WEEK-7 Full-Stack Web Application Development Record

AIM:

React, Node.js, Express.js, and SQL

Develop a full-stack web application using React for the front-end, Node.js and Express.js for the back-end, and a SQL database (MySQL or PostgreSQL) for data storage. Implement CRUD functionality and user authentication.

DESCRIPTION:

User authentication can be implemented using Node.js with Express.js, for which we need to install a few packages.

1. **Verify Node.js and npm installation:**

```
node -v
npm -v
```

1. **Create a backend folder:** Create a folder named `backend` and initialize Node.js:

```
npm init
```

This will generate a `package.json` file.

1. **Install required packages:**

```
npm i express nodemon cors
```

1. **Update package.json scripts:**

```
"scripts": {
    "start": "nodemon index.js"
}
```

1. **Start the server:**

```
npm start
```

PROGRAMS:

`index.js`

Sets up the Express server, handles JSON and URL-encoded data, and applies CORS for cross-origin access.

```
const express = require("express")
const app = express()
const port = 4000
const cors = require("cors")
const authRoutes = require("./routes/auth")
const authMiddleware = require("./middleware/authMiddleware")

app.use(express.json())
app.use(express.urlencoded({ extended: true }))
```

```
app.use(cors())

app.use("/api/auth", authRoutes)
app.use("/api/protected", authMiddleware, (req, res) => {
    res.json({ "message": `Hello user ${req.user.id}, you are authenticated
successfully` })
})

app.get("/", (req, res) => {
    res.send("API created successfully")
})

app.listen(port, () => console.log("server started"))
```

`routes/auth.js`

Handles registration and login.

```
const express = require("express")
const { addUser, findUser } = require("../models/users")
const { generateToken } = require("../config/jwt")
const bcrypt = require("bcryptjs")
const router = express.Router()

router.post("/register", async (req, res) => {
    const { username, password } = req.body

    if (findUser(username)) {
        return res.status(400).json({ "message": "User already exists" })
    }

    const newUser = await addUser(username, password)
    const token = generateToken(newUser)
    res.status(201).json({ "message": newUser, token })
})

router.post('/login', async (req, res) => {
    const { username, password } = req.body
    const user = findUser(username)

    if (!user) {
        return res.status(404).json({ "message": "User not found! Create
account" })
    }

    const isMatch = await bcrypt.compare(password, user.password)
    if (!isMatch) {
        return res.status(400).json({ "message": "Incorrect password" })
    }

    const token = generateToken(user)
    res.status(200).json({ "message": "Logged in successfully", token })
})

module.exports = router
```

`models/user.js`

Handles in-memory user storage and password hashing.

```
const bcrypt = require("bcryptjs")
let users = []

const addUser = async (username, password) => {
    const hashedPassword = await bcrypt.hash(password, 10)
    const newUser = { id: users.length + 1, username, password: hashedPassword
}
    users.push(newUser)
    return newUser
}

const findUser = (username) => {
    return users.find(user => user.username === username)
}

module.exports = { addUser, findUser }
```

`config/jwt.js`

Handles JWT creation and validation.

```
const jwt = require("jsonwebtoken")
require("dotenv").config()

const generateToken = (user) => {
    return jwt.sign({ id: user.id }, process.env.JWT_SECRET, { expiresIn: "1h"
})
}

const validateToken = (token) => {
    return jwt.verify(token, process.env.JWT_SECRET)
}

module.exports = { generateToken, validateToken }
```

`middleware/authMiddleware.js`

Validates the token and authorizes users.

```
const { validateToken } = require("../config/jwt")

const authMiddleware = (req, res, next) => {
    const token = req.header("Authorization")

    if (!token) {
        return res.status(401).json({ "message": "Access denied, login first!"
})
    }

    try {
        const verified = validateToken(token.replace("Bearer ", ""))
        req.user = verified
        next()
    } catch (err) {
        res.status(403).json({ "message": "Unauthorized" })
    }
}

module.exports = authMiddleware
```

RESULT:

The project ensures security with password hashing and token-based authentication. It features a modular structure for easy maintenance and supports registration, login, and protected routes with scalable database integration.