Sub:- Express Rollno:30 [ICT-3]

Assignment :- 2

A] write an assignment on Express framework with following topic

1] → Introduction :-

Express.JS is a web framwork for node.JS it is fast robust and asynchronous in nature.

→ Express is a fast, assertive, essential and moderate web framwork of node.JS you can assume express as a layer built on the top of the node.JS that hips manages a server and routes It provides a robust set of features to decrop web and mobile application

→ Let's see some of the core features of Express framework :

→ It can be used to design single-page multi-page and hybrid web Application

→ It allows to setup middleware to respond to HttP Reluest

→ 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 java");
  });

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

2] Middleware :-

—> Express is middleware are different types of functions that are invoked by the Express.J routing layer before the final request handler as the same specified middleware appears in the middle between ~~appea~~ an initial request and final intended rout in stock middleware functions are always involved in order which they are added

—> what is middleware function :

— middleware function are that access to get request and responce object in request - response etcle

—> It can perform following task :

—> It can create 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"),
});
```

3] 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 the in many ways so that you win validate and sanitize your express requests and offered tools to determine if the request is valid or not, which

data was mathed aording to your Validatoss and so on

—> You can install this Pakeges by using commands :

npm install express-validator

Eg : 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");
</pre>

</form>
```

—> index.js

```
const { check, validationResult } =
        require ('express-validator');
const express = require ('express');

app.express ();
app.get ("/", function (req, res) {
    res.sender ("sempleform");
  });
```

```
app.post('/saveData', [
    check('email', "Email length should be 10 to
        30 character").isEmail().isLength
        ({min:10, max:30}),
    check('pass', 'Password length must be 8 to 10
        character').
        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");
});
```

4] 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:

→ 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'})
})

when you request to home page, the index.js Pug file will be rendered as html

5] REST API :-

→ In 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 those APIs and the endpoints is called (Representational state Transfer) REST

→ Restful URLS 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');
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((book1) => { res.status(200).
                send(book); })
      .catch((err) => if(err)
                return console.error(err);
    })
})
```

→ API security best Practices :-

→ To Protect API following best practicies will be useful

→ use HTTP methods of API Routes :-

— Imagine, you buiding a node.js Restful, API for creating, updating or retriving data

→ As a best practice your API o 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 wrong while servicing a request you must set the correct status code for that in response.

2xx if everything was okay
3xx if resource was moved
4xx if request cannot be fulfilled because of a client error
5xx if something went wrong on the API side.

→ use HTTP headers to send method :

To attach metadata about the payload you are about to send use HTTP header 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 token it values 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 explained.

→
```
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: 'Rutvi'
            }, done)
        })
    })
```

7] Tokens Types and usage :

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 application

→ usage :-

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

2] Aqess tokens :

access tokens are used to grat access to specific resource or operation on server

—> usage : when an access token expires a feqsh token can exchanged for a new access token withought requiring user to log in again

3] Refresh token :

Refresh token are often use in conjuction with access tokens to enable longer-lived sessions without requiring the to re-enter credentials

—> usage :

After successful authenticaition a user is provided with an access token

4] CSRF - Tokens (cross-sites Request Forged):

used to protect against cross-site forged attack by ensuring that request are only mode 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 resourse

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

6] Custom token :

Depending on your applications specific requirements you might create custom token like ossion managment or user-specific tokens

→ usage :

Custom tokens could be used to implement various authentication or Authentication your application's unique needs.

B] function to procced CRud operation

1] Database connectivity :

mongoose.connect("uri", {authes params});

this code will execute connectivity with the database when seaver starts

2] .findBYId :

The quest 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 on quest param and also in body but getting datain body will require request method post.

Ex 1= req.quest.id
2 = req.params
3 = req.body.id (comm 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 or changes to db. we can use after storing saved object into variable as below

const user = new userschema (createuser)
let users = await user.save()

5) . Findone () :

The method finds an object with given key value pairs returns the whole single object from the table.

user findone ({email : 'Rutvi@gmail.com'})

it will return it finds an record in which emails is Rutvi@gmail.com

6] FindByIdAndUpdate ();

userschema.AdBYIDAndUPdate ({_id : 123})
Table

7] Find By id And Delete ();

user shema.Find Byid And Delete ({_id ; 123})

8] - Find ()

const users = usersschema.Find();

c] Mongoose Schema, Model, Document, and API CRUD, Search in Express APP.

1) Mongoose Schema & Model

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

2) Model Search in Express

```
routes.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);
  }
});
```

D] Mongoose Schema datatype, validation:

→ Datatype :

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: /^\S+@\S+\.\S+$/,
    },
  age : {
        type : number,
        min : 18,
        max: 100
    }
});
```

E] Schema Referencing and querying in

1] Schema Referencing :

- install mongoose : npm install mongoose
- model

```
const mongoose = require('mongoose');
const authorSchema = new mongoose.Schema({
  name: String,
  books : [ {type: mongoose.Schema.Types.ObjectId
            ref: 'Book' }],
});
const bookSchema = new mongoose.Schema({
  title : String,
  Author: { type : mongoose.Schema.Types.
        ObjectId, ref: 'Author' },
});
```

```
const Author = mongoose.model('Author', authorSchema);
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.params.authorId;
       const books = await Book.find({ author: authorId
       res.json(books);
     }
     catch (error) {
       console.error(error);
       res.status(500).json({ error: 'internal server' });
     }
});

// creting Author

router.get('/author-of-book/:bookId', async (req, res)=
  { try {
       const bookId = req.params.bookId;
       const book = await Book.findById(bookId).
            populate('author');
       res.json(book.author);
     }
```

```
    catch (error) {
        console.error(error);
        res.status(500).json({ error: 'Internal
                server error' });
    }
});
module.exports = routes;
```