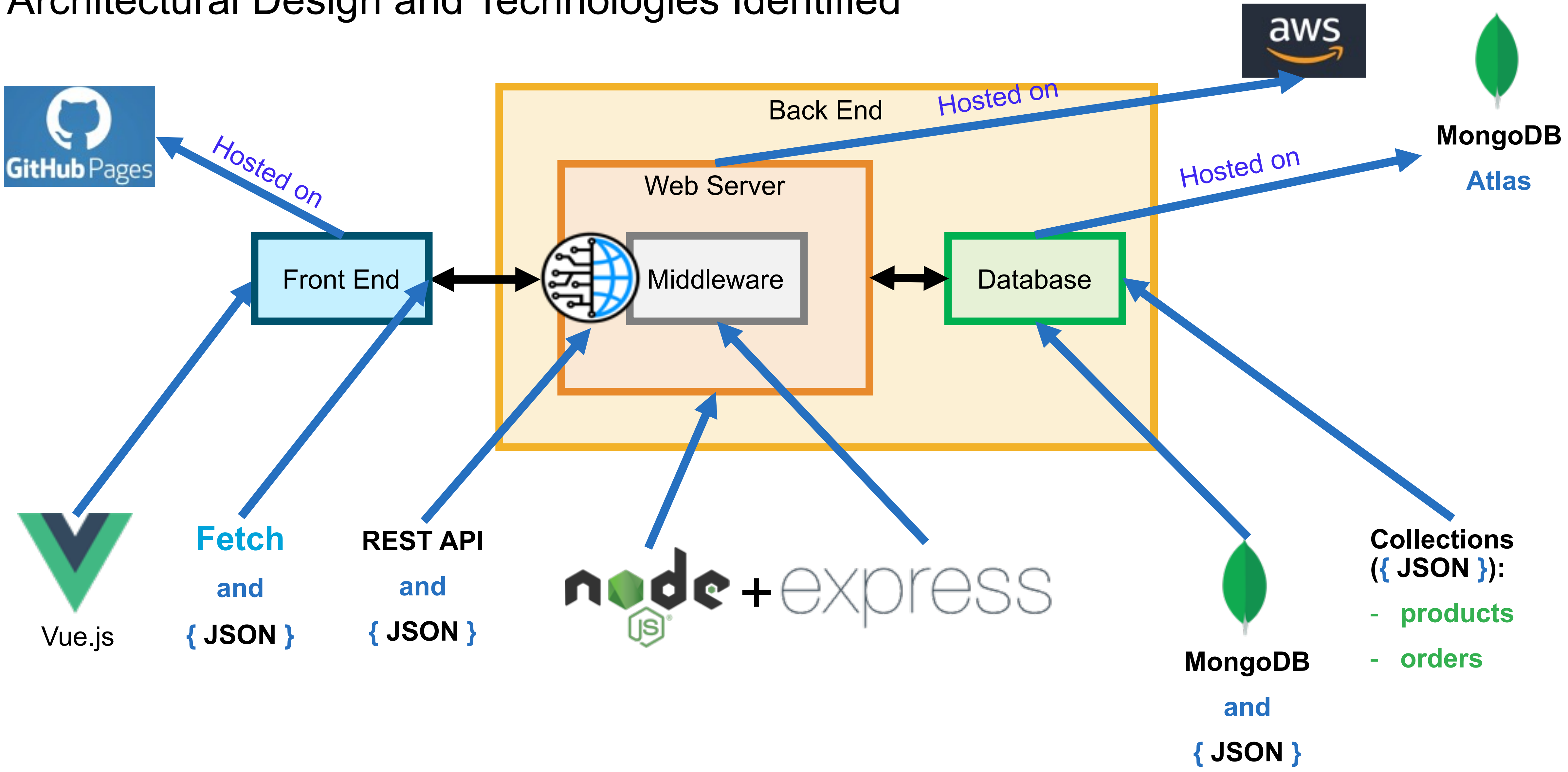
Outline and Learning Objectives

- **The Big Picture:**
  - to understand the overall architectural design, and technologies individuated for the technological solution
- **Fetch:**
  - to understand different solutions for Client-Server Communications
  - to master master the Fetch technology for retrieving data from the server
  - to learn what is CORS and how to manage configure the server accordingly
- **REST API:**
  - to learn the basics of REST Services and how design/implement them with Express.js
  - To learn an initial approach to test REST Services
- **CourseWork 2 (CW2) Requirements**
- **Suggestions for Reading**

The Big Picture

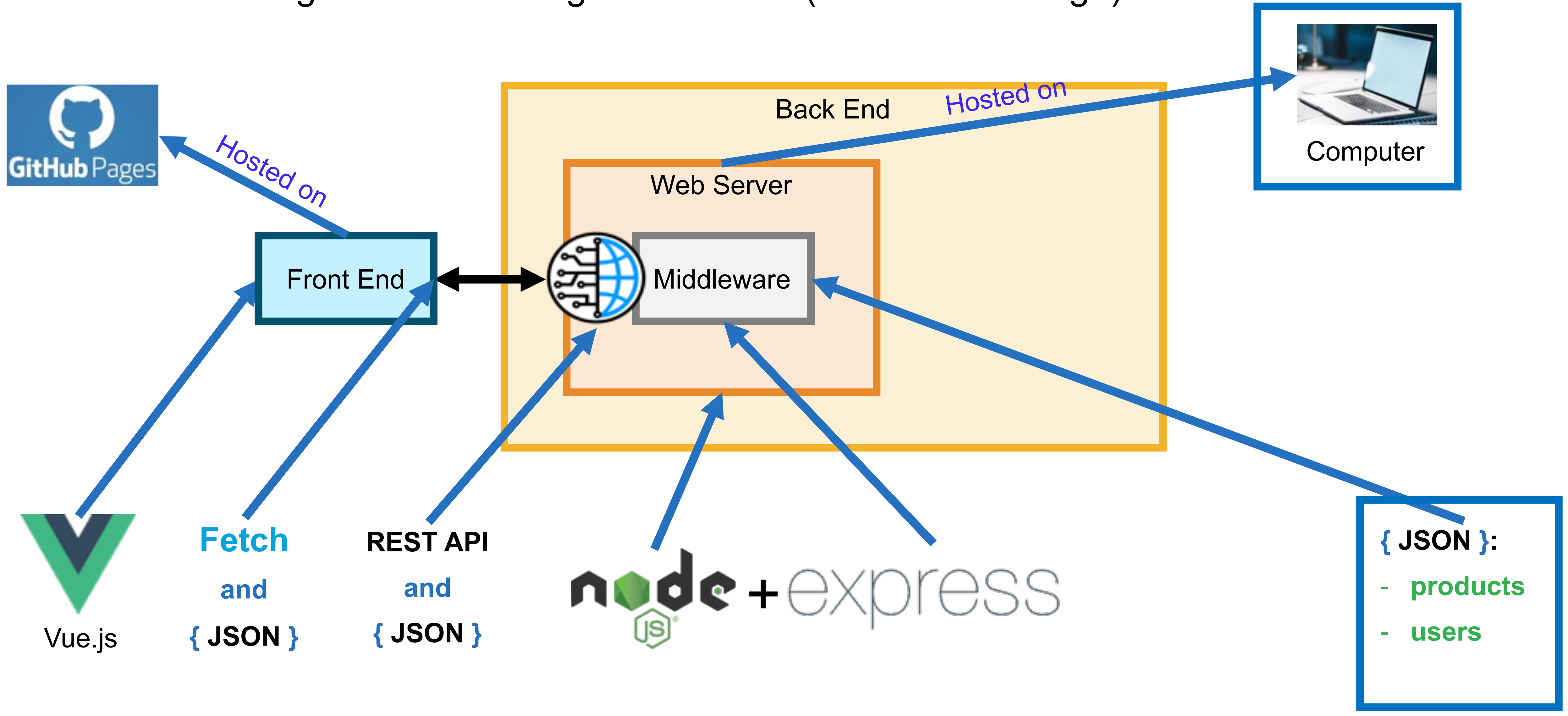
The Big Picture: Architecture and Technologies

