CS-457-WAF

WEB APPLICATION FRAMEWORK

26-09-2024

Lecture No. 2

Lecture No. 1 (Course Outline).

→ Synchronous:

- Executes line by line code consecutively in a sequential manner.
- Blocking Architecture → The code waits for an operation to complete.

→ Asynchronous :

- Allows multiple operations to be performed concurrently without waiting. (No Delay).
- Non-Blocking Architecture — execution flow is not blocked.
- Asynchronous code is handled with call Backs.

* Call Back Hell :→ Nested call Backs upto 5 Levels
  - Promises
  - Async /Await.

→ set timeout(( ) => { // Asynchronous function.
  }, 3000);

- Event Loop.

Note: Javascript is single Threaded.

→ CallBack Hell:

- CallBacks are nested within other callbacks to the extent where the code is difficult to read.
- Old pattern to deal with asynchronous code.
- Use promises / Async-Await to avoid callback hell.

→ PROMISE:

- An object that manages asynchronous operation
- Wrap a Promise object around asynchronous code.
- It returns a promise — pending < resolve
  reject.
- new Promise ((resolve, reject) => {asynchronous code}

* function chaining.

→ Async / Await:

Async — makes a function return a promise.
Await — makes an asynchronous function wait for a promise.

Task: signup()
sendEmail()
Login()
getData()
displayData()

→ some delay.
CallBacks
Promise
Async / Await
Error Handling.

Discover
Event Loop.

Tuesday 01/10/2024.

LECTURE NO. 3

→ Why Asynchronous Code?

- To avoid blocking code.
- To develop scalable, responsive applications/robust application.

Problem: It is difficult to preserve order of event

* ES6 → ECMASCRIPT?

↳ Specification - It make sures That across The browse behaviour of javascript remains same.

→ JS RunTime Environment:

1) Execution Engine (v8).
2) web API.
3) Callback Queue
4) Event Loop.

→ v8 Engine embedded in Node - Not as it is Front End removed (DOM ,etc).

- Memory Heap → object's
- Call Stack.
- CallBack Queue.

→ Javascript is single-Threaded. But due to event loop it can behave like multi-Threaded.

→ If call stack is empty, pop The call back queue and push ~~it~~ ~~to~~ The call back function to call Stack. Else wait

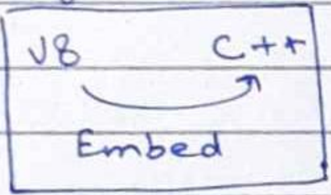
- npm i node-fetch

→ <u>Node JS</u>:

Javascripl — Every browser has a javascript engine.
. Chrome — V8 Engine
. Firefox — Spider Monkey.

Node js
- V8 Engine is embedded with C++, makes it possible to access machine level task using javascript.

Ryan Dahl → Node js



• Now we can also execule javascript outside browser using node js.

- Node js is a runtime environmenl, nol a framework or library.
- V8 is not embedded as a whole, most of it's features are removed: Example:- alert("WAF") // Error.
- we can work with filesyslem using node.
- node REPL.

R - READ
E - EVAL
P - PRINT
L - LOOP

* npm init / npm init -y

// Adding scripls

```
"amazon" : "node index.js"
npm run amazon.
```

→ Modules:

- Export module:

```
module.exports = add
```

- Import module:

const ~~requ~~math = require("./math");

Multiple functions.

```
module.exports = add        //only one will be exported
module.exports = sub        using this method. Here only
                            sub func. will be exported.
```

Solution: Export as an object.

- module.exports = { add, sub}

OR

- module.exports = {

```
addFunction : add,          // user defined names
subFunction : sub}          to access add, sub func.
```

8/10/2024.

LECTURE # 04

→ Modules:

↳ Built-in Module (File system).

```
const fs = require("fs");
```

```
fs.writeFileSync("./test.txt", "WAF");
```

↳ Blocking.

```
fs.writeFile ("./test.txt", "WAD", (err) => {});
```

↳ Non-blocking — Asynchronous, Callback func as a parameter. (Must).

- Multithread pool.
  - used for sync code.
  - if too many threads in the pool the system may become unresponsive.

→ Reading File:

↗ Encoding.

```
const result = fs.readFileSync("path", "UFT-8");
```

↳ we can store value using this. It returns value.

```
fs.readFile("./email.txt", "uft-8", (err, data) => {
    console.log(data);
});
```

↳ Doesnot return value. Need to deal with data inside callback function.

* Asynchronous Method does not return anything.

