11/11/2024

APDS7311

POE FINAL

All code included in submission.

Final POE requirements have been met as per rubric.

Contents

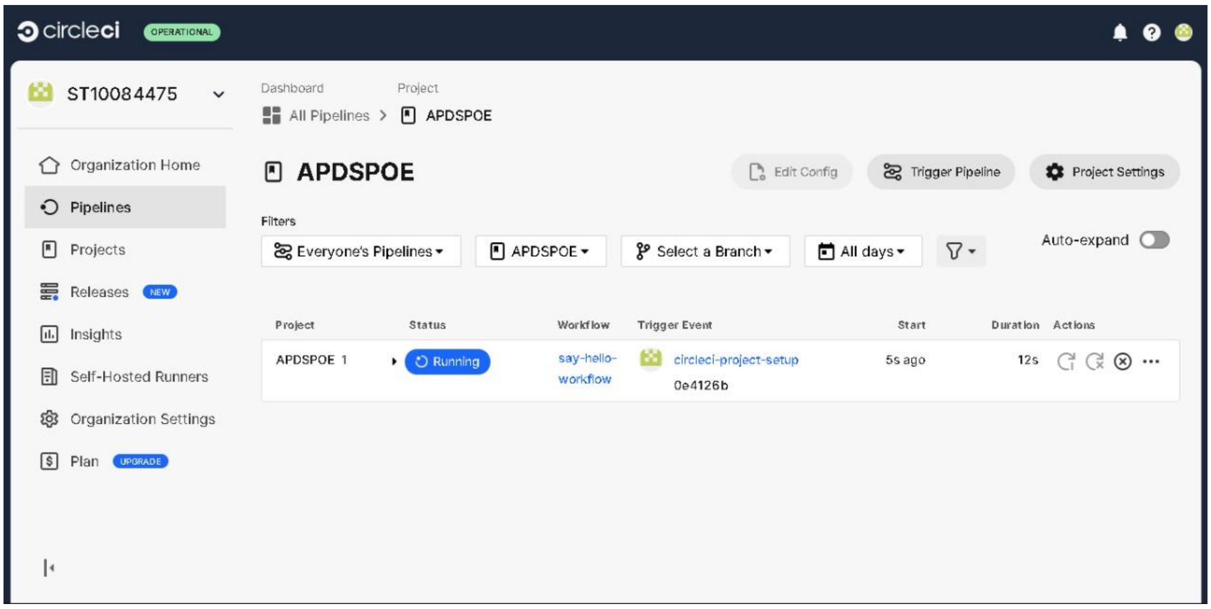
Demo Video Link (YouTube): https://youtu.be/0OSEy9PzX1w?si=XzMR6ej-BS0lAOuN

