Node JS

1.Project Creation

- ☐ Create a directory > **mkdir event-mgmt-service** in powershell or cmd prompt.
- ☐ Open vs code and open project folder.
- ☐ From the menu open terminal (powershell or cmd prompt)
- ☐ In the terminal type > **npm init -y**
- ☐ Create necessary files and directories.
    1. config -> db.js
    2. routes -> index.js
    3. controllers -> auth.controllers.js, event.controllers.js, report.controllers.js
    4. middlewares
    5. models ->users.model.js, events.model.js, enrolls.model.js
    6. server.js
    7. .env

2.Install Necessary Dependancies

- ☐ In the terminal, type >**npm install** dependencies_name
    1. **body-parse**r - req.body handler
    2. **cors** - Cross-Origin Policy
    3. **dotenv** - to handle .env file
    4. **express** - framework
    5. **helmet** - encapsulate headers - config settings
    6. **jsonwebtoken** - jwt token generator
    7. **sequelize** - nodejs ORM for DB — refer https://sequelize.org/
    8. **mariadb** - connector for mysql
    9. **nodemon** - for development purpose

3.Setup server

- ☐ Open server.js and type the following

```
const express = require("express");
const http = require("http");
const cors = require('cors');
const helmet = require("helmet");
const PORT = process.env.PORT || 8085;
const app = express();
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(helmet({crossOriginResourcePolicy: false}));
app.disable('etag');
app.use(cors());


// testing api for the server

app.get('/ems/1', (req, res) => {
    console.log(`Service is Working...on ${PORT}`)
    return res.send(`Service is Working...on ${PORT}`);
})

// initialized the server
app.listen(PORT, () => {
  console.log(`Server listening on ${PORT}`);
}
);
```

4.Config package.json

- ☐ Enter your name of the project and version.
- ☐ Modify script{}

```
    "scripts": {
```

"test": "echo \"Error: no test specified\" && exit 1",
"start": "node server.js",
"dev": "nodemon server.js"
},

- ☐ Finally start the server > **npm run dev**
- ☐ Note: if any changes done on .env, then you have to restart the service manually by type > **rs** and then click enter

5.Testing the server

- ☐ Open browser and type : localhost:8085/ems/1

Task1: Create a hardcoded Object and call it via api

1.create a sample object for events inside middleware directory(file name - sampleData.js

```
export const eventList = [
    {id:1,
    eventName: "CS Workshop",
    eventDate: "2025-10-30",
    eventLocation: 'Anna University CIT Campus',
    eventLink: 'www.annauniv.com/worship/cs/2025',
    eventVideoLink : 'https://www.youtube.com/watch?v=gbKY8MDrMC0',
    eventImageLink: '',
    publishFrom : '2025-07-01 00:00:00.000z',
    publishTo : '2025-10-20 23:59:59.000z',
    status: 1
    },
..
];
```

2. Import eventList from sampleData.js on server.js file and use it like

```
app.get('/ems/event/list', (req, res) => {
    // creating sample list
    return res.json(eventData);
})
```

3. On browser type > localhost:8085/ems/event/list.

Task2: Create an api to get event enrolled data

1.create a sample object for eventEnrolled data inside middleware directory(file name - sampleData.js)

```
    export const eventEnrolledData = [
{
        id: 101,
        userId: 331,
        eventId: 1,
        name: "Jeevin",
        gender: 1,
        emailAddress: 'jeevinck@jr.org',
        mobileNumber: '9876543210',
        country: 'India',
        state: 'Tamil Nadu',
        district: 'Theni',
        status: 1 // 1 - applied, 2 - paid, 3 - attended
    },
```

and create an api endpoint as **/ems/enrolled/list** (repeat step 2 and 3)

Authentication

1.Login

- ☐ Open another tab in terminal and install bcryptjs library
- ☐ > **npm i bcryptjs**
- ☐ Go to controllers ->auth.contolllers.js

```
const bcrypt = require('bcryptjs');

const users = [];

exports.login = (req,res) => {}
exports.register = (req,res) => {}
```

