JS

Outline

- Introduction to:
  - ReST
  - JSON
  - AJAX

Introduction to REST, JSON & AJAX

REST

REST, or **RE**presentational **S**tate **T**ransfer, is an architectural style for providing communication standards between computer systems on the web.

A RESTful API is an application program interface (API) that uses HTTP requests to GET, POST, PUT, and DELETE data.

REST-compliant systems are characterized by:

- Statelessness
- Separation of concerns

Systems that follow the REST paradigm are stateless, meaning that the server does not need to know anything about what state the client is in and vice versa.

This constraint of statelessness is enforced through the use of resources, rather than commands.

Resources are the nouns of the Web - they describe any object, document, or thing that you may need to store or send to other services.

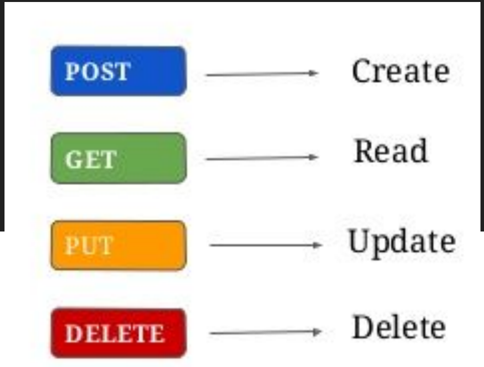
These constraints help RESTful applications achieve reliability, quick performance, and scalability, as components that can be managed, updated, and reused without affecting the system as a whole, even during operation of the system.

Separation of concerns

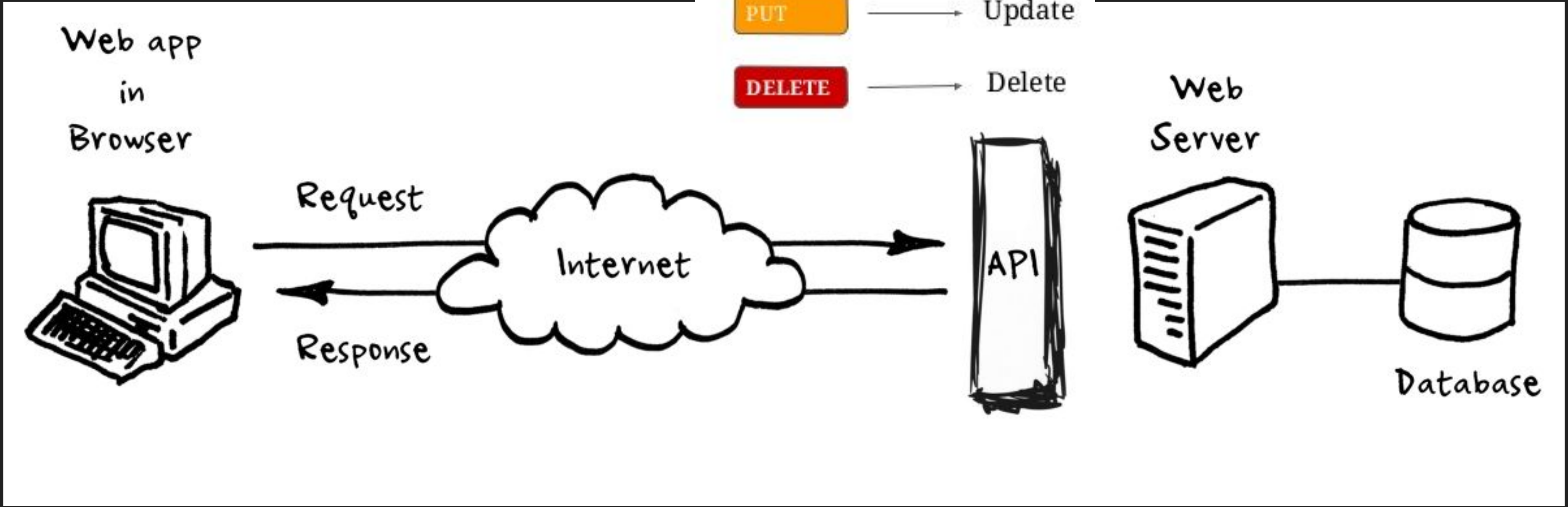
In the REST architectural style, the implementation of the client and the implementation of the server can be done independently without each knowing about the other.