Pipeline: https://github.com/ST10084475/APDSPOE

GitHub Submission Link: https://github.com/VCWVL/apds7311-poe-Hkad786.git

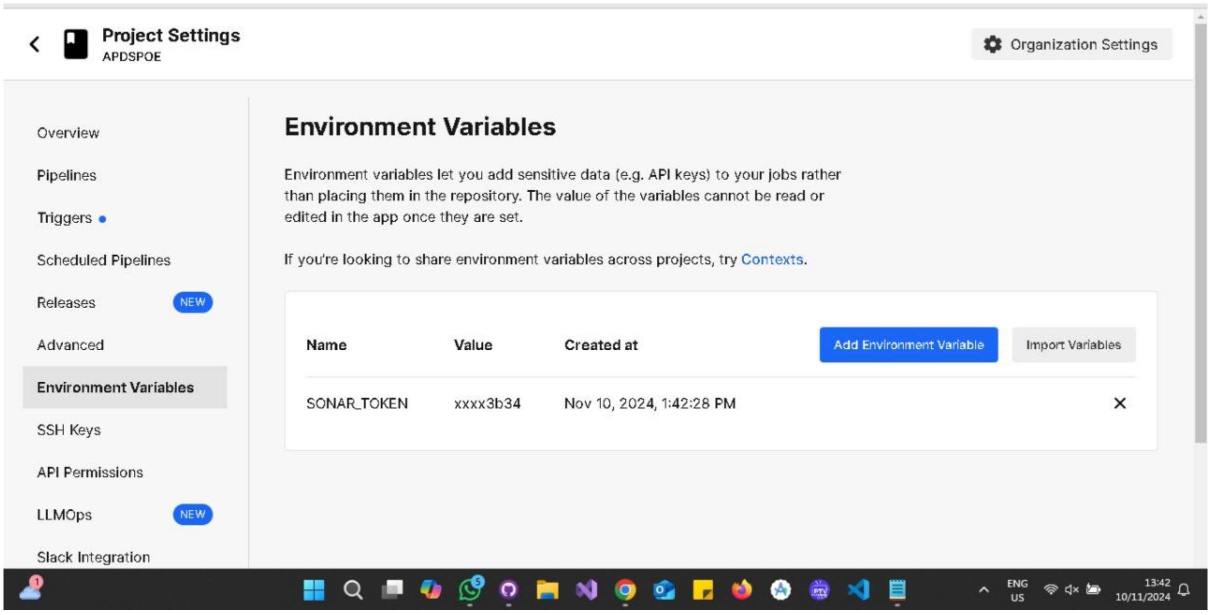
DevSecOps Pipeline

DevSecOps Pipeline	• No or limited static login information is applied	• Accounts are preconfigured and functional; no registration process is possible.	• The provided software shows additional research to provide an exceptional implementation
[30 Marks]	**0 – 9 Marks**	**10 – 20 Marks**	**20-30 Marks**



Project	Status	Workflow	Trigger Event	Start	Duration	Actions
APDSPOE 1	Running	say-hello-workflow	circleci-project-setup 0e4126b	5s ago	12s	

PipeLine created on circle ci



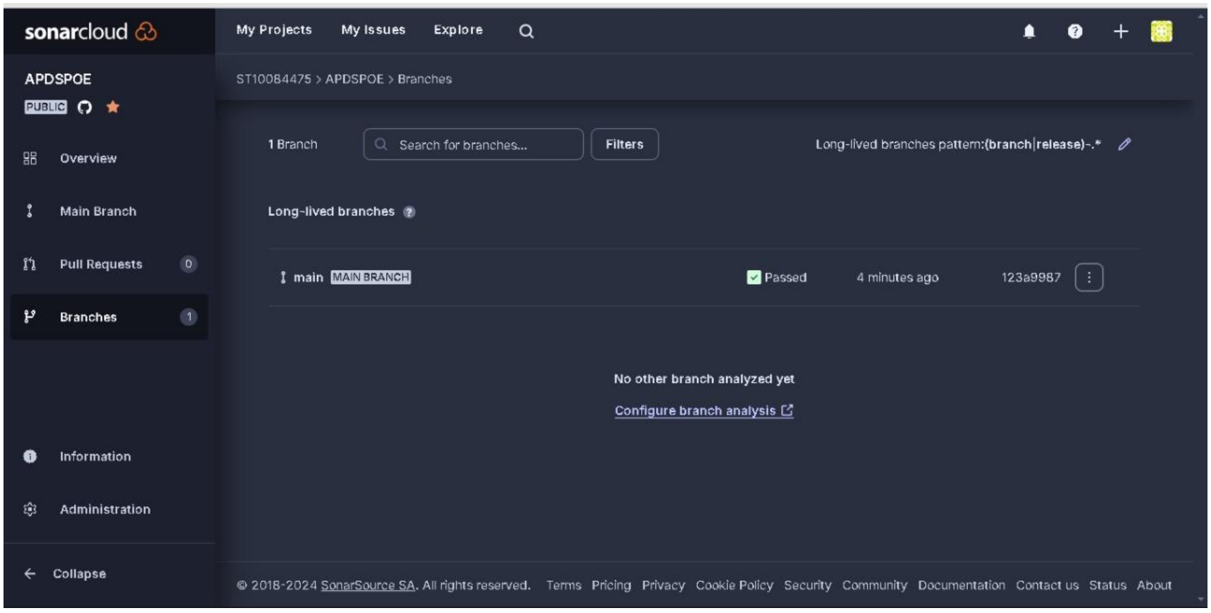
Environment Variables

Environment variables let you add sensitive data (e.g. API keys) to your jobs rather than placing them in the repository. The value of the variables cannot be read or edited in the app once they are set.