-> Append File:

```
fs.apendFileSync("./test.txt", '\nWAF');

                    new Date().getFullYear().
                                    toString()
fs.apendFile("./test.txt", '\n 2024', (err)=>
            {});
```

-> copy File:

```
fs.cpSync("./test.txt", "./testcopy.txt");

fs.cp("./test.txt", "./testcopy2.Txt", ()=>{});
```

OR

```
fs.copyFile();
fs.copyFileSync();
```

-> Delete File:

```
fs.unlink("./testcopy.txt", (err)=>{});
~~fs.unlink("./~~
fs.unlinkSync("./testcopy2.txt");
```

-> File Stats:

```
const stat = fs.statSync("path");
fs.stat("Path", (err,data)=>{ // });
```

→ Make Directory:

```
fs.mkdirSync("new");
fs.mkdir("___", ()=>{});
fs.mkdirSync("new/2023/1", {recursive: true});
```

↳

for nested folders.

Remove Directory.

Force property. force: true.

→ Web Server:

```
const http = require("http");

const server= http.createserver((request,response){
  console.log("New request is received");

  response.end("Message on Server");
});
```

Tuesday 22/10/2024.

LECTURE No. 5

→ Modules:

- URL — Query Parameters

  /products $\boxed{?}$ $\underbrace{\text{username = umer \&\& id = 1}}_{\text{Query Parameters.}}$

- HTML as a response
- HTML Template.
- Path Module
- OS Module
- Event — Event Emitter.

★ Path Module (Built-in module).

```
const path = require ("path");
  console.log (__ filename );
~~path~~          __ dirname
  console.log(path.basename(--filename));
           "      .ext name           "
  console.log(path.parse(--filename));

   const file = path.parse("______");
                   ↘ It returns an obj.
```

```
console.log( path.join ("folder1", "folder2", "home.html");
```

path.resolve?
path.format.

→ OS Module. → os.totalmem();
os.freemem();

→ Event Module.

Lecture no. 6 (Lab) → Task 1. 25/10/2024

Tuesday. 29/10/2024.

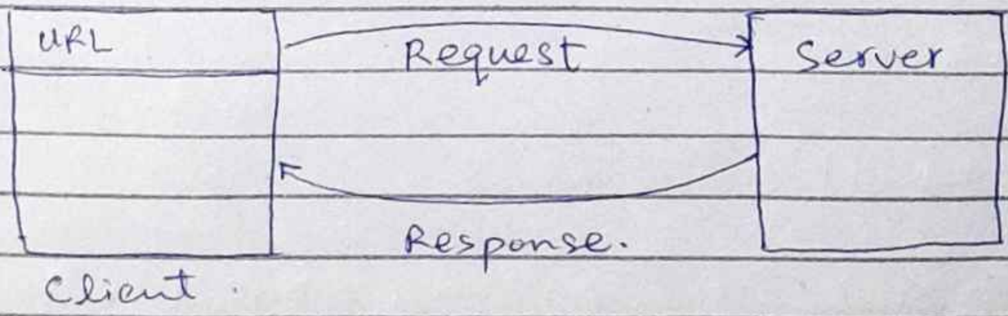
LECTURE No. 7.

→ HTTP Methods :

SSR → Server Side Rendering.

1) GET :

Get Request — want some data from server.



2) POST :

POST Request — send and mutate data.

Exp: Signup Forms.

form data goes along with request to server.

* Express is framework that uses Node js arhitecture

3) PATCH

4) PUT

5) DELETE.

[// npm uninstall express
L. To uninstall package.

→ Versioning / Semantic Versioning

^ 4.21.1 → Crucial.

First Part → 4. //Major Release / Breaking update.

Second Part → 21 // Bug fixes (Recommended).

Third Part → 1 // Minor fixes (optional).

→ New feature - e.g new route added.

// npm i express@version
↳ we can use exact version Through This method.

1.0.0 → 1.0.1
1.0.1 → 1.0.2.] Minor.

1.0.2 → 1.1.0] Feature Added.

^ → Carrot Symbol - Update 2ND & 3RD Part but doesnot change major version

~ → only Minor fixes.

4.18.3
4.18.4 $\boxed{4.19.0}$ X.
4.18.5

"express": "4.0.0 — 5.1.1"
" : " >=4.0.1 < 4.99.999"
" : " latest"

^4.18.x

→ REST API:

- Restfull API.

- GET/users → HTML response (get and return user data).
- GET/api/users → JSON response.
  ↳ API Tells we have to send JSON data.

SSR — Server Side Rendering. (Fast).
CSR — Client Side Rendering.

⇒ mockaroo.com // To generate mock data.

for dev dependencies

npm i packagename $\underbrace{\text{--save-dev}}_{\text{flag for dev dependgency}}$

Friday. 01/11/2024

LECTURE NO. 8 (Lab)

→ POSTMAN

→ DYNAMIC PATH PARAMETER.

GET /api/users/1.
" " /2.
POST
PATCH
DELETE.

C - create
R - Read
U - update
D - Delete.

'...' → spread operator. (combines list with array of obj).

Tuesday. 5/11/2024.

LECTURE No. 9

→ REST Architecture:

REST — Representational State Transfer.

- an architectural style that provides standards to communicate b/w computer systems.

① Separate API's into logical resources.
↳ an object or representation which has data associated with it.
users, movies, books, products.