Architectural Design and Technologies Identified



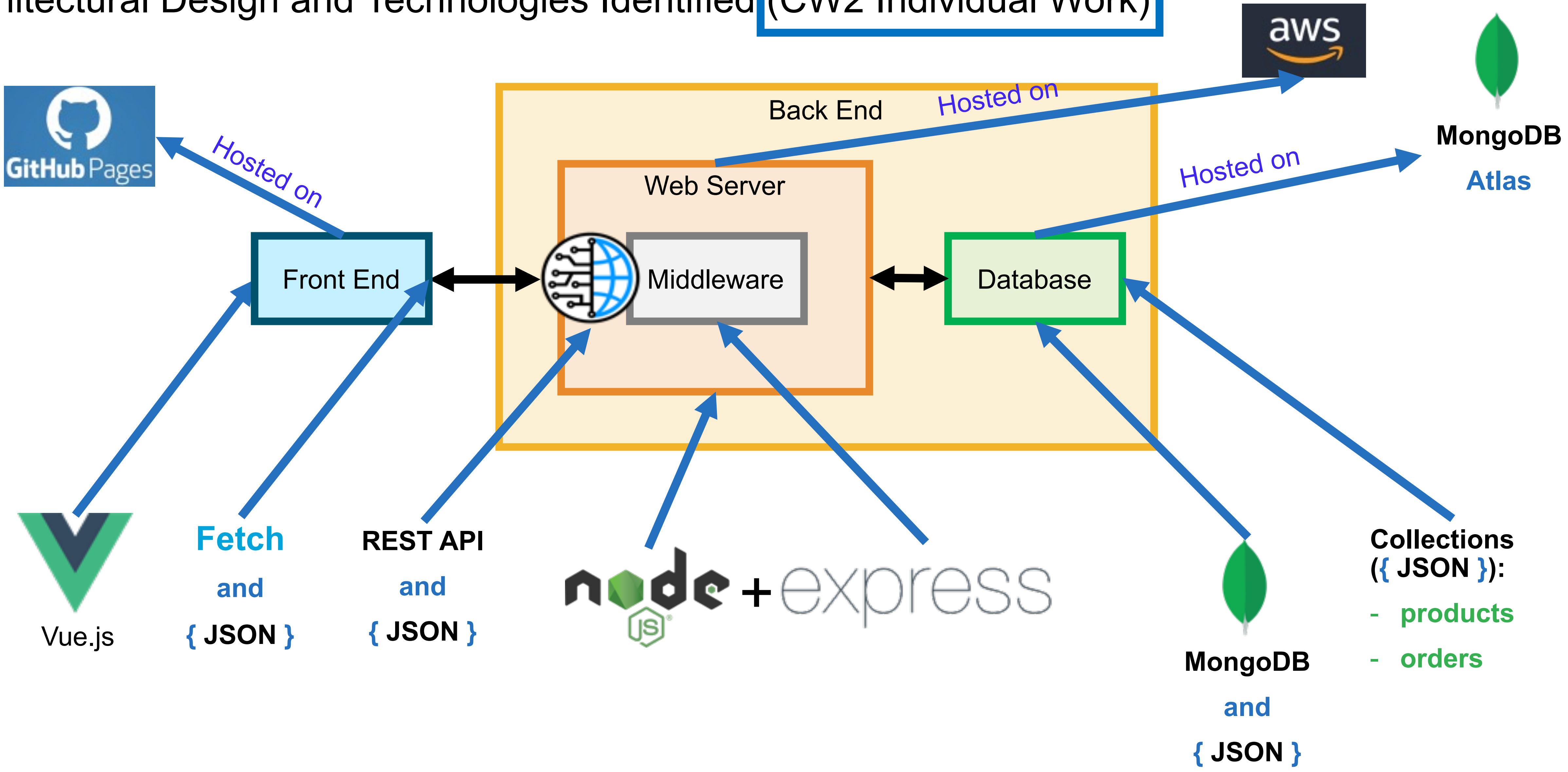
The Big Picture: our Current Stage

Architectural Design and Technologies Identified (our Current Stage)



The Big Picture for CW2 Individual Work

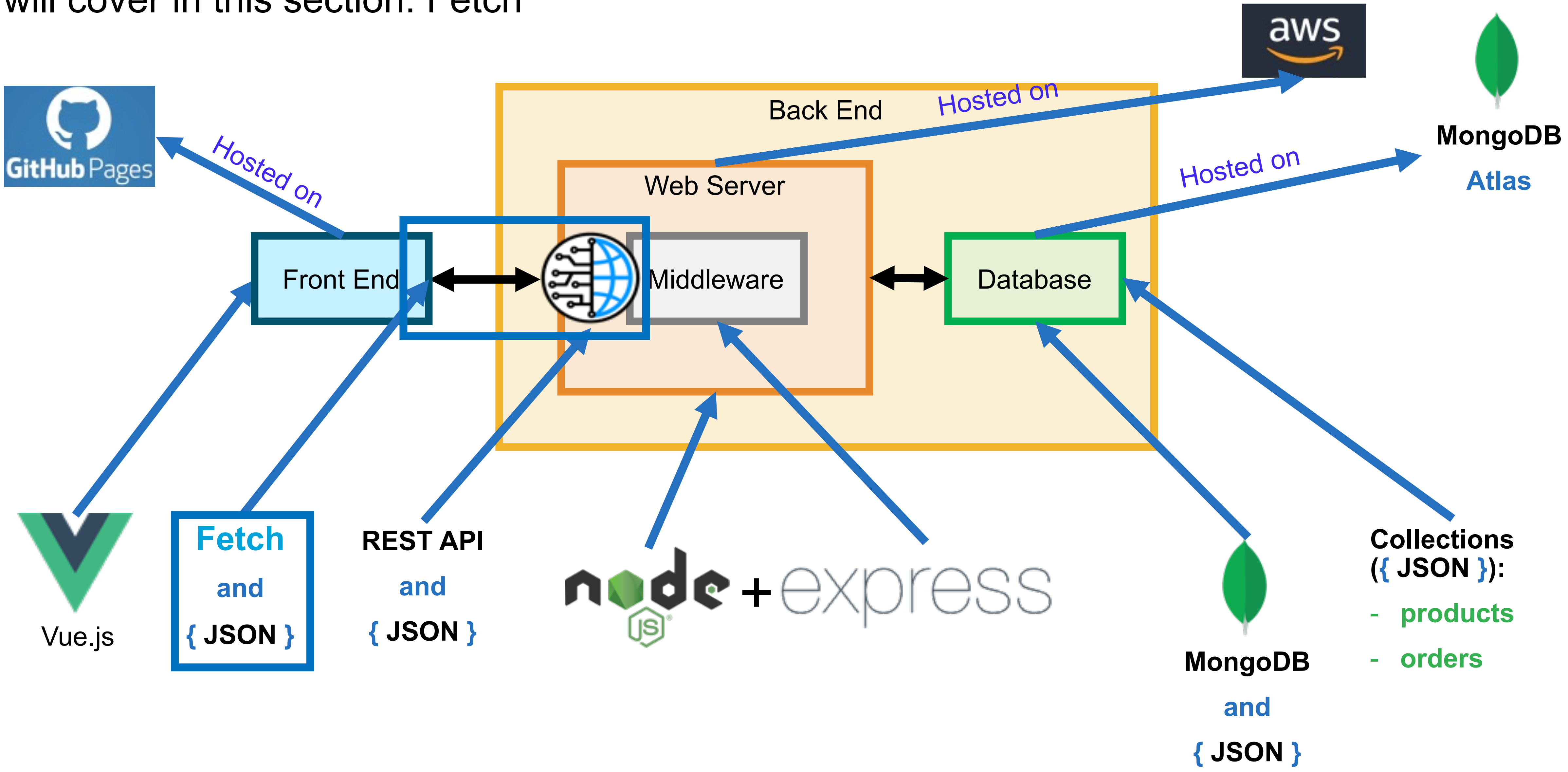
Architectural Design and Technologies Identified (CW2 Individual Work)



