Introduction to NodeJS

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

NodeJS API

You can find more information on NodeJS API here. These packages allow us to run NodeJS in the backend same way we run C++ and Java. There are many APIs, but we will expore the only a few in this class:

1. Crypto
2. File System (fs)
3. DNS
4. URL
5. HTTP

Crypto

The `node:crypto` module provides cryptographic functionality that includes a set of wrappers for OpenSSL's hash, HMAC, cipher, decipher, sign, and verify functions. Below are some examples of using the Crypto module.

Asynchronously generate a random string

```
const {
    randomBytes,
} = require('node:crypto');

randomBytes(11, (err, buf) => {
    if (err) throw err;
    console.log(`${buf.length} bytes of random data:
                    ${buf.toString('hex')}`);
});
```

```
//Example Answer
11 bytes of random data: cebb9143091b667bbeb68a
```

Asynchronously generate a UUID (Universally Unique Identifier)

```
require('node:crypto');
console.log(crypto.randomUUID());

//Example Answer
ee1dc045-dc91-41d8-8282-e570339c3ca2
```

Storing Passwords with Scrypt

```
// Input values
const password = 'Password123';
const salt = 'uniqueSaltValue'; //Unique for every user
const keyLength = 11; // Length of the derived password

crypto.scrypt(password, salt, keyLength, (err, derivedKey) => {
    if (err) throw err;
    console.log('Derived key (hex):', derivedKey.toString('hex'));
});

//Answer for Password123
Derived key (hex): 9b754db6a3aec4edfbe7a3
```

File System

This module enables interacting with the file system in a way modeled on standard POSIX functions.

Writing a file

```
const data = "Hello CS355!!";
fs.writeFile("input.txt", data, "utf8", (err) => {
    console.log(err);
});

//Check your file explorer for input.txt
```

Reading a file

```
fs.readFile("./input.txt", "utf8", (err, data) => {
    if(err) throw err;
    else console.log(data);
})

//Answer Example
Hello from Cs355!!
```

Creating Folders

```
fs.mkdir("./directory", (err) => {
    if(err) console.log("Failed to create folder.");
    else {
        for(let i = 10; i <= 15; i++){
            fs.writeFileSync(`./directory/${i}.txt`, `File ${i}`,
            (err) => {
                if(err) console.log(`Error writing file ${i}`);
            })
        }
        displayDir();
    }
})

function displayDir(){
    fs.readdir("./directory", (err, files) => {
        if(err) throw err;
        for(let file of files) {
            console.log(file);
        }
    })
}
```

Domain Name System (DNS)

The dns module enables name resolution. For example, use it to look up IP addresses of host names. www.google.com is given an IP address, dns find this IP address.

DNS Resolve

```
const domain = "google.com"
```

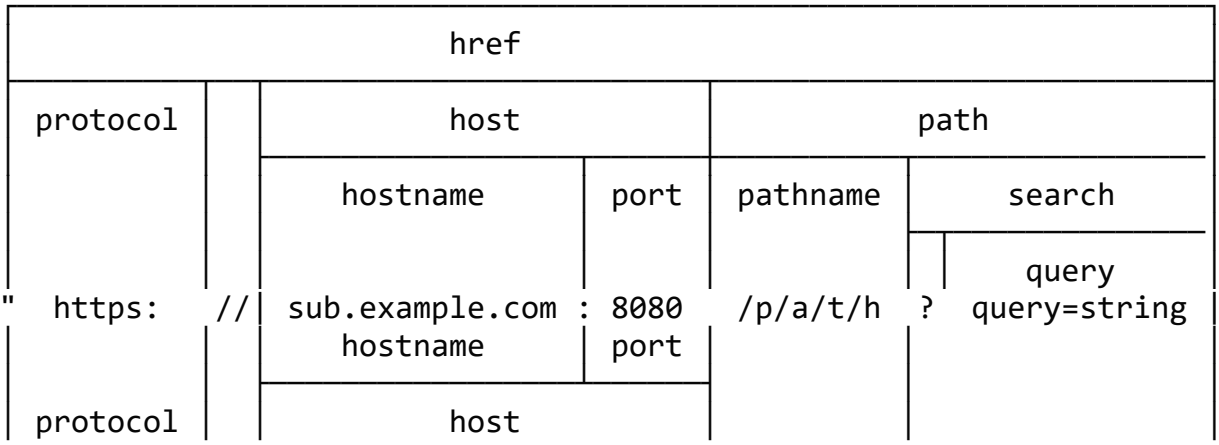
```
dns.resolve(domain, (err, address) => {
    if(err) throw err;
    else console.log(`IP of ${domain}: ${address}`);
})

//Answer
IP of google.com: 142.250.65.238
```