Separating the user interface concerns from the data storage concerns, we improve the flexibility of the interface across platforms and improve scalability by simplifying the server components.

Additionally, the separation allows each component the ability to evolve independently.



POST	→ Create
GET	→ Read
PUT	→ Update
DELETE	→ Delete



Making Requests

REST requires that a client make a request to the server in order to retrieve or modify data on the server. A request generally consists of:

- an HTTP verb, which defines what kind of operation to perform
- a header, which allows the client to pass along information about the request
- a path to a resource
- an optional message body containing data

HTTP Verbs

There are 4 basic HTTP verbs we use in requests to interact with resources in a REST system:

- GET — retrieve a specific resource (by id) or a collection of resources
- POST — create a new resource
- PUT — update a specific resource (by id)
- DELETE — remove a specific resource by id

Response codes

Responses from the server contain status codes to alert the client to information about the success of the operation. For example:

- 200 (OK) This is the standard response for successful HTTP requests.
- 201 (CREATED) This is the standard response for an HTTP request that resulted in an item being successfully created.
- 204 (NO CONTENT) This is the standard response for successful HTTP requests, where nothing is being returned in the response body.
- 400 (BAD REQUEST) The request cannot be processed because of bad request syntax, excessive size, or another client error.
- 403 (FORBIDDEN) The client does not have permission to access this resource.
- 404 (NOT FOUND) The resource could not be found at this time. It is possible it was deleted, or does not exist yet.
- 500 (INTERNAL SERVER ERROR) The generic answer for an unexpected failure if there is no more specific information available.

For each HTTP verb, there are expected status codes a server should return upon success:

- GET — return 200 (OK)
- POST — return 201 (CREATED)
- PUT — return 200 (OK)
- DELETE — return 204 (NO CONTENT) If the operation fails, return the most specific status code possible corresponding to the problem that was encountered.

Read more: https://www.codecademy.com/articles/what-is-rest

JSON

JSON, or **J**ava**S**cript **O**bject **N**otation, is a minimal, readable format for structuring data. It is used primarily to transmit data between a server and web application, as an alternative to XML.

JSON Syntax Rules:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

The JSON format is syntactically identical to the code for creating JavaScript objects.

Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

Note: The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.

- JSON objects are written inside curly braces.

```
{"firstName":"John", "lastName":"Doe"}
```

- JSON arrays are written inside square brackets.

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

Converting a JSON Text to a JS Object

```
var text = '{ "employees" : [' +

'{ "firstName":"John" , "lastName":"Doe" },' +

'{ "firstName":"Anna" , "lastName":"Smith" },' +

'{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

Use the JavaScript built-in function JSON.parse() to convert the string into a JavaScript object:

```
var obj = JSON.parse(text);
```

Continuation

This is a continuation from last class, i assume you are upto date on understanding

- Json

If not, you might consider starting from slide 1.

AJAX

AJAX stands for Asynchronous JavaScript And XML. It is not a programming language and its a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX allows you to:

- Read data from a web server - after the page has loaded
- Update a web page without reloading the page
- Send data to a web server - in the background

XMLHttpRequest

XMLHttpRequest is a built-in browser object that allows to make HTTP requests in JavaScript.

XMLHttpRequest has two modes of operation: synchronous and asynchronous.

For a mock API service, we will be using JSON Placeholder
https://jsonplaceholder.typicode.com/

To do an async request, we need 3 steps:

1. Create XMLHttpRequest

```
let xhr = new XMLHttpRequest(); // no arguments
```

2. Initialize it.