Modify the code as

```
const bcrypt = require('bcryptjs');

const users = [];

exports.login = async(req,res) => {
     const { emailAddress, password } = req.body;

    // Find the user in memory
    const user = users.find(u => u.emailAddress === emailAddress);
    if (!user) {
        return res.status(400).send('Email is not found');
    }

    try {
        // Compare the provided password with the stored hashed
password
        if (await bcrypt.compare(password, user.password)) {

            res.send('Login successful');
        } else {
            res.status(400).send('Invalid username or password');
        }
    } catch {
        res.status(500).send('Error logging in');
    }
}

exports.register = async(req,res) => {
    try {
        const { username, password } = req.body;

        // Check if user already exists
        const existingUser = users.find(user => user.username ===
username);
```

```
        if (existingUser) {
            return res.status(409).send('Username already exists');
        }

        // Hash the password (ALWAYS hash passwords)
        const hashedPassword = await bcrypt.hash(password, 10);

        // Store the user data in memory
        const newUser = {
            id: Date.now().toString(), // Simple unique ID
            username,
            password: hashedPassword
        };
        users.push(newUser);

        res.status(201).send('User registered successfully');
    } catch {
        res.status(500).send('Error registering user');
    }
}
```

Routing API

- ☐ Go to routes-> [index.js]() and type

```
const express = require('express');
const router = express.Router();
const Auth = require("../controllers/auth.controllers");

//Auth API

router.post('/register',Auth.register);
router.post('/login',Auth.login);

module.exports = router;
```

- ☐ Go to [server.js]() and add

```
const router = require('./routes/index');
```

```
app.use("/ems/v1",router);
```

- ☐ Open POSTMAN app
- ☐ Click ‘+’ to create a new request
- ☐ Choose “POST” and enter the URL :
  localhost:8085/ems/v1/auth/register
- ☐ Choose the **Body** tab, then select **raw** option and create a JSON Object

```
{
"emailAddress" : "jeevin@gmail.com",
"password": "12345678"
}
```

- ☐ click the send button on the postman.
- ☐ Similarly modify **localhost:8085/ems/v1/auth/login** and verify.

Create a Database:

- ☐ Open phpmyadmin
- ☐ Click New to create a new database -> **events_db**
- ☐ Create table -> **users**

1.id Primary	int(11)
2.email_address Index	varchar(50)
3.password	text
4.full_name	varchar(100)
5.date_of_birth	date
6.gender	smallint(10)
7.mobile	varchar(10)
8.occupation	text
9.address_line_1	text
10.address_line_2	text
11.district	text
12.state	text
13.country	text
14.status	smallint(6)

15.created_at	datetime

Connecting Service with Database

- ☐ On VS code move to config->db.js file
  Refer this link for more info:
  https://sequelize.org/docs/v6/getting-started/

```
const { Sequelize } = require('sequelize');

const eventDB = new Sequelize('events_db', 'root', '', {
  host: 'localhost',
  dialect: 'mariadb'
});


module.exports = eventDB;
```

- ☐ To test the connection,
  Add this line in server.js

```
const eventDB = require ('./config/db')
...
..
// initialized the server
app.listen(PORT, async() => {
  console.log(`Server listening on ${PORT}`);
  try {
  await eventDB.authenticate();
  console.log('Connection has been established
successfully.');
} catch (error) {
  console.error('Unable to connect to the database:', error);
}
}
);
```

If you see
Executing (default): SELECT 1+1 AS result
Connection has been established successfully.

then it is connected to the database.

Auth Controllers

To create a functions to handle authentications login and register

- ☐ Go to controllers -> auth.controllers.js
- ☐ Import sequelizeand DB config like

```
const { QueryTypes } = require('sequelize');
const db = require('../config/db');
```

Update register function as below

```
exports.register = async(req,res) => {
    console.log("register request body", req.body);

    try {
        const { emailAddress, password, fullName, dob, gender,
occupation,mobile,addressLine1, addressLine2, district, state, country
            } = req.body;

        const existingUser = await db.Sequelize.query(`select email_address from
users where email_address=${emailAddress}`,
{type: QueryTypes.SELECT});

        if (existingUser) {
            return res.status(409).send('Email already exists');
        }
        const status = 1;
        const createdAt = new Date();

        let insertQuery =`INSERT INTO users (email_address, password, full_name,
date_of_birth, gender, mobile, occupation, address_line_1,
 address_line_2, country, state, district, status, created_at)
                         VALUES (:emailAddress, :password, :fullName, :dob,
 :gender, :mobile, :occupation, :addressLine1, :addressLine2, :country, :state,
:district,:status, :createdAt )`;

        const addUser = await Sequelize.query(insertQuery,{
            replacements:{emailAddress, password, fullName, dob,
                            gender, mobile, occupation, addressLine1,
                            addressLine2, country, state, district,
                            status, createdAt},
            types:QueryTypes.INSERT,
        });

        res.status(201).send('User registered successfully');
    } catch {
        res.status(500).send('Error registering user');
    }
}
```

then open postman or react app, then pass value to endpoint
localhost:8085/ems/v1/auth/register

```
{"emailAddress" : "jeevin@gmail.com",
"password" : "123456",
"full_name" :"jeevin"
 }
```

