

# 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 Dependencies

- ☐ In the terminal, type > **npm install** dependencies\_name
  1. **body-parser** - 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\" && 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.contollers.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 sequelize and 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**

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.fullName;
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 Successful**”.

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

### Update a user entry

- ☐ 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

<input type="checkbox"/>	7	<b>mobile</b>	varchar(10)	utf8mb4_general_ci	No	None
<input type="checkbox"/>	8	<b>occupation</b>	text	utf8mb4_general_ci	No	None
<input type="checkbox"/>	9	<b>address_line_1</b>	text	utf8mb4_general_ci	No	None
<input type="checkbox"/>	10	<b>address_line_2</b>	text	utf8mb4_general_ci	No	None
<input type="checkbox"/>	11	<b>district</b>	text	utf8mb4_general_ci	No	None
<input type="checkbox"/>	12	<b>state</b>	text	utf8mb4_general_ci	No	None
<input type="checkbox"/>	13	<b>country</b>	text	utf8mb4_general_ci	No	None
<input type="checkbox"/>	14	<b>status</b>	smallint(6)		No	None
<input type="checkbox"/>	15	<b>created_at</b>	datetime		No	None

☐ Check all    With selected:

---

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**

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

-> *unique (event\_name,event\_date)*

### **Table Schema for Tbl enrolls**

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

-> *unique (event\_id,email\_address)*



## Please refer on your own

- Sequelize models
- Bcrypt, JWT

### [events.model.js](#)

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

```
module.exports = (eventDB) => {  
  const Events = eventDB.define('events',  
    {  
      id: {  
        type: DataTypes.INTEGER,  
        primaryKey: true,  
        autoIncrement: true,  
        allowNull: false  
      },  
      event_name: {  
        type: DataTypes.STRING(100),  
        allowNull: false  
      },  
      event_description: {  
        type: DataTypes.TEXT,  
        allowNull: false  
      },  
      event_date: {  
        type: DataTypes.DATEONLY,  
        allowNull: false  
      },  
      event_location: {  
        type: DataTypes.STRING(200),  
        allowNull: false  
      },  
      event_link: {  
        type: DataTypes.TEXT,  
        allowNull: false  
      }  
    }  
  );  
  return Events;  
};
```

```

    },
    event_video_link: {
      type: DataTypes.TEXT,
      allowNull: false
    },
    event_image_link: {
      type: DataTypes.TEXT,
      allowNull: false
    },
    publish_from: {
      type: DataTypes.DATE,
      allowNull: false

    },
    publish_to: {
      type: DataTypes.DATE,
      allowNull: false
    },
    status: {
      type: DataTypes.TINYINT,
      allowNull: false
    }
  },
  {
    updatedAt: false,
    createdAt: false,
    timestamps: false,
    underscored: false,
  });

  return Events;
};

```

[events.controller.js](#)

```
const { QueryTypes, DataTypes, Sequelize } = require('sequelize');
const eventDB = require('../config/db');
const eventsTbl = require("../models/events.model") (eventDB)

exports.add = async(req, res) =>{
  console.log("req.body", req.body);
  const {
    eventName,
    eventDescription,
    eventDate,
    eventLocation,
    eventLink,
    eventVideoLink,
    eventImageLink,
    publishFrom,
    publishTo,
    status
  } = req.body;

  try{
    let result = await eventsTbl.create({
      event_name : eventName,
      event_description : eventDescription,
      event_date : eventDate || null,
      event_location : eventLocation,
      event_link : eventLink,
      event_video_link : eventVideoLink,
      event_image_link : eventImageLink,
      publish_from : publishFrom,
      publish_to : publishTo,
      status:status??1
    })
    console.log("event creation",result);
    return res.send("Event Created successfully")
  }
  catch(err){
    console.log("err",err)
    return res.status(504).send("Error in event creation")
  }
}
```

```
}
```

```
exports.list = async(req,res) =>{
try{
  const result = await eventsTbl.findAll({
    attributes:['id','event_name','event_description','event_date',
      'event_location', 'event_link', 'event_video_link',
'event_image_link',
      'publish_from', 'publish_to', 'status']
  });

  // let eventNames = [];

  // for(let i=0;i<result.length;i++)
  // {
  //   eventNames.push(result[i].event_name);
  //   console.log("events list",i, result[i].event_name);
  // }

  return res.status(200).json(result);
}
catch(err){
  return res.status(501).send("error in fetching event list",err)
}
}
```

```
exports.list = async(req,res) =>{
console.log("getting events active list")
try{
  const now = new Date();
  const result = await eventsTbl.findAll({
    attributes:['id','event_name','event_description','event_date',
      'event_location', 'event_link', 'event_video_link',
'event_image_link',
      'publish_from', 'publish_to', 'status'],
    where: {
```

```

        status: 1,
        publish_from: {[Op.lte]: now,},
        [Op.or]: [
            { publish_to: { [Op.gte]: now } },
            { publish_to: null } // Allow open-ended events
        ]
    }
},
);
return res.status(200).json(result);
}
catch(err){
    return res.status(501).send("error in fetching event list",err)
}
}