Client Server Communication with `Fetch`

The Big Picture and Fetch

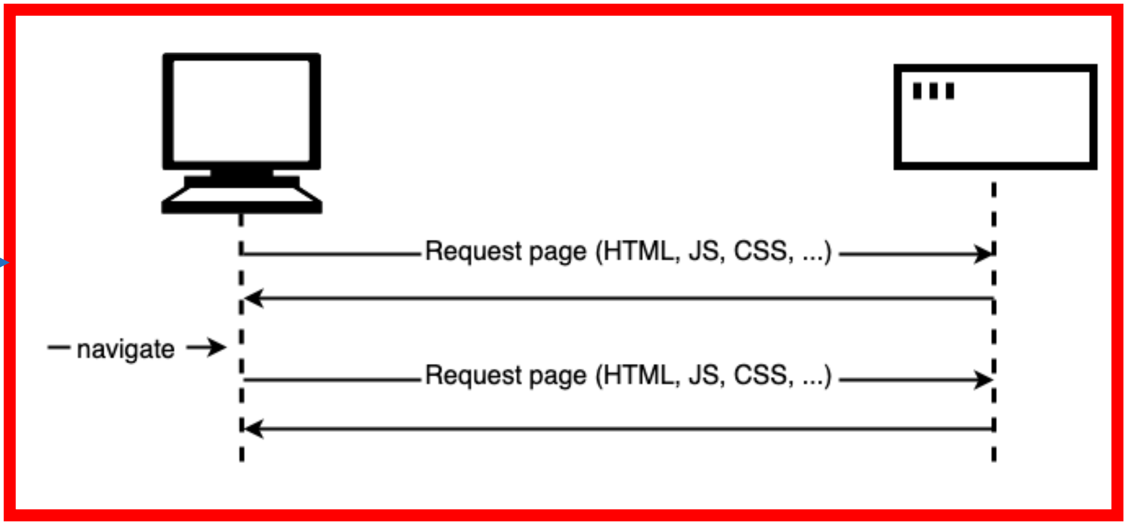
We will cover in this section: Fetch



Client-Server Communication

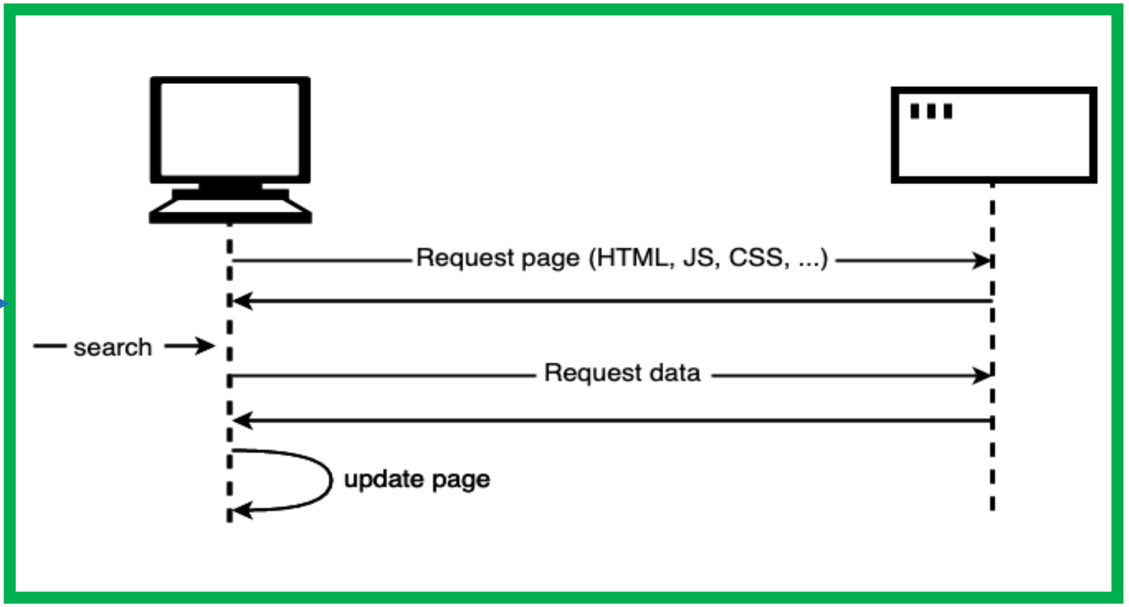
The "old" way (loading the full page every time)

- To display a webpage, a request is sent to the server and **a full page** is returned.
- The problem is that **even to update a small part of the page**, you need to **load the entire page again**.
- This can be wasteful (**download again everything, even what has not changed**) and results in a **poor user experience** (user has to wait for the page to reload).


Request page (HTML, JS, CSS, ...)
navigate
Request page (HTML, JS, CSS, ...)

Asynchronous JavaScript and XML (AJAX) and Fetch

- This led to the creation of **technologies** that allow web pages to **request small chunks of data** (e.g., **XML**, **JSON**) and **display them only when needed**.
- This is achieved by using APIs like `XMLHttpRequest` or — more recently — the `Fetch` API.
- **Pages updates are a lot quicker,** and the user **do not have to wait for the page to refresh**, meaning that the **site feels faster** and offers a **better user experience**.
- **Less data is downloaded** on each update, meaning **less wasted bandwidth**, which can be a **major issue on mobile devices** and in **developing countries** that do not have fast Internet service.



