**NODE JS**

* Node js is powerful js runtime environment which is built on google chromes v8 js engine.
* it is cross platform and open source
* Single threaded, based on event driven, non-blocking i/o model.
* An **event-driven** architecture is a programming model where the flow of the program is determined by **events**—such as user interactions, sensor outputs, or messages from other programs. In this model, components of the system wait for events to occur, and once an event occurs, they respond to it.
* with node js we can build apis, do server side programming
* open w3 schools and have some glance
* https://www.w3schools.com/nodejs/nodejs\_intro.asp

**JavaScript Runtime Environment**

* Node.js is built on the **V8 JavaScript engine** (the same engine used by Google Chrome), which compiles and executes JavaScript code. This makes it possible to run JavaScript outside of a web browser.
* In traditional web development, JavaScript is used on the client side (in browsers). Node.js allows you to use JavaScript on the server side, making it a full-stack language.

**Use node js:**

* Api with db behind it
* Data streaming websites like utube
* Real time chat application

**Don’t use:**

* Applications like heavy server-side programming(cpu-intensive)

**Synchronous File Writing**

* In synchronous file writing, the program waits for the file operation to complete before moving on to the next line of code. This means the code execution is blocked until the writing operation finishes.
* Blocking Behavior: Since the code execution halts until the operation is complete, it can be problematic in performance-sensitive applications, especially when dealing with large files or multiple I/O operations.

**Asynchronous File Writing:**

* **Definition**: In asynchronous file writing, the program doesn't wait for the file operation to complete before moving to the next line of code. The file write operation is non-blocking, and a callback (or promise) is used to handle the result once the operation finishes.
* **Non-blocking Behavior**: The asynchronous approach ensures that the program can continue performing other tasks while waiting for the I/O operation to finish.

**Creating a simple web server:**

const server=http.createServer((req, res)=>{

    //console.log(req);

    res.end('hello from server');

});

server.listen(8000,'127.0.0.1',()=>{

  console.log('listening to requests on port 8000');

});

**Routing**: implementing different actions for different urls.

**HTTP Header:** It is a piece of info about the response that we are sending back.

**HTTP Status code:** like 404 etc

**API:** API is a service from which we can request some data and we back that data to client

**SERVERS / ROUTING :**

const server=http.createServer((req, res)=>{

    const pathName=req.url;

if(pathName==='/' || pathName==='/overview')

{

    res.setHeader('Content-Type', 'text/html');

    res.end('<h1>This is OVERVIEW</h1>');

}

else if(pathName=='/product')

{

    res.end('this is PRODUCT');

}

else{

    res.writeHead(404,{

        'content-type': 'text/html'

    });

    res.end('<h1>PAGE NOT FOUND</h1>')

}

});

server.listen(8000,'127.0.0.1',()=>{

  console.log('listening to requests on port 8000');

});

. in path is usually where the script is running

\_\_dirname is where the current file is located

NPM is a software that we basically use to manage the third party open resource packages that we choose to include in our project.

* + Simple dependencies or regular dependencies are simple packages that contain some code that we will include in our code
  + Express is a dependency where is slugify is package which is small tool which we use to make more readable URLs out of names

Besides these regular dependencies we also have development dependencies which are tools for development. We simply use them to develop our applications.

**Regular Dependencies (dependencies)**

* **What they are**: These are the packages that your application needs to run in production. They are essential for the core functionality of your application and are typically required when your app is deployed or running in a live environment.
* **Example**: Libraries like express, mongoose, or axios, which your app needs to function properly in production.

**Development Dependencies (devDependencies)**

What they are: These are the packages that are only needed during development. These dependencies are not required when your application is running in production. They are typically tools used to help with tasks like testing, transpiling code (e.g., Babel), bundling (e.g., Webpack), linting (e.g., ESLint), and more.

Example: Tools like mocha, jest, eslint, webpack, and babel are common development dependencies because they help with tasks during development, but they are not needed in a production environment.

npm install nodemon --save-dev is nodemon cmd only works locally

npm i nodemon –global is cmd to use globally

which is similar to devtools in springboot. It will automatically restart the server eachtime we make changes.

**SLUG:**

const slugify=require('slugify'); //slug is basically just the last part of a url that contains a unique string that identifies the resouce that website is displaying

//for ex http://localhost:8000/product?id=0 if open product avacado i should not get ?id=0 rather ishould get smthg meaningful like /product/avocado

**Package versioning:**

If I have version no ^1.18.9, in this 1 is is called major version , 18 is called minor version and 9 is called **patch version**. If any bugs is there in 1.18 then they will fix and release 1.18.1 and soo on

**Minor versions** comes with add on features in them , if they have any releases then 1.19 wil be released. It will not break the code form previous versions.

**Major version** are done when there are huge releases .For example: slugify has changed the no of parameters etc then new version will come and break the previous code.

^ says that we are ready for minor and patch releases

Npm outdated ->gives latest versions

If we want to install particular version then npm install [slugify@1.0.0](mailto:slugify@1.0.0)

~1.18.11 this symbol accept only patch releases.