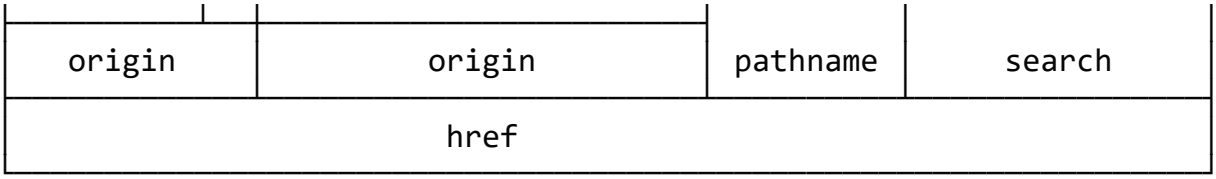
DNS Resolve Multiple Hosts

```
const domains = ["google.com", "amazon.com", "ebay.com", "cuny.edu"];

for(let domain of domains) {
    dns.resolve(domain, (err, address) => {
        if(err) console.log(`Error on ${domain}`);
        else console.log(`IP of ${domain}: ${address}`);
    });
}

//Answer Example
//Note the order
//Happens because resolve-function is asynchronous
IP of google.com: 142.251.41.14
IP of cuny.edu: 128.228.254.200
IP of ebay.com: 23.48.224.102,23.48.224.105
IP of amazon.com: 205.251.242.103,54.239.28.85,52.94.236.248
```

Uniform Resource Locator (URL)

A URL string is a structured string containing multiple meaningful components. When parsed, a URL object is returned containing properties for each of these components.





Creating URL Object

```
let urlObj = new URL("https://www.google.com/search?q=how+to+learn+js"
console.log(urlObj);

//Answer
URL {
    href: 'https://www.google.com/search?q=how+to+learn+js',
    origin: 'https://www.google.com',
    protocol: 'https:',
    username: '',
    password: '',
    host: 'www.google.com',
    hostname: 'www.google.com',
    port: '',
    pathname: '/search',
    search: '?q=how+to+learn+js',
    searchParams: URLSearchParams { 'q' => 'how to learn js' },
    hash: ''
}
```

Note that the new URL object created is dictionary-style, json-style or js-object style (however you like to think about it). Furthermore, note that one of the key-value pairs is "serachParams", where the value is another object. Search the NodeJS URL API to learn how to get each parameter and more.

HyperText Trasfer Protocol (HTTP)

The HyperText Transfer Protocol (HTTP) is a foundational communication protocol for the World Wide Web. It defines how clients (e.g., web browsers) and servers communicate by exchanging requests and responses. HTTP is a stateless and application-layer protocol that runs on top of TCP/IP.

- **Stateless:** Each request/response pair is independent, and the server does not retain information about past interactions.
- **Text-Based**: Communication is in plain text, making it human-readable.
- **Request/Response Model**:
  - Client: Sends an HTTP request (e.g., browser, API client).
  - Server: Responds to the request with an HTTP response.

HTTP defines several methods for different types of operations. These methods indicate the desired action to be performed for a resource.

Common HTTP Request Methods

Method	Description
GET	Retrieves data from the server. The **request** has no body.
POST	Sends data to the server to create or update a resource. Includes a request body.
HEAD	Similar to GET, but the **response** only retrieves the headers, not the body of content.
PUT	Replaces an existing resource with the provided data or creates one if it doesn't exist.
DELETE	Deletes the specified resource from the server.

Common HTTP Response Codes

Code	Description
200	200 OK: Standard success for most requests.
301	301 Moved Permanently: Redirect clients to a new URL.
403	403 Forbidden: Rejected access
404	404 Not Found: Resource doesn't exist at the requested URL.
500	500 Internal Server Error: Something is wrong on the server side.