Asynchronous Communication, Fetch and Vue.js

- **Fetching data** from server is an **asynchronous operation**,
  - meaning that you have to **wait for that operation to complete** (e.g., the data is returned from the server) **before you can do anything with that response** (**otherwise**, an **error will be thrown**).
- The `Fetch` API is a **modern replacement** for **Asynchronous Javascript and XML (AJAX)** and `XMLHttpRequest` (XHR); `Fetch` has been introduced to **make asynchronous HTTP requests easier**.

```
data: {
 products: [], //products,
   ...
},
created: function() {
   fetch("http://localhost:3000/collections/products").then(
      function(response) {
         response.json().then(
            function(json) {
               //console.log(json);
               // note that we used 'webstore.products'
               instead of 'this.products'
               webstore.products = json;
            }
         )
      }
   );
},
```

- We need also to **configure our server** for **Cross-Origin Access** (we see this in the **next slide**)

- `created` runs **after the Vue instance is created**
- note that we used `webstore.products` instead of `this.products` to make sure we refer to the Vue instance
- `then()` is a method called on a **Promise** (a modern Javascript feature for performing asynchronous operations) when the result requested is available (**analogue to callbacks behavior**)
- Also `.json()` method returns a **Promise**

Next Slide

Cross-Origin Resource Sharing (CORS)

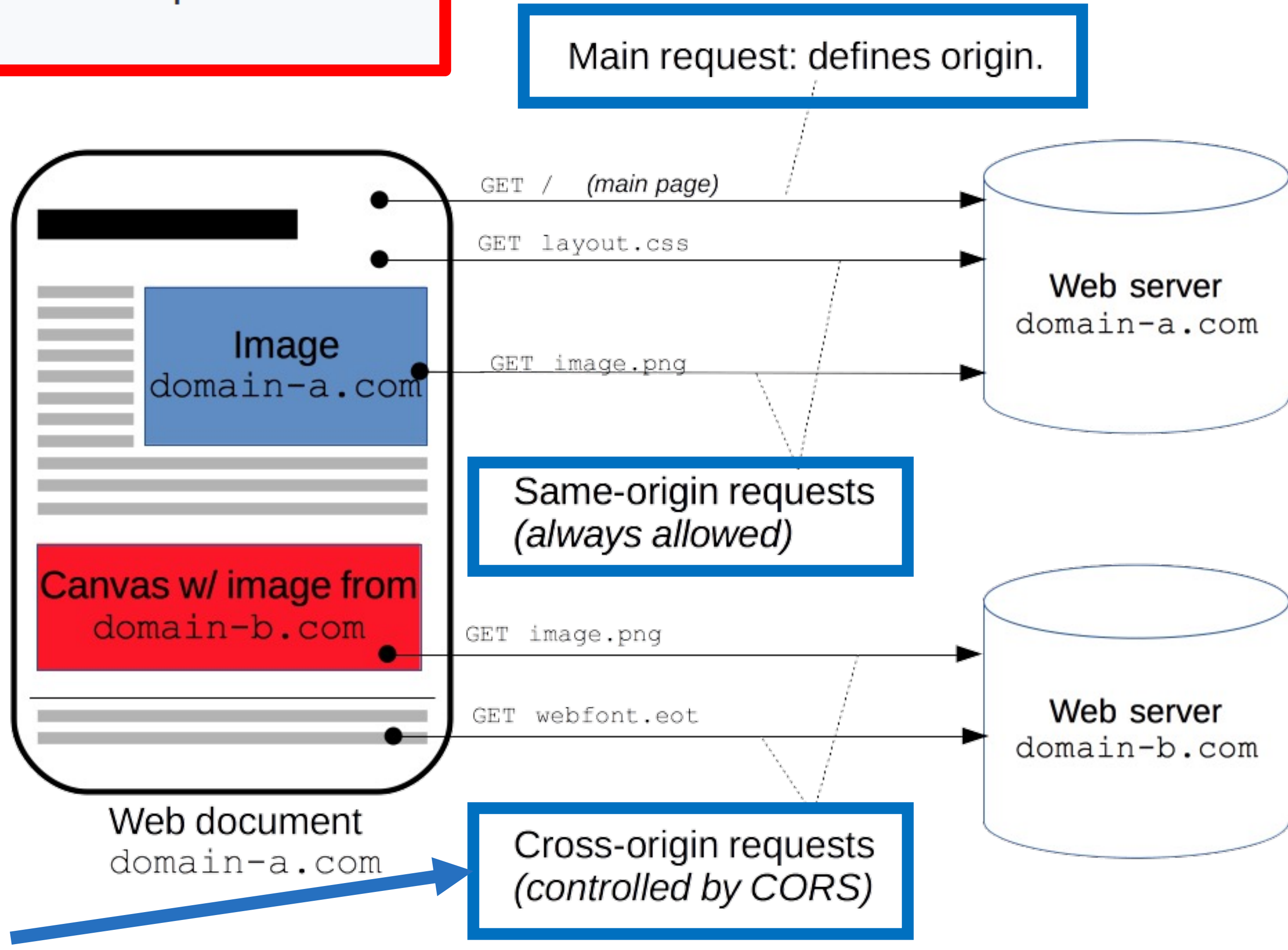
- Without configuring the server for CORS, **fetch will not work**, and you will get an **error** like:

```
Access to XMLHttpRequest at 'http://localhost:3000/' from
origin 'http://127.0.0.1:5500' has been blocked by CORS
policy: No 'Access-Control-Allow-Origin' header is present on
the requested resource.
```

Cross-Origin Resource Sharing (CORS):

- "is an HTTP-header based mechanism that **allows a server to indicate any origins** (domain, scheme, or **port**)
- other than its own **from which a browser should permit loading resources**." (source)
- **This is to prevent running of malicious code from an untrusted server**

- **These requests** can be **allowed** only if the **server is configured properly** for such requests (consider that also **a different port is considered a different origin**, e.g., http://localhost:3000/ and http://localhost:3001/)


Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image domain-a.com
Web server domain-a.com
Same-origin requests (always allowed)
Canvas w/ image from domain-b.com
GET image.png
GET webfont.eot
Web server domain-b.com
Web document domain-a.com
Cross-origin requests (controlled by CORS)

Configuring CORS

- `Express` has a module called `cors` to help manage this
- First, you need to install it with `npm`:

```
npm install cors
```