If got error   Try kula change panrom

Update the code

```
        const { emailAddress, password, fullName, dob, gender,
occupation,mobile,addressLine1, addressLine2, district, state, country
            } = req.body;
```

into

```
        const emailAddress = req.body.emailAddress;
        const password = req.body.password;
        const fullName = req.body.fulName;
        const dob = req.body.dob || '';
        const gender = req.body.gender || 1;
        const occupation = req.body.occupation || 'student';
        const mobile = req.body.mobile || '';
        const addressLine1 = req.body.addressLine1 || '';
        const addressLine2 = req.body.addressLine2 || '';
        const district = req.body.district || '';
        const state = req.body.state || '';
        const country = req.body.country || '';
        const status = 1;
        const createdAt = new Date();
```

After fixing typing issues with console.log, verify the data is entered by going to phpmyadmin, events_db -> users .

Now again click the send button on **POSTMAN APP**, it will send an error.

Login

- ☐ Move to auth.controllers.js -> exports.login section
- ☐ Update the code as below

```
exports.login = async(req,res) => {
    console.log("req.body", req.body)
     const { emailAddress, password } = req.body;
    const user = await eventDB.query(`select email_address, password from
users where email_address = :email`,
{replacements: { email: emailAddress },
type: QueryTypes.SELECT});

    if (user.length === 0) {
        return res.status(400).send('Email is not found');
    }

    try {
        // Compare the provided password with the stored hashed password
       // if (await bcrypt.compare(password, user.password)) {
       if(password === user.password){

            res.send('Login successful');
        }
        else {
            res.status(400).send('Invalid username or password');
        }
    } catch {
        res.status(500).send('Error logging in');
    }
}
```

After fixing bugs on the code, when you try on **POSTMAN APP**, you will receive "**Login Successfull**".

Since I have not added extra information while creating a user, now we are going to update the user entry.

Update a user entry

Code update

- ☐ Copy and paste the exports.register code and rename exports.register as **exports.updateUser** in auth.controllers.js file.
- ☐ Now update the code as below

```
exports.updateUser = async(req,res) => {
    console.log("update user request params", req.params);
    console.log("update user request body", req.body);
    const id = req.params.id;

  try {

    const existingUser = await eventDB.query(`select * from users where id =
:id`,
                                                    {replacements: { id: id },
                                                    type: QueryTypes.SELECT});

    if(existingUser.length === 0)
    {
        return res.status(404).send("ID is not found");
    }

        const emailAddress = req.body.emailAddress ||
existingUser[0].email_address;
        const password = req.body.password || existingUser[0].password;
        const fullName = req.body.fullName || existingUser[0].full_name;
        const dob = req.body.dob || existingUser[0].date_of_birth;
        const gender = req.body.gender || existingUser[0].gender;
        const occupation = req.body.occupation || existingUser[0].occupation;
        const mobile = req.body.mobile || existingUser[0].mobile ;
        const addressLine1 = req.body.addressLine1 ||
existingUser[0].address_line_1;
        const addressLine2 = req.body.addressLine2 ||
existingUser[0].address_line_2;
        const district = req.body.district || existingUser[0].district;
        const state = req.body.state || existingUser[0].state;
        const country = req.body.country || existingUser[0].country;
        const status = req.body.status || existingUser[0].status || 1;
        // const createdAt = new Date();


            let updateQuery = `UPDATE users SET
                email_address = :emailAddress,
                password = :password,
                full_name = :fullName,
                date_of_birth = :dob,
                gender = :gender,
                mobile = :mobile,
                occupation = :occupation,
                address_line_1 = :addressLine1,
                address_line_2 = :addressLine2,
                country = :country,
                state = :state,
                district = :district,
                status = :status,
            WHERE id = :id`;

        const updateUser = await eventDB.query(updateQuery,{
            replacements:{emailAddress, password, fullName, dob,
                            gender, mobile, occupation, addressLine1,
                            addressLine2, country, state, district,
                            status},
            types:QueryTypes.UPDATE,
        });
        console.log("updateuser", updateUser);

        res.status(201).send('User info updated successfully for id: ',id);
    } catch(err) {
        console.log("err",err)
        res.status(500).send('Error while updating user');
    }
}
```

- [ ] Then create an api route to this function by the following steps
- [ ] Go to routes -> index.js and add
  router.put("~~/auth~~/update/:id", Auth.updateUser);
- [ ] Try it on POSTMAN App with this sample JSON