```

```

exports.getEvent = async(req,res) =>{
    const id = req.params.eventId;
    console.log("id",id);
    try{
        const result = await eventsTbl.findOne({
            attributes:['id','event_name','event_description','event_date',
                'event_location', 'event_link', 'event_video_link',
                'event_image_link',
                'publish_from', 'publish_to', 'status'],
            where:{id:id}})

        return res.status(200).json(result)
    }
    catch(err){
        return res.status(501).send(err)
    }
}

```

```

}

exports.updateEvent = async(req,res) =>{

  console.log("req body of update event", req.body);
  const { eventDescription,eventDate, eventLocation,
    eventLink, eventVideoLink, eventImageLink,
    publishFrom, publishTo, status
    } = req.body;

  let values = {
    event_name : req.body.eventName,
    event_description: eventDescription,
    event_date: eventDate,
    event_location: eventLocation,
    event_link: eventLink,
    event_video_link: eventVideoLink,
    event_image_link: eventImageLink,
    publish_from: publishFrom,
    publish_to: publishTo,
    status: status
  }

  try{
    const result = await eventsTbl.update(values,{
      where:{id:req.params.eventId}})

    return res.status(200).json(result)
  }
  catch(err){
    return res.status(501).send(err)
  }
}

```

## enroll.controllers.js

```
const eventDB = require('../config/db');
const enrollTbl = require("../models/enrolls.model") (eventDB);

exports.add = async(req,res) =>{
  console.log("enroll re body", req.body);
  const {
    eventId,
    fullName,
    emailAddress,
    mobile,
    country,
    state,
    district,
    status,
    meta1,
    meta2,
    meta3,
    createdBy
  } = req.body;

  try{
    let values = {
      event_id : eventId,
      full_name : fullName,
      email_address : emailAddress,
      mobile : mobile,
      country: country,
      state : state,
      district : district,
      status : status,
      meta_1 : meta1,
      meta_2 : meta2,
      meta_3 : meta3,
      created_by : createdBy
    }
  }
```

```

        const result = await enrollTbl.create(values)

        console.log("result", result);

        return res.status(200).send("Enrolled Successfully and enrolled id
:",result);
    }
    catch(err){
        console.log("error on enroll add", err);
        return res.status(504).send('Error:unable to enroll');
    }
}

exports.list = async (req,res) =>{
    try{
        const result = await enrollTbl.findAll({
            attributes:['id','event_id','full_name', 'email_address','mobile',
                'country','state', 'district', 'status', 'meta_1', 'meta_2',
'meta_3',
                'created_by','created_at', 'updated_by', 'updated_at']
        });

        return res.status(200).json(result);
    }
    catch (err){
        return res.status(501).send("error in fetching enrolled list",err);
    }
}

exports.getEnroll = async(req,res) =>{
    const id = req.params.eventId;
    const email = req.params.emailAddress;
    console.log("event id",id);
    try{
        const result = await enrollTbl.findOne({
            attributes:['id','event_id','full_name', 'email_address','mobile',
                'country','state', 'district', 'status', 'meta_1', 'meta_2',
'meta_3',
                'created_by','created_at', 'updated_by', 'updated_at'],

