Name : Navadiya Nensi B.
Roll No : 22

Topic : Express
Date : 2/9/2023

PAGE NO. : 1
DATE : / /

Assignment : 2

Que : 1 Write an assignment an express framework with following topics :

→ Introduction :

- Express.js is a web framework for Node.js. It is fast, robust and asynchronous in nature.
- Express is a fast, assertive, essential and moderate web framework of Node.js. You can assume express as a layer built on the top of the Node.js that helps manages a server and routes. It provides a robust set of features to develop web and mobile application.
- Let's see some of the core fectures of express framework :-
- It can be used to design single-page, multi-page and hybrid web application.
- It allows to setup middlewcure to respond to HTTP request.
- It allows to dynamically render HTML pages based on passing arguments to template.

→ Why use express :

- ultra fast I/O
- Asynchronous and single threaded
- MVC like structure
- Robust API makes routing easy
- File : basic_express.js

```
var express = require("express");
var app = express();
app.get("/", function (req, res)
{
    res.send("Welcome to JavaIn");
});

app.listen(8000, () => {
    console.log("server listing on port 8000");
});
```

⟹ Middlecoare :-

- Express.js middlecoare are different types of functions that are invoked by the Express.js routing layer before the final request handler. As the same specified, middlecoare appears in the middle between an initial request and final intended route. In stack, middlecoare functions are always invoked in order which they are added.

→ What is middlecoare function :-

- Middlecoare functions are that access to get request and response object in request - response cycle.

- It can perform following task :-

- It can execute any code
- It can make changes to the request and response objects.
- It can end the request - response cycle.

→ File :- simple.express.js

```
var express = require ("express");
var app = express();
app.get("/", function (req, res) {
    res.send ("Welcome to express");
});

app.use (function (req, res) {
    Console.log ('%s %s', req.method, req.url);
    next();
});

var server = app.listen (8000, function()
{
    console.log ("Example of this app is listing
                on 8000");
});
```

⟹ Express Validator :-

- Express validator is a set of express.js middleware that wraps the extensive collection of validators and sanitizers offered by validator.js.
- It allows you to combine them in many ways so that you can validate and sanitize your express requests, and offered tools to determine if the request is valid or not, which data was matched according to your validators, and so on.
- You can install this package by using this command :-

```
npm install express-validator
```

→ E.g. Html file :-

```
<form action = "SaveData" method = "post">
<pre>
  Enter Email :- <input type = "text" name = "email"><br>
  Enter password :- <input type = "password" name = "pass">
                                                    <br>
  <input type = "Submit" value = "Submit form">

</pre>
</form>
```

→ index.js :

```
const {check , validationResult } = require ('express-
                                        validator');
const express = require ('express');
app. express ();
app.get ("/", function (req, res) {
    res. sender ("sample form");
});

app.post ('/saveData', [
  check ('email', "Email length should be 10 to 30
        characters"). is Email (). is length ({ min : 10 ,
                                        max : 30 }),
  check ('Pass', "Passcoord length must be 8 to 10
                                        characters').
        islength ({min : 8 , max : 10 })
], (req, res) => {
const errors = validationResult (req);
if (! errors. isEmpty ()) {
    res.json (errors)
}
else {
    res. send ("successfully validated");
}
});

app.listen (8000, () => {
    console.log ("server listing on port 8000");
});
```

⟹ Template Engine :—

- A template engine enables you to use static template files in your application. At runtime, the template engine replace variables in a template file with actual values, and transform the template into an HTML file sent to client. This approach makes it easier to design an HTML page.
- Some popular template engines that work with Express are pug, mustache and EJS. The express application generator uses jade as it's default, but it also supports several others.
- To render template files, set the following setting properties :

- Views, the directory where the template files are located. e.g. app.set ('views', './views')
- View engine, the template engine to use. For example, to use the pug template engine: app.set ('view engine', 'pug').

→ To install package following command use :-

npm install pug --save

app.set ('view engine', 'pug')

- Create pug template file named index.pug in the views directory, with following command :

```
html
  head
    title = title
  body
    h1 = message
```

- app.get ('/', (req, res) => {
    res.render ('index', {title: 'Hey' message: 'Hello there !'})
  })

- When you request to home page, the index.pug file will be rendered as html.