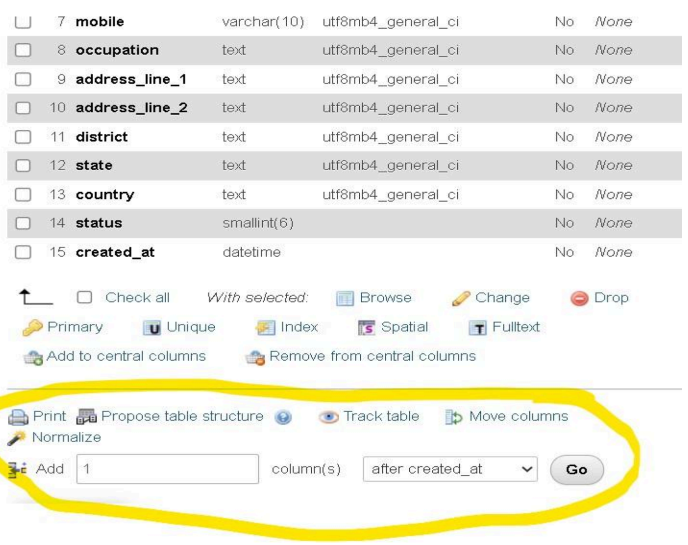
```
{"emailAddress" : "jeevin@gmail.com",
"password" : "123456",
"fullName" : "jck",
"dob" : "1900-06-01",
"gender" : 1,
"occupation" : "S/w Engg",
"mobile": "9876543210",
"addressLine1" : "2/27 JR Quarters",
"addressLine2" : "Block 2, Kannivilai",
"district" : "Thoothukudi",
"state" : "Tamil Nadu",
"country" : "India"
 }
```

- [ ] localhost:8085/ems/v1/auth/update/1 <- id is “1” the sample id
- [ ] Make sure the request sent by put method
- [ ] After debugging and fixing bugs, you will receive

```
User info updated successfully for id: 0
```

Adding Role

- [ ] We are going to add one field **“role”** to the entire system.
- [ ] First add **role - varchar(50)** on the users table in events_db database (phpmyadmin)
- [ ] On phpmyadmin, expand the users table and select columns



☐	7	**mobile**	varchar(10)	utf8mb4_general_ci	No	*None*
☐	8	**occupation**	text	utf8mb4_general_ci	No	*None*
☐	9	**address_line_1**	text	utf8mb4_general_ci	No	*None*
☐	10	**address_line_2**	text	utf8mb4_general_ci	No	*None*
☐	11	**district**	text	utf8mb4_general_ci	No	*None*
☐	12	**state**	text	utf8mb4_general_ci	No	*None*
☐	13	**country**	text	utf8mb4_general_ci	No	*None*
☐	14	**status**	smallint(6)		No	*None*
☐	15	**created_at**	datetime		No	*None*

Add 1 column(s) after created_at

```
ALTER TABLE `users` ADD `role` VARCHAR(50) NOT NULL AFTER `created_at`;
```

☐ Add the new field "role" on register and update functions wherever it is necessary (on all queries)

Likewise we have to do for

1.Event

- ☐ POST -> /event/add
- ☐ GET -> /event/list
- ☐ GET -> /event/:eventId
- ☐ PUT -> /event/:eventId

2.Enroll

- ☐ POST -> /event/:eventId/enroll
- ☐ GET -> /event/:eventId/list
- ☐ GET -> /event/:eventId/enroll/:enrollId
- ☐ PUT -> /event/:eventId/enroll/:enrollId

3.User (optional)

- ☐ GET -> /user/list
- ☐ GET -> /user/:id
- ☐ PUT -> /user/:id

4.Report (optional)

- ☐ GET -> /report/enroll/all

Table Schema for Tbl events 11 column

id	*int primary auto_increment*
event_name	*varchar(100)*
event_description	*text*
event_date	*date*
event_location	*varchar(200)*
event_link	*text*
event_video_link	*text*
event_image_link	*text*
publish_from	*datetime*
publish_to	*datetime*
status	*tinyint*

-> unique (event_name,event_date)

Table Schema for Tbl enrolls 16 column

id	*int primary auto_increment*
event_id	*int*
full_name	*varchar(250)*
email_address	*varchar(250)*
mobile	*varchar(10)*
country	*varchar(100)*
state	*varchar(100)*
district	*varchar(100)*
status	*tinyint*
meta_1	*text*
meta_2	*text*
meta_3	*text*
created_by	*varchar(100)*
created_at	*datetime*
updated_by	*varchar(100)*
updated_at	*datetime*

-> unique (event_id,email_address)

Please refer on your own

- Sequelize modals
- Bcrypt, JWT