```
xhr.open(method, URL, [async, user, password])
```

It specifies the main parameters of the request:

- method – HTTP-method. Usually "GET" or "POST".
- URL – the URL to request.
- async – if explicitly set to false, then the request is synchronous, we'll cover that a bit later.
- user, password – login and password for basic HTTP auth (if required).

3. Send it out.

```
xhr.send([body])
```

This method opens the connection and sends the request to server. The optional body parameter contains the request body.

4. Listen to events for response.

These three are the most widely used:

- load – when the result is ready, that includes HTTP errors like 404.
- error – when the request couldn’t be made, e.g. network down or invalid URL.
- progress – triggers periodically during the download, reports how much downloaded.

1. Create a new XMLHttpRequest object

```
let xhr = new XMLHttpRequest();
```

2. Configure it: GET-request

```
xhr.open('GET', 'https://jsonplaceholder.typicode.com/posts');
```

3. Send the request over the network

```
xhr.send();
```

4. This will be called after the response is received

```
xhr.onload = function() {
  if (xhr.status != 200) { // analyze HTTP status of the response
    alert(`Error ${xhr.status}: ${xhr.statusText}`); // e.g. 404: Not Found
  } else { // show the result
    alert(`Done, got ${xhr.response.length} bytes`); // responseText is the server
  }
};
```

```
xhr.onprogress = function(event) {
  if (event.lengthComputable) {
    alert(`Received ${event.loaded} of ${event.total} bytes`);
  } else {
    alert(`Received ${event.loaded} bytes`); // no Content-Length
  }

};

xhr.onerror = function() {
  alert("Request failed");
};
```

Fetch API

Method fetch() is the modern way of sending requests over HTTP.

It evolved for several years and continues to improve, right now the support is pretty solid among browsers.

The basic syntax is:

```
let promise = fetch(url, [options])
```

- url – the URL to access.
- options – optional parameters: method, headers etc.

The browser starts the request right away and returns a promise.

This kind of functionality was previously achieved using XMLHttpRequest.

Fetch provides a better alternative that can be easily used by other technologies such as Service Workers.

Fetch also provides a single logical place to define other HTTP-related concepts such as CORS and extensions to HTTP.

Read more:

- https://javascript.info/fetch-basics
- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

```
fetch('https://jsonplaceholder.typicode.com/todos)
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        console.log('success', data);
    })
    .catch(function (error) {
        console.log('error', error);
    });
```

Here is a full example

https://code.dcoder.tech/files/code/5ef0c6dd93af3305303ab098/fetch-api

Introduction to NPM and NodeJS

NPM

npm (Node Package Manager) was first created as a package manager for Node.js.

npm is the world's largest Software Registry with over 800,000 code packages.

npm is installed with Node.js. This means that you have to install Node.js to get npm installed on your computer.

Open-source developers use npm to share software.

npm includes a CLI (Command Line Client) that can be used to download and install software.

```
npm install <package>
```

npm package.json

All npm packages are defined in files called package.json.

The content of package.json must be written in JSON.

At least two fields must be present in the definition file: name and version.

```
{
"name" : "foo",
"version" : "1.2.3",
"description" : "A package for fooing things",
"main" : "foo.js",
"keywords" : ["foo", "fool", "foolish"],
"author" : "John Doe",
"licence" : "ISC"
}
```

If you want to share your own software in the npm registry, you can sign in at: https://www.npmjs.com

Publishing a Package

You can publish any directory from your computer as long as the directory has a package.json file.

1. Check if npm is installed

```
C:\>npm
```

1. Check if you are logged in

```
C:\>npm whoami
```

If not, log in

```
C:\>npm login

Username: <your username>
Password: <your password>
```

3. Navigate to your project and publish your project