- Then you need to `require` and `use` it as a `middleware`

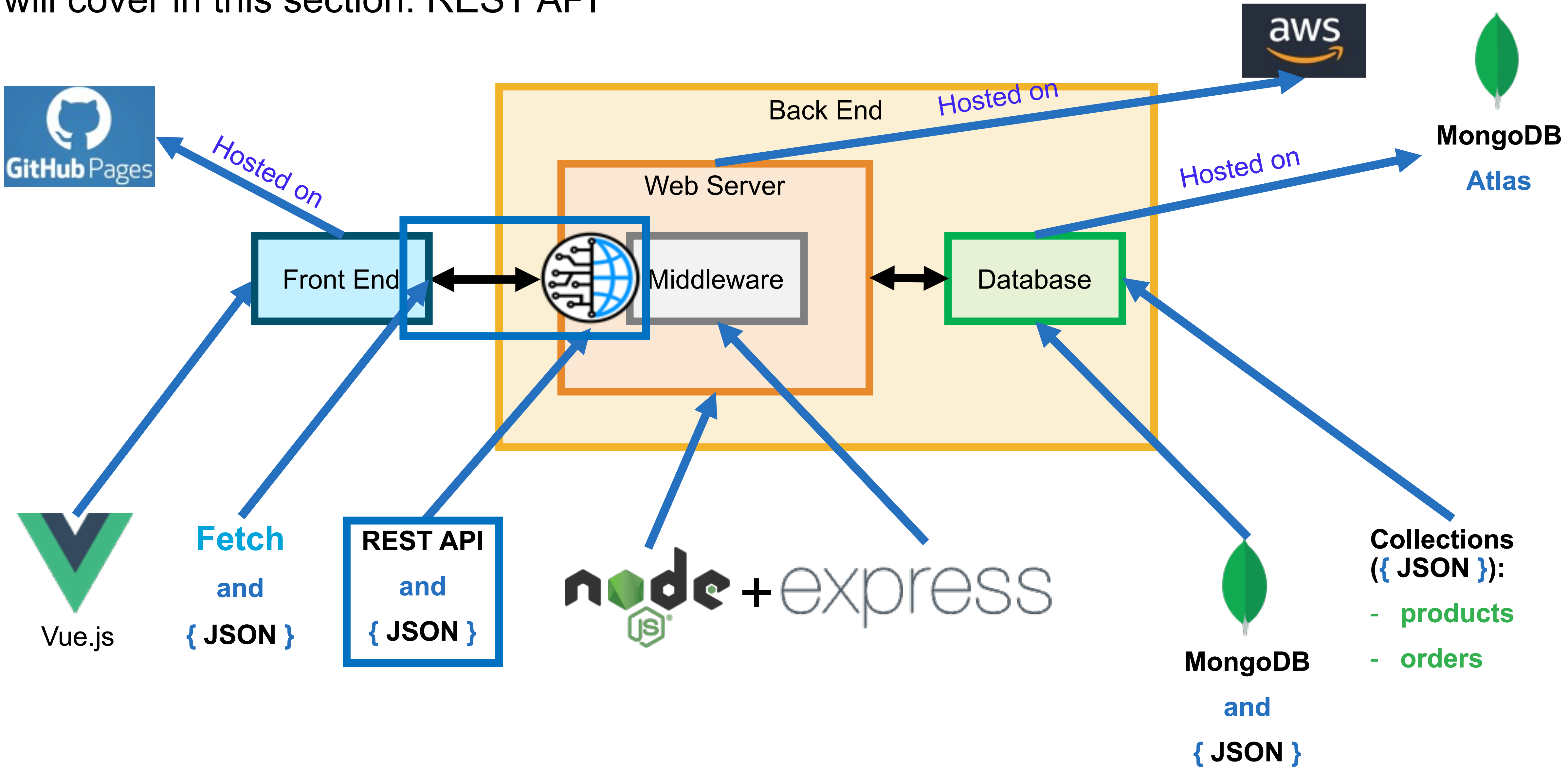
```
const cors = require("cors");
...
app.use(cors());
```

- this by default **enables from any origin** (all requests will be managed), **not very secure**, **this approach will be enough for your coursework**
- **To make it more secure**, you can **configure cors module with options**, for instance by enabling only an origin (e.g., http://127.0.0.1:3001 ; remember that also the port need to be specified, same ip but different ports are considered as different domains);

REST API

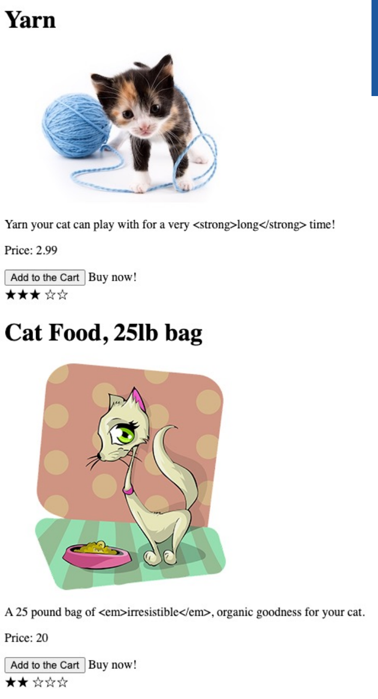
The Big Picture and REST API

We will cover in this section: REST API



REST API Example

Example related to our project-based learning approach and scenario



Yarn

Yarn your cat can play with for a very <strong>long</strong> time!

Price: 2.99

Add to the Cart Buy now!

★★★☆☆

Cat Food, 25lb bag

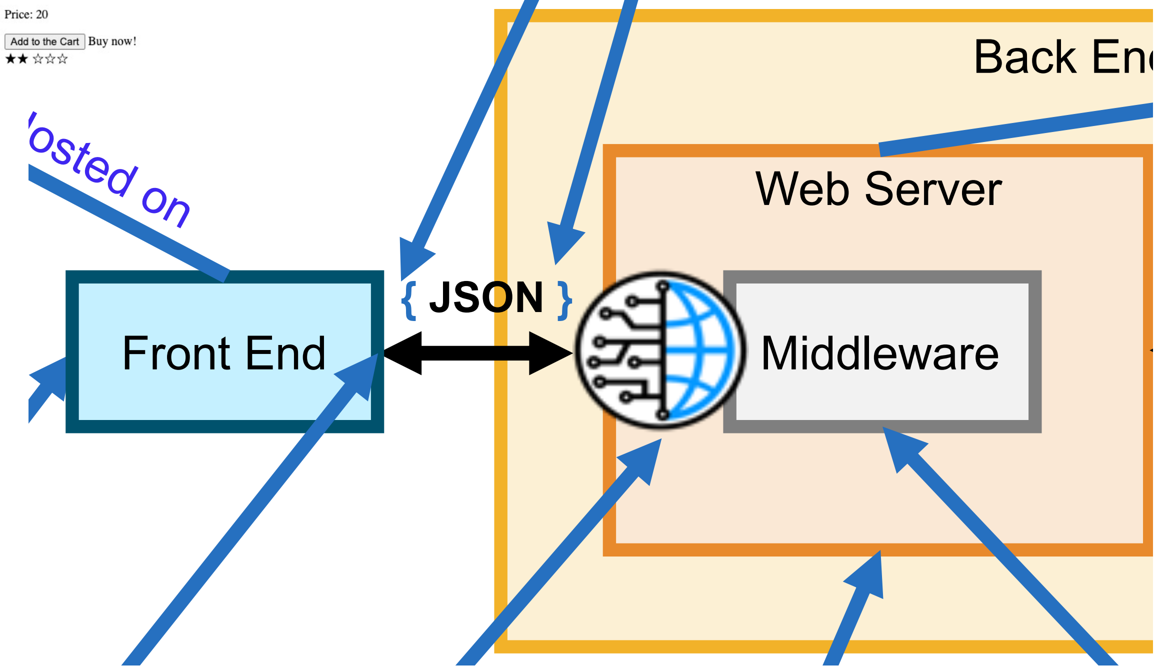
A 25 pound bag of <em>irresistible</em>, organic goodness for your cat.