```



```

    where:{event_id:id, email_address: email}
    });

    let values = {
      id : result.id,
      eventId : result.event_id,
      fullName : result.full_name,
      emailAddress : result.email_address,
      mobile : result.mobile,
      country: result.country,
      state : result.state,
      district: result.district,
      status : result.status,
      meta1 : result.meta1,
      meta2 : result.meta2,
      meta3 : result.meta3,
      createdBy : result.created_by,
      createdAt : result.created_at,
      updatedBy : result.updated_by,
      updatedAt : result.updated_at
    }

    return res.status(200).json(values);
  }
  catch(err)
  {
    return res.status(501).send(err);
  }
}

exports.updateEnroll = async(req,res) =>{
  console.log("req body of updating enroll data", req.body);
  const { fullName,
    emailAddress,
    mobile,
    country,
    state,
    district,
    status,

```

```

        meta1,
        meta2,
        meta3,
        updatedBy
    } = req.body;
    let values = {
        full_name : fullName,
        email_address : emailAddress,
        mobile : mobile,
        country : country,
        state : state,
        district : district,
        status : status,
        meta_1 : meta1,
        meta_2 : meta2,
        meta_3 : meta3,
        updated_by: updatedBy,
        updated_at : new Date()
    }
    try{
        const result = await
enrollTbl.update(values,{where:{id:req.params.enrollId}});

        return res.status(200).json(result);
    }
    catch(err){
        return res.status(501).send(err);
    }
}

```

### [report.controllers.js](#)

```

const eventDB = require('../config/db');
const eventsTbl = require("../models/events.model") (eventDB);
const enrollTbl = require("../models/enrolls.model") (eventDB);

exports.report = async(req,res) =>

```

```

{
  try{
    const getEvents = await eventsTbl.findAll({where:{status:1}})
    console.log("getEvents length", getEvents.length);

    let data =[];

    for(let i=0; i<getEvents.length; i++)
    {
      let value = {
        eventId : getEvents[i].id,
        eventname : getEvents[i].event_name,
      }
      let getEnrollCount = await
enrollTbl.count({where:{id:getEvents[i].id}});
      value.eventCount = getEnrollCount;
      data.push(value);
    }

    return res.status(200).json(data);

  }
  catch(err)
  {
    return res.status(504).send("error in getting report")
  }
}

```

## Login with jwt

At the top of the file

```

require('dotenv').config()
const jwt = require('jsonwebtoken');
const secret = process.env.SECRET;

```

And add your secret key in .env file like SECRET='mysecretkey'

```
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, id,
role from users where email_address = :email`,
    {replacements: { email: emailAddress },
    type: QueryTypes.SELECT});

  console.log("USER", user);
  console.log("secret",secret);

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

  try {
    console.log("password",password);
    console.log("user.password", user[0].password);
    // Compare the provided password with the stored hashed password
    const verifyPassword = await bcrypt.compare(password,user[0].password);
    if (verifyPassword) {
      const token = jwt.sign({
        exp: Math.floor(Date.now() / 1000) + (60 * 60),
        data: {'email':user[0].email_address,'role':user[0].role, 'userId':
user[0].id}
      }, secret);

      return res.status(200).json({jwt:token,user:user[0].id});
    } else {
      return res.status(400).send('Invalid email or password');
    }
  } catch (err){
    console.log("error in logging in",err);
    return res.status(500).send('Error logging in');
  }
}
```

## Update password

```
exports.updatePassword = async(req,res) => {
  console.log("req.body", req.body)
  const { emailAddress, password } = req.body;

  const user = await eventDB.query(`select email_address, password, id from
users where email_address = :email`,
    {replacements: { email: emailAddress },
    type: QueryTypes.SELECT});

  console.log("USER", user);

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

  try {
    const hashedPassword = await bcrypt.hash(req.body.password,10);
    const userId = user[0].id
    const query = `UPDATE users SET password=:hashedPassword where
id=:userId`;
    const updatePassword = await eventDB.query(query,{
      replacements:{ hashedPassword,userId},
      types:QueryTypes.UPDATE,
    });

    return res.status(200).send('password updated successfully');

  } catch (err) {
    console.log("err in updating password",err);
    return res.status(500).send('Error updating password',err);
  }
}
```