```
C:\Users\myuser>cd myproject
C:\Users\myuser\myproject>npm publish
```

Node.js

Node.js is an open-source server side runtime environment built on Chrome's V8 JavaScript engine.

It provides an event driven, non-blocking (asynchronous) I/O and cross-platform runtime environment for building highly scalable server-side application using JavaScript.

Node.js can be used to build different types of applications such as command line application, web application, real-time chat application, REST API server etc.

Node.js official website: https://nodejs.org

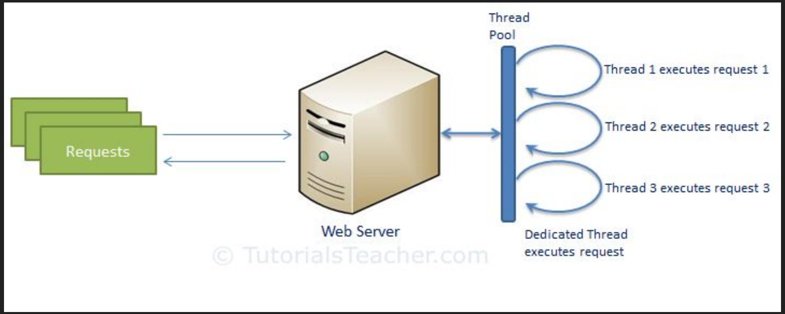
Advantages of Node.js

- Node.js is an open-source framework under MIT license. (MIT license is a free software license originating at the Massachusetts Institute of Technology (MIT).)
- Uses JavaScript to build entire server side application.
- Lightweight framework that includes bare minimum modules. Other modules can be included as per the need of an application.
- Asynchronous by default. So it performs faster than other frameworks.
- Cross-platform framework that runs on Windows, MAC or Linux

Node.js Process Model

Traditional Web Server Model

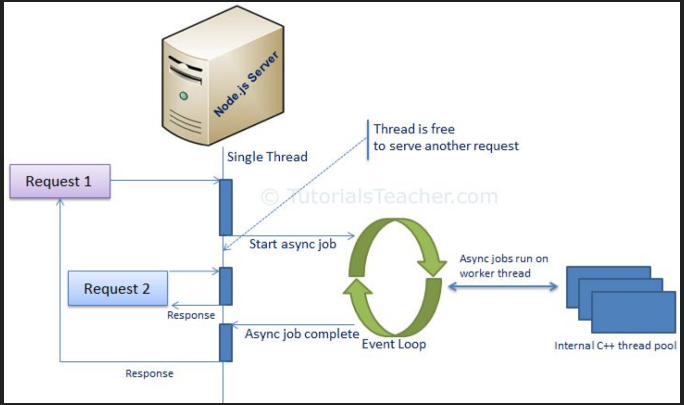
In the traditional web server model, each request is handled by a dedicated thread from the thread pool. If no thread is available in the thread pool at any point of time then the request waits till the next available thread.



Thread Pool
Thread 1 executes request 1
Thread 2 executes request 2
Thread 3 executes request 3
Requests
Web Server
Dedicated Thread executes request
© TutorialsTeacher.com

Node.js Server Model

Node.js runs in a single process and the application code runs in a single thread and thereby needs less resources than other platforms.



Source: https://www.tutorialsteacher.com/nodejs/what-is-nodejs

Verify Installation

Verify node version

```
node -v
```

Verify npm version

```
npm -v
```

For Windows users:

Make sure that **/usr/local/bin** is in your **$PATH**

Node.js Console - REPL

Node.js comes with virtual environment called REPL (aka Node shell). REPL stands for Read-Eval-Print-Loop.

It is a quick and easy way to test simple Node.js/JavaScript code.

Node.js Module

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder.

Node.js includes three types of modules:

- Core Modules
- Local Modules
- Third Party Modules

Node.js Core Modules

The core modules include bare minimum functionalities of Node.js.

Most useful core modules include:

- **http** module - includes classes, methods and events to create Node.js http server.
- **url** module - includes methods for URL resolution and parsing.
- **querystring** module - includes methods to deal with query string.
- **path** module - includes methods to deal with file paths.
- **fs** module - includes classes, methods, and events to work with file I/O.
- **util** module - includes utility functions useful for programmers.

Loading Core Modules

In order to use Node.js core or NPM modules, you first need to import it using require() function as shown below.