Common Request Headers

Header	Description
Host	Specifies the domain name of the server being requested.
User-Agent	Identifies the client making the request (e.g., browser type/version).
Accept	Specifies the MIME types the client can handle (e.g., text/html).
Content-Type	Specifies the type of data being sent (e.g., application/json).
Authorization	Carries credentials for authentication.
Cookie	Contains stored cookies for the server.
Accept-Language	Specifies the preferred languages for the response. It helps the server serve content in the appropriate language.
If-Modified-Since	Tells the server to send the resource only if it has been modified since the given date.
Connection	Specifies whether the connection should be kept alive after the current transaction or closed. Common values are keep-alive and close

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
```

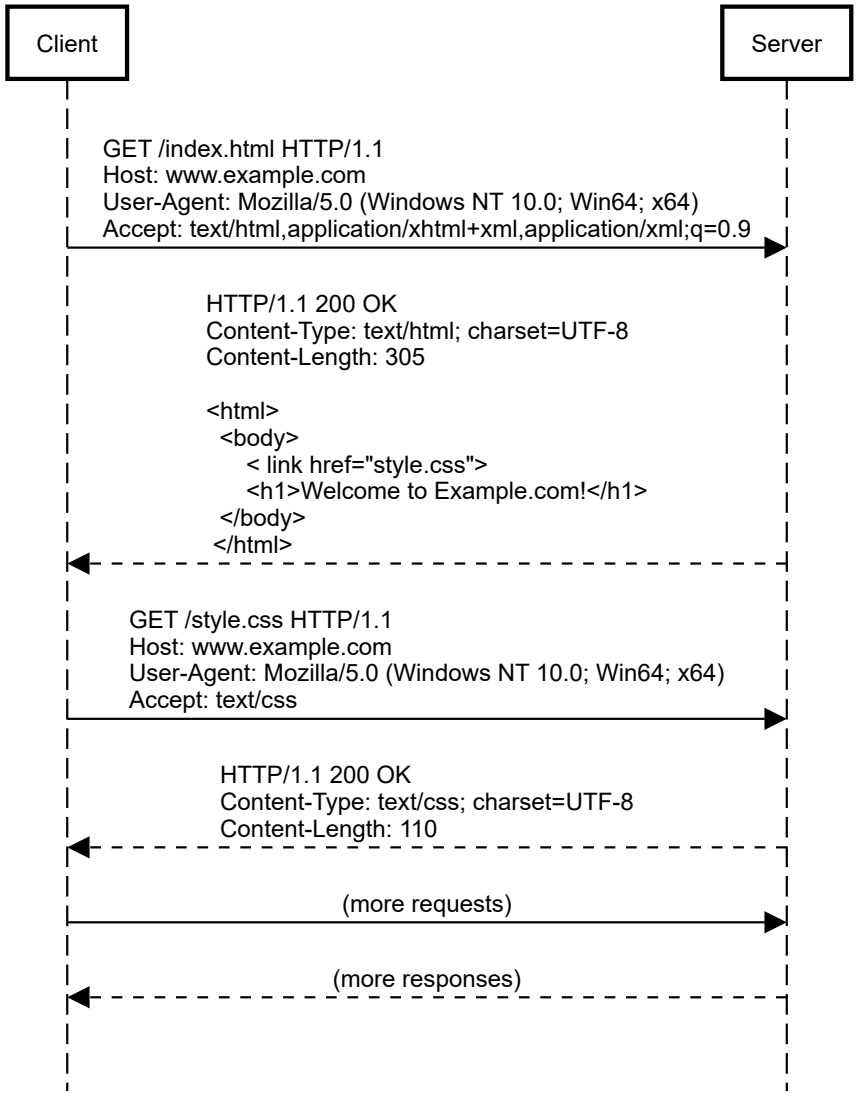
Common Response Headers

Header	Description
Content-Type	Specifies the format of the response body (e.g., application/json).
Content-Length	Indicates the size of the response body in bytes.
Cache-Control	Directs caching mechanisms on how to store the response.
Set-Cookie	Instructs the client to store cookies.
WWW-Authenticate	Defines how the client should authenticate itself to access the resource.
Location	Used in redirects to indicate the new URL.

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 305

//Body of response
```

HTML Request-Response Diagram

HTTP Req-Res


Client
Server
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 305
<html>
<body>
< link href="style.css">
<h1>Welcome to Example.com!</h1>
</body>
</html>
GET /style.css HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/css
HTTP/1.1 200 OK
Content-Type: text/css; charset=UTF-8
Content-Length: 110
(more requests)
(more responses)