## Get user data

```
exports.getUser = async(req,res) => {

  console.log("get user request params", req.params);
  const id = parseInt(req.params.primaryId);
  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("unable to find user data");
    }

    let value ={
      fullName : existingUser[0].full_name,
      dob : existingUser[0].date_of_birth,
      gender : existingUser[0].gender,
      occupation : existingUser[0].occupation,
      mobile : existingUser[0].mobile,
      addressLine1 : existingUser[0].address_line_1,
      addressLine2 : existingUser[0].address_line_2,
      district : existingUser[0].district,
      state : existingUser[0].state,
      country : existingUser[0].country,
      role : existingUser[0].role,
      status : existingUser[0].status,
      createdAt : existingUser[0].created_at,
    }

    console.log("user value", value);
    res.status(201).json(value);
  } catch(err) {
    console.log("err",err);
    res.status(500).send('Error while getting user');
```

```
}  
}
```

## React App

contexts/AppHeader.jsx

```
// src/components/AppHeader.js  
import { AppBar, Toolbar, Typography, Button } from '@mui/material';  
import { Link, useNavigate } from 'react-router-dom';  
import { useAuth } from '../AuthContext';  
  
function AppHeader() {  
  const { isAuthenticated, logout, user } = useAuth();  
  const navigate = useNavigate();  
  
  const handleLogout = () => {  
    logout();  
    navigate('/');  
  };  
  
  return (  
    <AppBar >  
      <Toolbar>  
        <Typography variant="h6" sx={{ flexGrow: 1 }}>  
          My App  
        </Typography>  
  
        {!isAuthenticated &&  
          <>  
            <Button color="inherit" component={Link} to="/">Home</Button>  
          </>  
        }  
      </Toolbar>  
    </AppBar>  
  );  
}
```

```

    <Button color="inherit" component={Link} to="/login">Login</Button>
    <Button color="inherit" component={Link} to="/register">Register</Button>
  </>

}

{isAuthenticated && (
  <>
    <Button color="inherit" component={Link}
to="/dashboard">Dashboard</Button>
    <Button color="inherit" onClick={handleLogout}>Logout</Button>
    <Typography sx={{ ml: 2 }}>{user?.data.emailAddress}</Typography>
  </>
)}
</Toolbar>
</AppBar>
);
}

export default AppHeader;

```

## contexts/AuthContext.jsx

```

// src/contexts/AuthContext.js
import { createContext, useContext, useEffect, useState, useRef } from 'react';
import { jwtDecode } from 'jwt-decode';
const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [authToken, setAuthToken] = useState(null);
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [showSessionWarning, setShowSessionWarning] = useState(false);

  const logoutTimerRef = useRef(null);
  const warningTimerRef = useRef(null);

```



```

useEffect(() => {
  const storedToken = localStorage.getItem('authToken');
  if (storedToken) {
    try {
      const decoded = jwtDecode(storedToken);
      if (decoded.exp * 1000 > Date.now()) {
        setAuthToken(storedToken);
        setUser(decoded);
        scheduleAutoLogout(decoded.exp);
      } else {
        console.log('Token expired or invalid. Removing from localStorage.');
```

```

        localStorage.removeItem('authToken');
      }
    } catch (err) {
      console.error('Failed to decode token or invalid token:', err);
      localStorage.removeItem('authToken'); // Invalid token
    }
  }
  setLoading(false);
}, []);

const login = (token) => {
  localStorage.setItem('authToken', token);
  const decoded = jwtDecode(token);
  setAuthToken(token);
  setUser(decoded);
  scheduleAutoLogout(decoded.exp);
};

const logout = () => {
  clearTimeout(logoutTimerRef.current);
  clearTimeout(warningTimerRef.current);
  localStorage.removeItem('authToken');
  setAuthToken(null);
  setUser(null);
  setShowSessionWarning(false);
};

const scheduleAutoLogout = (exp) => {