Price: 20

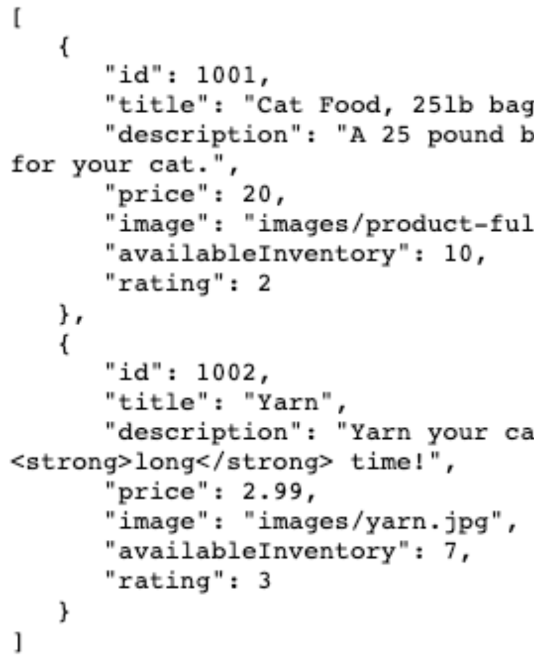
Add to the Cart Buy now!

★★☆☆☆

- Front end performs a fetch by calling a REST Service returning the products
- The service in the middleware manages the request and send back the products as a JSON
- The front end uses the JSON to show the products



{ JSON }

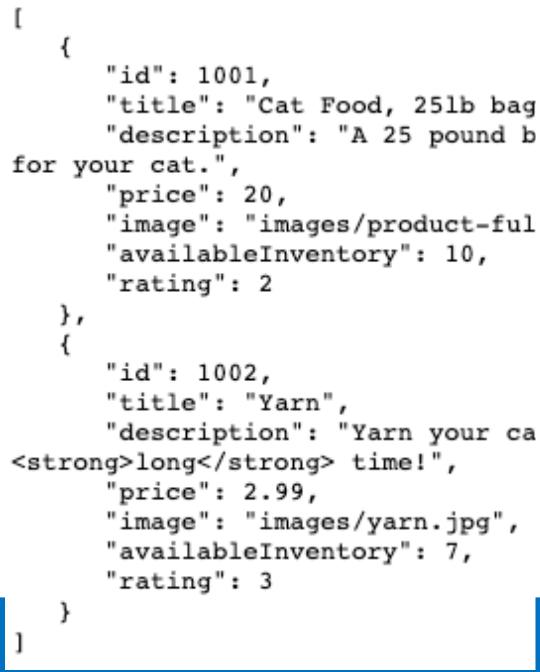


```
[
    {
        "id": 1001,
        "title": "Cat Food, 25lb bag
        "description": "A 25 pound b
for your cat.",
        "price": 20,
        "image": "images/product-ful
        "availableInventory": 10,
        "rating": 2
    },
    {
        "id": 1002,
        "title": "Yarn",
        "description": "Yarn your ca
<strong>long</strong> time!",
        "price": 2.99,
        "image": "images/yarn.jpg",
        "availableInventory": 7,
        "rating": 3
    }
]
```

APIs Do Not Have to Use JSON

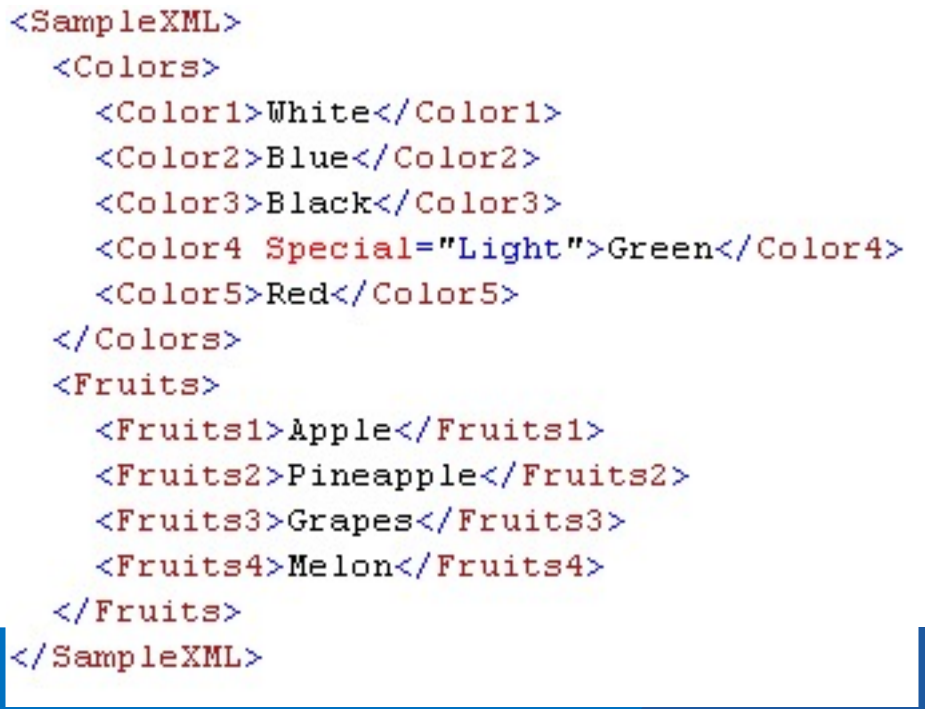
- They can use other data formats (e.g., XML).
- We use JSON here because:
    - JSON can be managed very well by all our technologies (e.g., Vue.js , Express, MongoDB, etc.),
    - In fact, it plays nicely with browser-based JavaScript, and
    - is one of the most popular API choices

{ JSON }



```
[
    {
        "id": 1001,
        "title": "Cat Food, 25lb bag
        "description": "A 25 pound b
for your cat.",
        "price": 20,
        "image": "images/product-ful
        "availableInventory": 10,
        "rating": 2
    },
    {
        "id": 1002,
        "title": "Yarn",
        "description": "Yarn your ca
<strong>long</strong> time!",
        "price": 2.99,
        "image": "images/yarn.jpg",
        "availableInventory": 7,
        "rating": 3
    }
]
```

XML

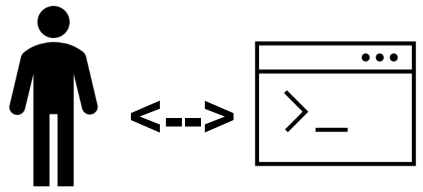


```
<SampleXML>
  <Colors>
    <Color1>White</Color1>
    <Color2>Blue</Color2>
    <Color3>Black</Color3>
    <Color4 Special="Light">Green</Color4>
    <Color5>Red</Color5>
  </Colors>
  <Fruits>
    <Fruits1>Apple</Fruits1>
    <Fruits2>Pineapple</Fruits2>
    <Fruits3>Grapes</Fruits3>
    <Fruits4>Melon</Fruits4>
  </Fruits>
</SampleXML>
```

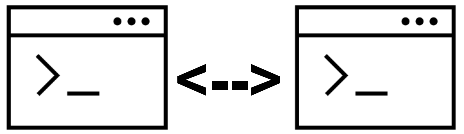
What is an API?

What is an **Application Programming Interface (API)**?