⇒ REST API :-

- An API is always needed to create mobile applications, single page applications, use AJAX calls and provide data to clients. An popular architectural style of how to structural and name these APIs and the endpoints is called (Representational state Transfer) REST.
- Restful URIs and methods provide us with almost all information we need to process a request.

- const express = require ('express');
const app = express();
const mongoose = require ('mongoose');
mongoose.
connect ('mongodb://localhost:27017/db');
const db = mongoose.connection;

app.get ('/', (req, res) => {
Book.find ({})
.then ((books) => {
res.send (books);
})
.catch (err) => {
res.status (400).send ('ERR')
})
});

app.post ('/', (req, res) => {
console.log (req.body).
var books = new Book ({
name = req.body.name,
price = req.body.price,
quantity : req.body.price
});
book1.save ();
.then ((book) => {res.status (200).send (book); })
.catch ((err) => if (err)
return console.erro(err); })

})

⟹ API Security best Practices :-

- To protect API following best - proticies will be usefull :-

→ Use HTTP methods API Routes :-

- imagine, you building a Node.js Restful API for creating, updating or retriving data.
- As a best practice, your API should always use nouns as resource identifiers.

- Routing can look like :-

- POST /users : /id
- GET /users
- GET /users : /id
- DELETE /users : /id
- PUT /users : /id

→ Use HTTP status code correctly :-

- If something goes coroing while servicing a request. you must set the correct status code for that in response :-

- 2xx, if everything was okay,
- 3xx, if resourse was moved,
- 4xx, if request cannot be fulfiled because of a client error

- 5xx, if something went wrong on the API side

→ Use HTTP headers to send metadata :-

- To attach metadata about the payload you are about to send, use HTTP headers, Headers like this can be information on :

  - Pagination
  - rate & limiting
  - or authentication

- If you need to stop or set any custom metadata in your headers, it was best practice to prefix them with x. For example, if you were using CSRF tokens, it ~~use~~ was a common way to name them x-Csrf-Token.

→ Black Box testing to your API :-

- Black box testing is a method of testing where the functionality of an application is exclaimed.

```
- const request = require ('supertest')
describe ('GET /users:/id', function () {
    it ('returns a user', function() {
        return request (app)
            .get ('/user')
            .set ('Accept', 'application/json')
```

```
.expect (200, {
    id: '1',
    name: 'John'
}, done)
})
})
```

→ Do JWT - Based, Stateless Authentication :-

- As your REST API must be stateless, so does your authentication layer for this, JWT is ideal.
- JWT consist of three parts :-

- Header : Containing the type of the token and hashing algorithm

- Payload : Containing the claims
- Signature

→ E.g. :

```
Const koa = require('koa');
const jwt = require('jwt');
const app = koa();
app.use(jwt({
    secret: 'very-secret'
}))

app.use(function * () {
    this.body = {
        secret: '42'
    }
})
```

- also you always have to make sure that all your API endpoints are only accessible through a secure connection using HTTPS.

⟹ Types of Tokens and their usage :—

- Types of tokens :

1. Authentication Token (JWT - JSON Web Token) :

- JSON web token are a popular way to securely transmit info between parties as a JSON object. JWT is often used for authentication purpose in web applications.

- Usage : JWTs are generated upon successfull login. They contain information like user ID, expiration time, and other claims. client stores JWT and include it in headers of subsequent request to authenticate it.

2. Access tokens :

- Access tokens are used to grant access to specific resource or operation on a server.

- Usage : When an access token expires, a fresh token can exchanged for a new access token withought requiring user to log in again.

3. Refresh token :

- Refresh tokens are often used in conjuction with access tokens to enable longer-lived sessions without requiring the user to re-enter credenticals.

- Usage : After successful authentication, a user is provided with an access token.

4. CSRF - Tokens (cross-sites Request Forgery) :

- Used to protect against cross-site forgery attack by ensuring that request are only mode ~~fo~~ from the same website or application.

- Usage : Servers generate a unique CSRF token and include it in the response.

5. Bearer tokens :

- Are the type of access tokens often that used in authentication headers to grant access to protected resource..