```

```

const expiryMs = exp * 1000 - Date.now();
const warningBeforeMs = 60 * 1000; // 1 minute before expiry

if (expiryMs <= 0) {
  logout();
} else {
  logoutTimerRef.current = setTimeout(logout, expiryMs);

  if (expiryMs > warningBeforeMs) {
    warningTimerRef.current = setTimeout(() => {
      setShowSessionWarning(true);
    }, expiryMs - warningBeforeMs);
  }
}
};

const isAuthenticated = !!authToken;

return (
  <AuthContext.Provider value={{ authToken, user, login, logout,
isAuthenticated, loading, showSessionWarning, setShowSessionWarning }}>
    {children}
  </AuthContext.Provider>
);
}

export const useAuth = () => useContext(AuthContext);

```

## component/SessionWarningDialog.jsx

```

// components/SessionWarningDialog.js
import { Dialog, DialogTitle, DialogContent, DialogActions, Button, Typography }
from '@mui/material';
import { useAuth } from '../contexts/AuthContext';

function SessionWarningDialog() {
  const { showSessionWarning, setShowSessionWarning, logout } = useAuth();

```

```

const handleLogoutNow = () => {
  setShowSessionWarning(false);
  logout();
};

const handleDismiss = () => {
  setShowSessionWarning(false);
  // Optional: trigger token refresh here
};

return (
  <Dialog open={showSessionWarning} onClose={handleDismiss}>
    <DialogTitle>Session Expiring</DialogTitle>
    <DialogContent>
      <Typography>
        Your session will expire in 1 minute. Please save your work or extend
your session.
      </Typography>
    </DialogContent>
    <DialogActions>
      <Button onClick={handleLogoutNow} color="error">Logout Now</Button>
      <Button onClick={handleDismiss} autoFocus>Continue</Button>
    </DialogActions>
  </Dialog>
);
}

export default SessionWarningDialog;

```

## contexts/PrivateRoute.jsx

```

// src/components/PrivateRoute.js
import { Navigate, Outlet } from 'react-router-dom';
import { useAuth } from '../AuthContext';

```

```

function PrivateRoute() {
  const { isAuthenticated, loading } = useAuth();

  // While authentication state is loading, render nothing or a loading spinner
  if (loading) {
    return (
      <div className="flex items-center justify-center min-h-screen bg-gray-100">
        <div className="animate-spin rounded-full h-12 w-12 border-b-2
border-gray-900"></div>
        <p className="ml-4 text-gray-700">Loading authentication...</p>
      </div>
    );
  }

  // If authenticated, render the child routes, otherwise redirect to login
  return isAuthenticated ? <Outlet /> : <Navigate to="/login" replace />;
}

export default PrivateRoute;

```

/contexts/RoleRoute.jsx

```

import { Navigate, Outlet } from 'react-router-dom';
import { useAuth } from './AuthContext';

function RoleRoute({ allowedRoles }) {
  const { isAuthenticated, user, loading } = useAuth();

  // While authentication state is loading, render nothing or a loading spinner
  if (loading) {
    return (
      <div className="flex items-center justify-center min-h-screen bg-gray-100">
        <div className="animate-spin rounded-full h-12 w-12 border-b-2
border-gray-900"></div>
        <p className="ml-4 text-gray-700">Checking user roles...</p>
      </div>
    );
  }

```

```

}

// If not authenticated, redirect to login
if (!isAuthenticated) {
  return <Navigate to="/login" replace />;
}

// If authenticated but user or user.role is missing, redirect to unauthorized
// This handles cases where the token might be valid but user data is
incomplete
if (!user || !user.data.role || !allowedRoles.includes(user.data.role)) {
  return <Navigate to="/unauthorized" replace />;
}

// If authenticated and has the allowed role, render the child routes
return <Outlet />;
}

export default RoleRoute;

```

## Pages/Enroll.jsx

```

import { Formik, Form, Field } from "formik";
import * as Yup from "yup";
import {
  Box,
  Button,
  TextField,
  Typography,
  Stack,
  Paper,
  FormControl,
  InputLabel,
  Select,