② ~~Explore~~ Expose structure resources based url
http://localhost:8080/api/users.

③ HTTP methods.

CRUD.

R. GET/api/users/1 → Path Parameter.
↳ shows json data response. Read
GET/users

* Isend JSON Data ?.

C POST /api/users.
verb Noun. Create
POST /users.

U PUT/api/users/id → complete obj update.
PATCH/api/users/id → just 1 or 2 attributes.

D DELETE /api/users/id Delete.

*. GRAPHQL API ?

④ → JSON Data:
↳ Array of object.

[{___}, {___}, ---].

Id: " ";
first_name: " ";

Friday. 08/11/2024.

LECTURE No. 10 (Lab)

→ Middleware:

- manipulate request and response object before sending the response.

* Custom Built Middleware.

* 3rd Party Middleware. → app.use(express.json())

Paraenthesis show its a 3rd Party MW

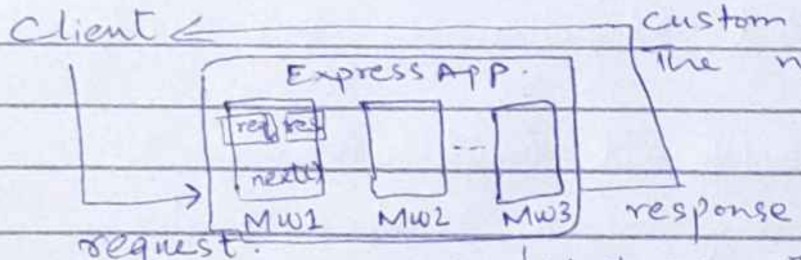
- Route Handler function is also a middleware.
- Last Middleware has to be a Route Handler.
- Custom Build MW has Three parameters.

(req, res, next).

returns a middleware func

↳ we use next() func in custom middleware to invoke the next middleware

└ can be any route handler



↳ last MW must be RouteHandler.

Example:

```
const logger= (req, res, next)=>{
    console.log("Custom Middleware invoked.--");
    next();
}
```

↳ // specify next() must ~~Be~~ otherwise infinite sending request response will be generated on Postman.

```
app.use(logger);
```

→ Ordering of Middleware ~~also~~ in code is also imp. Routes above middleware will not be impacted by the middleware.

toISOstring()?

npm i morgan

Tuesday

12/11/2024

LECTURE No. 11

→ Morgan:

- npm i morgan
- 3rd Party Middleware
- Allows us to see the request data in console
- app.use(morgan());

Q: How to invoke different Middlewares for different Routes?

→ Mounting Routes on Different MiddleWares.

- Apply middleware on a certain routes.
- Create a router for each resource
  - → users
  - → products
  - → books
  - → movies

```
const userRouter = express.Router();

      app.route
userRouter.~~app~~ route ("/api/users").get(getAllusers).post(
   createUsers);

   app.use('/api/users', userRouter)

   userRouter.route('/' ).get(..).post(....);
```

→ MONGO DB.

SQL (Relational)

- Tables
- Each Tables contains rows

ID	Name	Age	Email

- We use JOINS in SQL to extract data from Multiple Tables.

MongoDB (No SQL).

- Collection
- Each Collection has Document objects.

```
{ "_id": 1.
  name:
  Age:
  Email:
}
```

This JSON object is a document object.

- Doesn't Enforces schema
- It is more flexible because it's Schema-less DB.
- We can place object with objects → Nested Schema
- Embedded Document.

* MONGODB.

→ A document-oriented non-relational NOSQL database.

→ Store Data ~~is~~ in a document.

  └ Each document has field (key value pairs).

  └ Each document is a json object.

→ flexibility — Schema-less

→ Performance ⟵ embedded document
Indexing
Sharding.

→ Free and open Source.

Examples: Embedded Document.

```
{
    "_id": 1,
    "name": "umer",
    "email": "umer@gmail.com",
    "courses": ["WAF", "CC", "NCP"]
    "courseindex: [ {
                    "id": 1,
                    "title": "WebFrame"
                    "CrHrs": 3
                    "enrStudents": _
                    } , {_id - - - } ]
}
```

- Internally It's in BSON format.

Commands:

- Show dbs
- use dbname.
- db.courses.find()
- db.courses.Insertone() // To insert document in collection

```
. db.createCollection("collectionname"). // create new collection
. db.courses.drop(). // Drops collection "courses".
. db.courses.insertMany([{id:1}, {  }, {    }])

. db.courses.find({title: 'OOP'})
                   ↳ filter.
```

→ Insert Document

→ Fetch Document.

→ Update Document.

→ Delete Document.

```
                            ↗ filter
. db.courses.updateOne({title: 'oop'}, {$set: {enrStudents:
                                                    '100'}})
. db.courses.updateMany({title: 'OOP'}, {$set: {prereq: 'PSP'}})

. db.courses.deleteOne({title: 'COAL'})
. db.courses.deleteMany({title: 'OOP'})
```