- Usage : In HTTP headers, the "Authentication" headers is used to include a bearer token.

6. Custom token :

- Depending on your application's specific requirements, you might create custom token formats for various purposes. like session management or user-specific tokens.

- Usage : Custom tokens could be used to implement various authenication or authentication strangies that cater to your application's unique needs.

Que : 2
[B] Function to Proced CRUD Operation :—

(1) Database connectivity :

mongoose.connect("url", <auther params});

This code will execute connectivity with the database when server starts.

(2). findById :

- The query find a record into a table by passed object id from the client side there are several ways to capture an id to find in table.
- We can require id an query param and also in body but getting datain body will require request method post.

Ex : 1 : req.query.id
2 : req.params
3 : req.body.id (only for - Post)

(3) res.json(obj) :
- The syntax is used to send responce to client side in json format.

(4) .save() :
- The method refers to smethod to save data on changes to db. we can use after sharing saved object into variable as below.

```
const user = new userschema (create user)
let users = accait user.save()
```

(5) .findone() :
- The method finds an object with given key value pairs returns the whole single object from the table.

```
user findone ((emcil: 'nensi@gmcil.com)
```

- It will return it finds an record in which emcils is nensi@gmcil.com

(6) FindByIdAndUpdate();

```
userschema.AdByIDANDUpdate ({_id : 123})
```
↑
Table

(7) findById And Delete():

```
userschema.findById And Delete ({_id; 123})
```

(8) - find()

```
const users = userschema.find()
```

Que-3

[C] Mongoose schema, model, Document, and API CRUD, search in Express App.

(1) Mongoose schema / Model

```
const mongoose = require("mongoose");
const taskschema = new mongoose.schema({
  title: String
  description: string
});
const Task = mongoose.model('Task', taskSchema)
```

(2) Search in Express

```
router.get('/tasks/search', Async (req, res)
  => { const { title } = req.query;

try{
  const tasks = await Task.find({ title:
    new RegExp (title, 'i') });
  res.send (tasks);
```

```
} catch (error) {
        res.status(500).send(error);
    }
});
```

Que: 4

[D] Mongoose Schema datatype, validation :

→ Data type :

String : For text data
Number : For numeric data
Date : date and time
Boolean : True / False
Array : Array value
Objectid : references doc with same or different collections
Mixed : flexible and unstructured data
Buffer : binary data (image files)

→ Validation :

(1) Built - in validators :

```
const userschema = new mongoose.schema({
  username : {
      type : String,
      required : True,
      minlength : 3,
      maxlength : 20,
```

```
        unique : true
    },
    email : {
        Type : string,
        required : true,
        match : /^st@/st\.1sts/,
    },
    age : {
        type : number,
        min : 18,
        max : 100
    }
});
```

Que:5 [E] Schema Referencing and querying i :

(I) Schema Referencing :

- install mongoose : npm install mongoose
- model

```
const mongoose = require("mongoose");
const authorschema = new mongoose.schema({
    name : string,
    books : [{ type : mongoose.schema.Types.
            object Id, ref : 'Book' }],
});
```

```
Const bookschema = new mongoose.schema({
    title : string,
    Author : {type : mongoose. schema : Types,
            objectId , ref : 'Author' },
});
```

```
const Author = mongoose.model ('Author', uthorsce);
const Book = mongoose.model ('Book', bookschema);
```

(2) Querying with Schema references :

```
const express = require ('express');
const router = express.Router();
const Author = require ('../models/Author');
Const Book = require ('../models/book');

router.get ('/books-by-author/:authorId', async
        (req, res) => { try {
    Const authorId = req.purums.authorId;
    const books = await Book.find ({ Author :
                                    authorId)

        res.JSON (books);
    }
    catch (error) {
        console.error (error);
    res.status (500).json ({error : 'internal
                        server error' });
    }
});
```

```
// Creting Author

router.get ('/author-of-book/:bookId, async
                                (req, res) => {
  try {
    const bookId = req.params.bookId;
    const book = await Book.findById(bookid,
                    popular('author');
    res.json(book.author);
  }
  catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Internal
                              server error' });
  }
});

module.exports = routers;
```