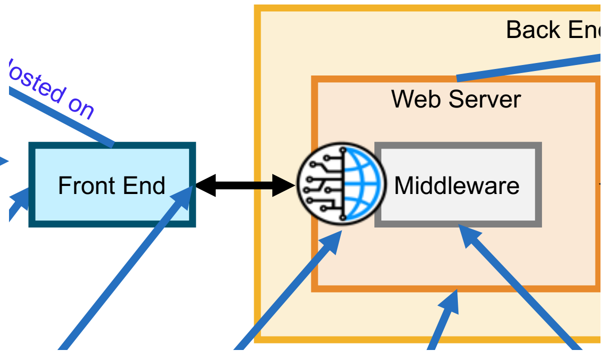
- **Graphical User Interface (GUI)**: **user <--> software**
  - Most software systems have a GUI
  - Except some, like command line tools (e.g., Git and Node.js)



- **Application Programming Interface (API)**: **software <--> software**
  - On the same machine: game <--> graphics driver
  - On different machines: client code <--> server code



- **API is language independent**
  - The two ends can be written in different programming languages
  - Node.js server (JS) <--> JS/Java/Python/C ...



CRUD Application Pattern

Create, Read, Update, Delete (CRUD) APIs

- There is a common application pattern: create, read, update, and delete.
  - It is shortened to CRUD.
- Lots of applications use CRUD. For **example**, a **photo-sharing app where anyone can upload photos**:
  - Users can **upload photos**; this is the **create step**.
  - Users can **browse photos**; this is the **read part**.
  - Users can **edit photos**; this would be an **update**.
  - Users can **delete photos** from the website. This would be, well, a **delete**

HTTP Methods

HTTP verbs (also known as HTTP methods)

- a client sends an HTTP request to the server;
- The request has a method;
- The server sees that method and responds accordingly.
- Common methods: GET, PUT, POST, and DELETE

GET (For CRUD API this is equivalent to: READ)

- The **most common** HTTP method anyone uses.
- As the name suggests, it **gets resources**.
  - When you **load the homepage**, you GET it.
  - When you load an image, you GET it.
- **GET methods should not change the state of your app;**
  - the other methods do that.
- **If you GET an image 500 times, the image should never change**.
  - The response can change (the server may decide to send a different picture)
  - but GETs should not cause that change.

POST, PUT and DELETE

POST (For CRUD API this is equivalent to: CREATE)

- Generally used to request a **change to the state of the server**.
    - You POST a blog entry;
    - you **POST a photo to your favourite social network**;
    - you POST when you **sign up for a new account on a website**.
- POST is used to **create records on servers**, **NOT to modify existing records**.

PUT (For CRUD API this is equivalent to: UPDATE)

- A better name might be update or change.
    - If I have published (POSTed) a **job profile online** and later want to **update it**, I would PUT those changes.
    - I could PUT **changes to a document, or to a blog entry**, or to something else.
- **You do not use PUT to delete entries**, though; **that is what DELETE is for**.
- If you try to PUT changes to a record that does not exist, the server can (but does not have to) create that record.

DELETE (For CRUD API this is equivalent to: DELETE)

- Like PUT, you basically specify **DELETE record 123.**
- You could DELETE a blog entry, or **DELETE a photo**, or DELETE a comment.

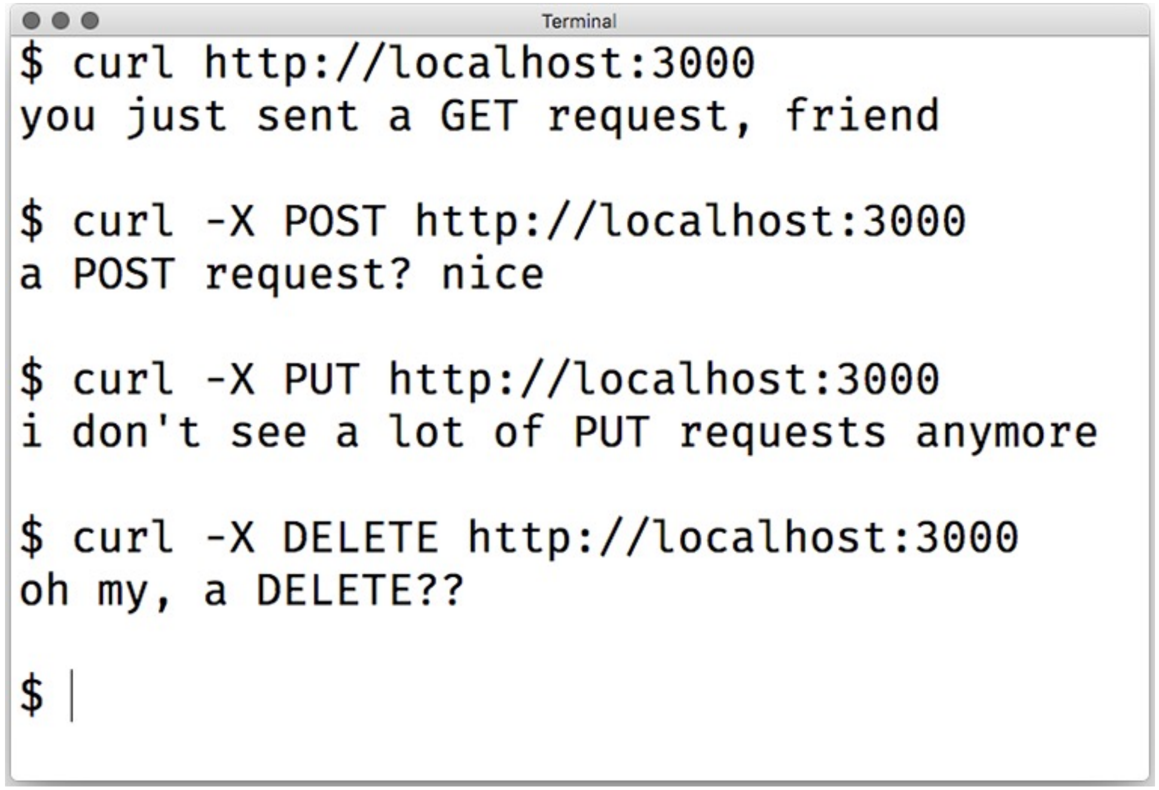
Handle HTTP Methods with Express Routing

An initial CRUD REST API with Express.js

```
var express = require("express");

var app = express();

app.get("/", function(req, res) {
   res.send(”A GET request, I read and send back the result for you");
});

app.post("/", function(req, res) {
   res.send("a POST request? Let’s create a new element");
});

app.put("/", function(req, res) {
   res.send(”Ok, let’s change an element");
});

app.delete("/", function(req, res) {
   res.send(”Are you sure??? Ok, let’s delete a record");
});

app.listen(3000, function() {
   console.log("CRUD app listening on port 3000");
});
```

cURL (client URL) Command-Line Tool

- In a browser (using the address bar), the HTTP request you send is always a GET (you cannot do POST, PUT, or DELETE from the address bar).
- you can use the handy `cURL` command-line tool to try sending different requests.
- `cURL` sends GET requests by default, but the `-X` argument can change the method.
  - For example, `curl -X PUT http://localhost:3000` will send a PUT request.



```
$ curl http://localhost:3000
you just sent a GET request, friend

$ curl -X POST http://localhost:3000
a POST request? nice

$ curl -X PUT http://localhost:3000
i don't see a lot of PUT requests anymore

$ curl -X DELETE http://localhost:3000
oh my, a DELETE??

$ |
```