```

```

MenuItem,
} from "@mui/material";
import { useNavigate, useParams } from "react-router-dom";
import { useAuth } from "../contexts/AuthContext";
import axios from "axios";
import { useState, useEffect } from "react";

const styles = {
  container: {
    height: "100vh",
    width: "100vw",
    display: "flex",
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#f3f3f3",
  },
  paper: {
    padding: "30px",
    width: "350px",
    borderRadius: "8px",
    backgroundColor: "#fff",
    boxShadow: "0px 4px 12px rgba(0, 0, 0, 0.1)",
  },
  title: {
    textAlign: "center",
    marginBottom: "20px",
    fontWeight: 600,
  },
  button: {
    marginTop: "16px",
    padding: "10px",
    fontSize: "16px",
    backgroundColor: "#1976d2",
    color: "white",
  },
};

export default function Enroll() {
  const navigate = useNavigate();

```

```
const { id,name } = useParams();
console.log("eventId", id);
const { user } = useAuth();
console.log("user", user);
const [enrolldata, setEnroll] = useState({});
const [userdata, setUser] = useState({});

useEffect(() => {
  const fetchData = async () => {
    await getUser();
    await getEnrollData();
  };

  fetchData();
}, []);

const getUser = async () => {
  try {
    const res = await axios.get(
      `http://localhost:8085/ems/v1/auth/getuser/${user.data.userId}`
    );
    console.log("user data", res.data);
    setUser(res.data);
  } catch (err) {
    console.log("err in getting user information", err);
  }
};

const getEnrollData = async () => {
  try {
    const res = await axios.get(
      `http://localhost:8085/ems/v1/enroll/${id}/${user.data.emailAddress}`
    );
    console.log("res", res.data);
    setEnroll(res.data);
  } catch (err) {
    console.log("err in getting enrolled data", err);
  }
};
```

```

const validationSchema = Yup.object({
  status: Yup.number().required("Email is required"),
});

console.log("enrolleddata", enrolldata);
return (
  <Box style={styles.container}>
    <Paper elevation={3} style={styles.paper}>
      <Typography variant="h5" style={styles.title}>
        Enroll for {name}
      </Typography>
      <Formik
        initialValues={{
          id: enrolldata || "",
          eventId: id || userdata.eventId,
          fullName: enrolldata.fullName || userdata.fullName,
          emailAddress: enrolldata.emailAddress || user.data.emailAddress,
          mobile: enrolldata.mobile || userdata.mobile,
          country: enrolldata.country || userdata.country,
          state: enrolldata.state || userdata.state,
          district: enrolldata.district || userdata.district,
          status: enrolldata.status || 0,
          meta1: enrolldata.meta1 || "",
          meta2: enrolldata.meta2 || "",
          meta3: enrolldata.meta3 || "",
          createdBy: user.data.emailAddress,
          updatedBy: user.data.emailAddress,
        }}
        enableReinitialize={true}
        validationSchema={validationSchema}
        onSubmit={async (values) => {
          console.log("Submitted values:", values);
          try {
            if (
              enrolldata.id === "" ||
              enrolldata.id === undefined ||
              enrolldata.id === null
            ) {

```



```

        console.log("new enrollement");
        const result = await axios.post(
            "http://localhost:8085/ems/v1/enroll/add",
            values
        );
        console.log("result of enroll add", result);
    } else {
        console.log("update enrollement");
        const result = await axios.put(
            `http://localhost:8085/ems/v1/enroll/${enrolldata.id}`,
            values
        );
        console.log("result of enroll update", result);
    }
} catch (err) {
    console.log("error in updating enrollment", err);
}
}}
>
(({ handleChange, handleBlur, values, errors, touched }) => {
    console.log("Formik values.status:", values.status);
    return (
        <Form>
            <Stack spacing={3}>
                <FormControl fullWidth>
                    <InputLabel id="demo-simple-select-label">
                        Status
                    </InputLabel>
                    <Select
                        name="status"
                        value={values.status}
                        label="Status"
                        onChange={handleChange}
                        error={touched.status && Boolean(errors.status)}
                        helperText={touched.status && errors.status}
                    >
                        <MenuItem value={1}>enrolled</MenuItem>
                        <MenuItem value={0}>unenrolled</MenuItem>
                    </Select>

```

```
        </FormControl>

        <Button type="submit" fullWidth variant="contained">
          Update
        </Button>
      </Stack>
    </Form>
  );
}}
</Formik>
</Paper>
</Box>
);
}
```