**DAY 1**

**How Node JS differs from Vanilla JS?**

* Node runs on server --- not browser
* The console in node is terminal window – not browser console
* Has global object instead of windows object
* By default Node has CommonJS modules instead of ES6 modules.
* Missing some APIs like fetch API

--dirname property

--filename property

**OS module:**

* Os.type()
* Os.version()
* Os.homedir()

**Path module:**

* Path.dirname(\_\_filename)
* Path.basename(\_\_filename)
* Path.extname(\_\_filename)
* Path.parse(\_\_filename)
* Path.join() method- recommended to use for file paths

fs.readFile(path.join(\_\_dirname, "files", "starter.txt"), (err, data) => {

  console.log(data.toString());

  if (err) throw err;

});

**fs module:**

* Fs.readFile(),
* Fs.writeFile(),
* Fs.appendFile(),

**DAY 2**

**Promises in Fs module:**

* Import promises and use the async functionality of all operations

const fsPromises = require("fs").promises;

* **Delete a file:**
* await fsPromises.unlink(path.join(\_\_dirname, "files", "starter.txt"));
* **Rename a file:**
* await fsPromises.rename(
* path.join(\_\_dirname, "files", "promiseWrite.txt"),
* path.join(\_\_dirname, "files", "promiseComplete.txt")
* );
* Use **fs.existsSync()** method to check weather a directory or a file exists or not
* Make new directory using **fs.mkdir()**  method.
* Delete a directory using **fs.rmdir()**  method.

**Streams- a better way to read and write files:**

fs.createReadStream()

fs.createWriteStream()

**Node Modules**

* NPM
* Installing modules / removing them
* Adding dev dependencies, scripts in package.json file

**DAY 4**

* Create express server
* Set get route
* **TIP:** Always make routes in a way that starting from / route and at the end /\* route for default. It is because express treats the routing in a waterfall manner. It means that if it does not find a match in the first route, it comes to the second and then so on.
* Send response
* **Res.sendFile(path,{root:\_\_dirname})**

**OR  
res.sendFile(path.join(\_\_dirname, "files", "new-page.html"));**

* **REGEX  
  app.get("^/$|index(.html)?")**=>In JavaScript regex, ^ asserts the start of a string, and $ asserts the end. Together, ^$ ensures that the entire string matches the specified pattern, requiring an exact match without any additional characters before or after. This is useful for strict pattern matching in the entire string.  
   =>To make something optional, just wrap it with brackets () and put a ? mark at the end. E.g:  
  **app.get(‘index(.html)?’) .** It will send same response for “index” and “index.html”
* **res.redirect()**

The status code by default send by express in this case is 302 that is for a temporary redirection of route. If we want to make a permanent redirection of any route then we must specify its status code that is 301

* **Route handlers:**(req,res,next)

**DAY 5 Middlewares**

**Three types:**

* Built-in
* Custom built middle wares
* Third party middle wares

**Commonly used Built-in middleware:**

* **app.use(express.urlencoded({extended:false}))**

is used to accept url encoded data such as from html forms.

* **app.use(express.json())**

is used to accept data in the form of json object.

* **app.use(express.static(path.join(\_\_dirname,’’./files”))**

is used to serve static files from the specified directory.

**CORS:**

* used for cross origin resource sharing
* We can limit the number of origins that can access our APIs using corsOptions object:

const whiteList = [

  "https://www.google.com",

  "http://127.0.0.1:5000",

  "http://localhost:5000",

  "http://localhost:5000/",

];

const corsOptions = {

  origin: (origin, callback) => {

    if (whiteList.indexOf(origin) !== -1 || !origin) {

      callback(null, true);

    } else {

      callback(new Error("Not allowed by CORS"));

    }

  },

  optionsSuccessStatus: 200,

};

app.use(cors(corsOptions));

* **Route handlers:**Just like middlewares.   
  We can specify the number of functions/middlewares and their order in which the request proceed before response

const one = (req, res, next) => {

  console.log("One");

  next();

};

const two = (req, res, next) => {

  console.log("two");

  next();

};

const three = (req, res) => {

  console.log("three");

  res.send("done with all three routers");

};

app.get("/hello(.html)?", [one, two, three]);

**DAY 6 Error Handler**

* Creating error handler custom middleware
* App.use() vs app.all()
* **app.all** will attach to the app’s implicit router. **app.all** attaches a particular piece of middleware to **all HTTP methods**, and if attached in the main config file will globally apply the middleware to all requests made to your app. Like **app.use**, it is also possible to specify a path for which the middleware should be applied.
* **app.all** also accepts a regex as its path parameter. **app.use** did not accept a regex in previous versions,, but now accepts regex.

**DAY 7 ROUTING**

Routing and MVC architecture of REST API