**Installing and deleting**

npm install express

npm uninstall express

**Setting Prettier**

It helps in automatically formatting our code

**How node.js works behind the scenes:**

V8 engine is fundamental part of nodejs

V8 engine converts js code into machine code which a computer can understand

Libuv is another dependency which is needed for node js

Libuv is open source library with a string focus on asynchronous IO.this layer gives node access to the underlying os, file system, networking and more.

2 important features is also done by libuv i.e eventloop, threadpool

**Node process and threads**

Node js process is instance of a program in execution on a computer

Thread is sequence of instructions.

Node js runs on singe thread. Even if there are single or million users. So that thread must be accessible to everyone and that should not be blocked.

Easy tasks are done by event loop and heavy are done by thread pool

A screen shot of a computer

Description automatically generated

**Event loop:**

Node.js is build around callback functions

It follows event driven architecture

* + Events are emitted to event loop like file reading completed, new http requests etc
  + Then event loop picks them up
  + And callbacks are called.

Eventloop does orchestration i.e it receives events, calls their callback functions and gives the more expensive task to threadpool.

**To prevent our thread from blocking:**

Don’t use sync versions of functions

Don’t perform large calculations

Careful with json in large objects

Don’t use complex regular expressions

**Event Loop and threadpool**

**Practising eventloop**

const fs = require("fs");

const crypto = require("crypto");

const start = Date.now();

process.env.UV\_THREADPOOL\_SIZE = 4;

setTimeout(() => console.log("Timer 1 done"), 0);

setImmediate(() => console.log("Immediate 1 finished"));

fs.readFile("text-file.txt", () => {

  console.log("I/O finished");

  console.log("--------------");

  setTimeout(() => console.log("timer 2 done"), 0);

  setTimeout(() => console.log("Timer 3 finished"), 5000);

  setImmediate(() => console.log("Immediate 2 is finished"));

  process.nextTick(() => console.log("Process.nextTick"));

});

crypto.pbkdf2Sync("password", "salt", 100000, 1024, "sha512");

{

  console.log(Date.now() - start, "password encrypted1");

}

crypto.pbkdf2Sync("password", "salt", 100000, 1024, "sha512");

{

  console.log(Date.now() - start, "password encrypted2");

}

crypto.pbkdf2Sync("password", "salt", 100000, 1024, "sha512");

{

  console.log(Date.now() - start, "password encrypted3");

}

crypto.pbkdf2("password", "salt", 100000, 1024, "sha512", () => {

  console.log(Date.now() - start, "password encrypted");

});

console.log("Helllo from the top level code");

**Events and event driven architecture**

**Event listeners:**

We can have multiple event listeners to the same event

**Express**

Express is minimal node js framework, it is actually build on top of node js which provides higher level of abstraction.

It is most popular node js framework. It contains a very set of features like complex routing, easier handling of requests and responses, middleware , server-side rendering.

Faster development of node js and express code is completely written in javascript

Easier to organize our application into mvc architecture (model view controller)

**POSTMAN:**

Allows us to do API Testing. Its little bit like a browser, we can do all kinds of requests and simply receive the response.

**Simple express code**

const express = require("express");

const app = express();

app.get("/", (req, res) => {

  res.status(404).json({ message: "hello from server side", app: "natures" });

});

// app.get("/p", (req, res) => {

//   res.status(200).send("hellooo");

// });

app.post("/", (req, res) => {

  res.send("you can  post to this endpoint");

});

const port = 3000;

app.listen(port, () => {

  console.log(`app running on port ${port}...`);

});

**API:**

It is a piece of software that can be used by another piece of software in order to allow applications to talk to each other.

**Rest API:**

It is basically a way of building web apis in a logical way, making them easy to consume

**REST API:**  is a set of conventions for building web services that use HTTP methods like GET, POST, PUT, DELETE.

A stateless REST API does not rely on any session data stored on the server, meaning each request is treated independently and must include all the information needed to process it. This approach promotes scalability, simplicity, and security. JWT (JSON Web Tokens) is commonly used for authentication in stateless APIs, as it allows the server to verify user identity without maintaining any session state.

**Drawbacks of Stateless REST APIs**

 **Redundant Information in Every Request**:

* Each request must contain all necessary information (e.g., authentication tokens, parameters, etc.), which can lead to larger requests and overhead for the client and the network.

 **Token Expiry**:

* Tokens (like JWT) typically have an expiration time. After the token expires, the client needs to re-authenticate, which can create an additional step for the user.

 **Less Control Over Clients**:

* Since the server does not keep track of client states, it has less control over things like session timeout, tracking user behavior, or maintaining custom session data.

**Http methods:**

1. **GET**: Retrieves data from the server (e.g., fetching a list of users).
2. **POST**: Sends data to the server to create a new resource (e.g., adding a new user).
3. **PUT**: Updates an existing resource (e.g., updating a user’s information).
4. **DELETE**: Removes a resource (e.g., deleting a user).
5. **PATCH**: Partially updates a resource (e.g., updating one field in a user's profile).