Example: a REST Service for Getting Products

A REST Service for returning our **Petstore App products**:

```
...
let app = express();
app.set('json spaces', 3);

app.get("/collections/products", function (req, res) {
   //res.send("calling this service worked");
   //res.json({result: "OK"});

   let products = [
      {
         "id": 1001,
         "title": "Cat Food, 25lb bag",
         ...
      },
      {
         "id": 1002,
         "title": "Yarn",
         ...
      }
   ];

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

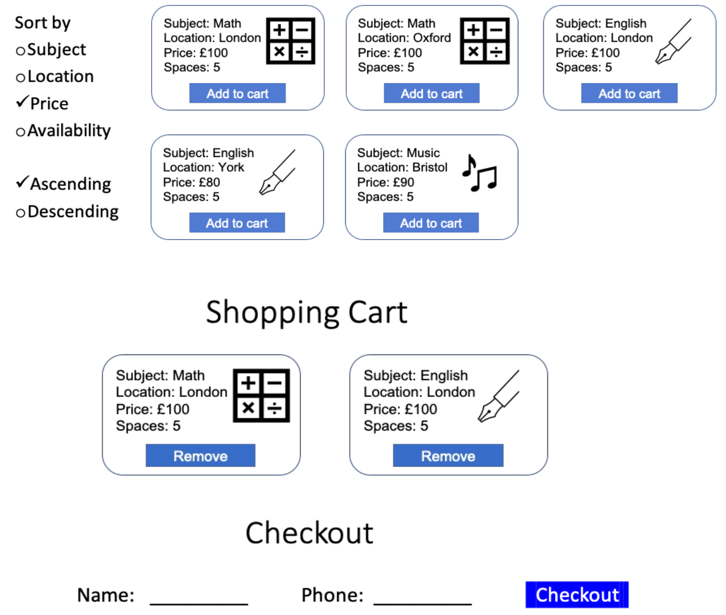
- `json spaces` setting beautifies JSON elements returned by services (by adding spaces among the different JSON elements and sub-elements)
- As in the commented `//res.send("calling this service worked");` potentially you can send back with your service also **text, xml or any other formats** if you want; however, nowadays, **usually JSON is used**
- As in the commented `//res.json({result: "OK"}); worked");` you can indicate there directly a JSON element, or preparing it as a variable as shown after (or better, as we will see, by retrieving JSON from a database)

CourseWork 2 (CW2) Requirements

CW2 Requirements

- **Back-End** of the After School Class App

- **Let's open the module page together:**
    - **Handbook**



Sort by
- ○ Subject
- ○ Location
- ✓ Price
- ○ Availability

- ✓ Ascending
- ○ Descending

Subject: Math
Location: London
Price: £100
Spaces: 5
Add to cart

Subject: Math
Location: Oxford
Price: £100
Spaces: 5
Add to cart

Subject: English
Location: London
Price: £100
Spaces: 5
Add to cart

Subject: English
Location: York
Price: £80
Spaces: 5
Add to cart

Subject: Music
Location: Bristol
Price: £90
Spaces: 5
Add to cart

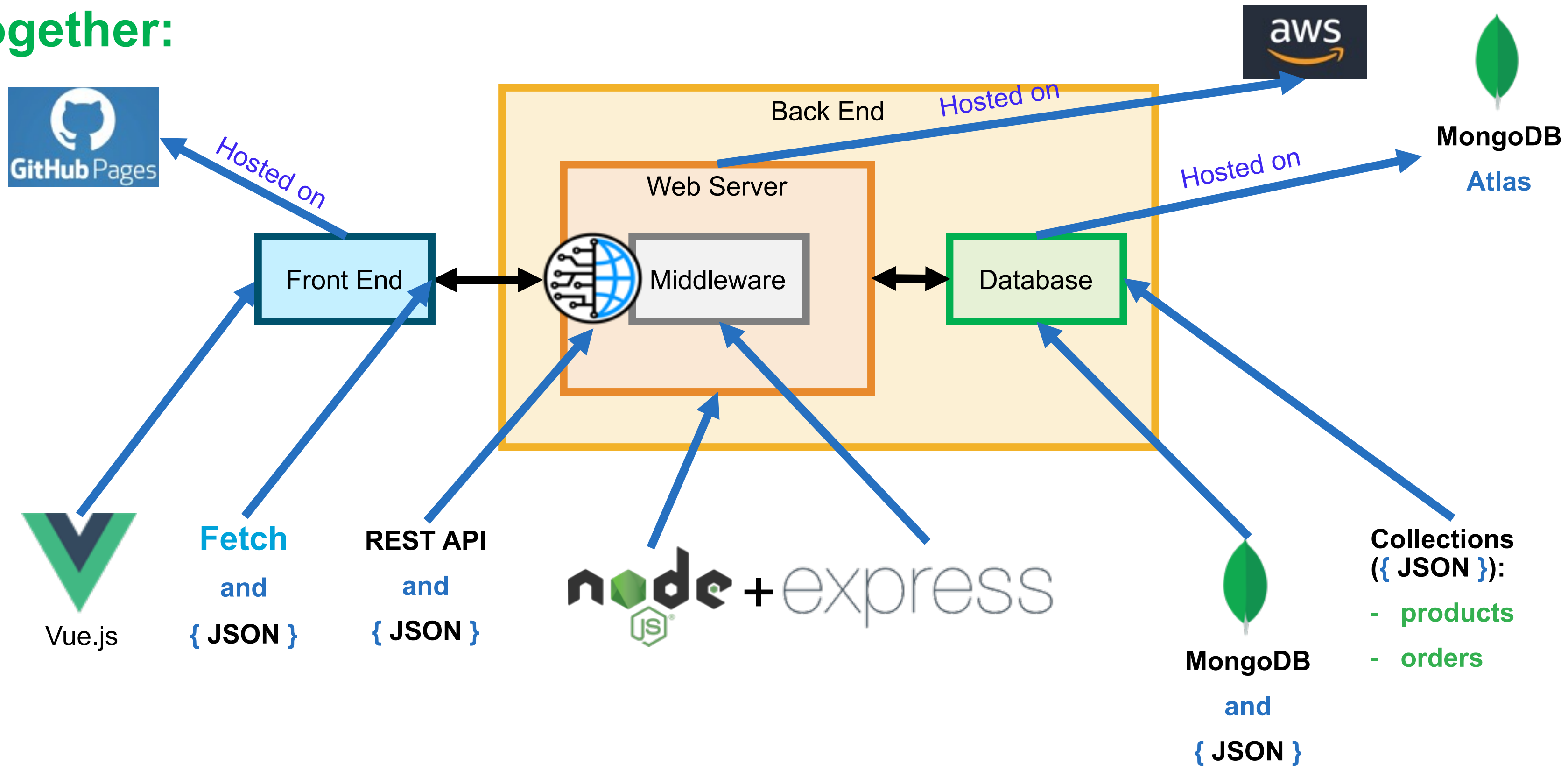
Shopping Cart

Subject: Math
Location: London
Price: £100
Spaces: 5
Remove

Subject: English
Location: London
Price: £100
Spaces: 5
Remove

Checkout

Name: __________ Phone: __________ Checkout



Suggestions for Reading

Reading

- MDN – Fetching Data from the Server

- Practical Node.js - Chapter 8

Questions?