```
var module = require('module_name');
```

For example, to load and use core http Module

```
var http = require('http');
```

In the above example, require() function returns an object because http module returns its functionality as an object, you can then use its properties and methods using dot notation e.g. http.createServer().

Node.js Local Module

Local modules are modules created locally in your Node.js application. These modules include different functionalities of your application in separate files and folders.

```
// lib.js
var lib = {
   square: function (x) {
       return x * x;
   },
   sum: function (a, b) {
       return a + b;
   }
};

module.exports = lib;
```

```
// main.js
var lib = require('./lib.js')

console.log(lib.square(5));  // 25
console.log(lib.sum(4, 3));   // 7
```

Node.js Web Server

Node.js provides capabilities to create your own web server which will handle HTTP requests asynchronously.

```
var http = require('http'); // 1 - Import
Node.js core module

var server = http.createServer(function (req,
res) {   // 2 - creating server

   //handle incoming requests here..

});

server.listen(5000); //3 - listen for any
incoming requests

console.log('Node.js web server at port 5000 is
running..')
```

GreensKiosk API web server

```
// server.js
var http = require('http'); // Import Node.js core module
var GreensKiosk = require('./GreensKiosk');

var server = http.createServer(function (req, res) {   // create web server

  // Set CORS headers
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader('Access-Control-Request-Method', '*');
  res.setHeader('Access-Control-Allow-Methods', 'OPTIONS, GET');
  res.setHeader('Access-Control-Allow-Headers', '*');
  if (req.method === 'OPTIONS') {
    res.writeHead(200);
    res.end();
    return;
  }

  if (req.url == '/') { // check the URL of the current request
    // set response header
    res.writeHead(200, { 'Content-Type': 'text/html' });
    // set response content
    res.write('<html><body><p>Greens Kiosk API.</p></body></html>');
    res.end();
  }
  else if (req.url == "/products") { // check the URL of the current request
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.write(JSON.stringify(GreensKiosk.getItems()));
    res.end();
  }
  else if (req.url == "/products/fruits") {
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.write(JSON.stringify(GreensKiosk.getItems('fruits')));
    res.end();
  }
  else if (req.url == '/products/vegetables') {
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.write(JSON.stringify(GreensKiosk.getItems('vegetables')));
```

```
// GreensKiosk.js
let products = [
{
    name: 'Mangoes',
    category: 'fruits'
},
{
    name: 'Apples',
    category: 'fruits'
},
{
    name: 'Bananas',
    category: 'fruits'
},
{
    name: 'Oranges',
    category: 'fruits'
},
{
    name: 'Grapes',
    category: 'fruits'
},
{
    name: 'Kales',
    category: 'vegetables'
},
{
    name: 'Onions',
    category: 'vegetables'
},
{
    name: 'Tomatoes',
    category: 'vegetables'
},
{
    name: 'Cabbages',
    category: 'vegetables'
```

Assignment

```html
<!DOCTYPE HTML>
<html>

<head>
   <title>Greens Kiosk</title>
</head>

<body>
   <h1 id="title">Welcome to Greens Kiosk</h1>
   <p>We sell fruits and vegetables</p>
   <h2>Products</h2>
   <ul id="products">
       <!-- Show products list from API -->
   </ul>
   <h3>Fruits</h3>
   <ul id="fruList">
       <!-- Show fruits from API -->
   </ul>
   <h3>Vegetables</h3>
   <ul id="vegList">
       <!-- Show vegetables from API -->
   </ul>
</body>

</html>
```

Questions:

1. Show products from /products
2. Show fruits from /products/fruits
3. Show vegetables from /products/vegetables

Next class

- What is React
- Environment Setup
- Your First React App
- Introduction to JSX
- Rendering Elements

Video of the week - https://www.youtube.com/watch?v=wbB3lVyUvAM
https://www.youtube.com/watch?v=7YcW25PHnAA