If you're looking to share environment variables across projects, try Contexts.

Name	Value	Created at
SONAR_TOKEN	xxxx3b34	Nov 10, 2024, 1:42:28 PM

Environment variables added for SonarCloud



ST10084475 > APDSPOE > Branches

1 Branch | Search for branches... | Filters | Long-lived branches pattern:(branch|release)-.*

Long-lived branches

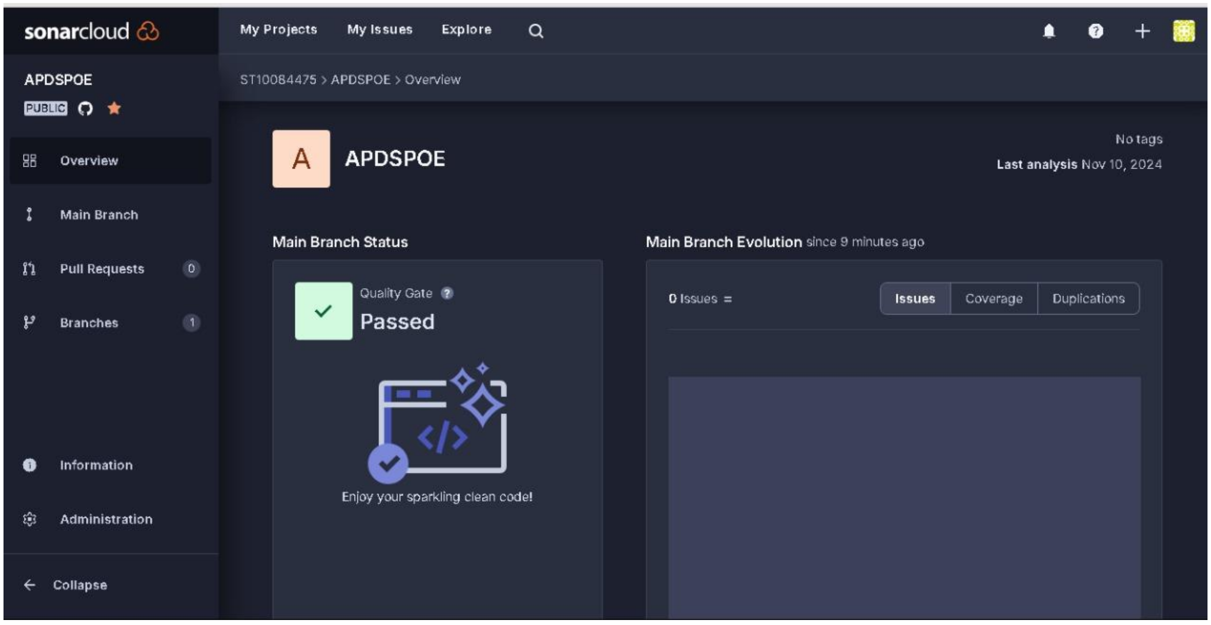
main MAIN BRANCH ✓ Passed 4 minutes ago 123a9987

No other branch analyzed yet

Configure branch analysis

© 2018-2024 SonarSource SA. All rights reserved. Terms Pricing Privacy Cookie Policy Security Community Documentation Contact us Status About

SonarCloud Scan run



ST10084475 > APDSPOE > Overview

A APDSPOE

No tags

Last analysis Nov 10, 2024

Main Branch Status

Quality Gate

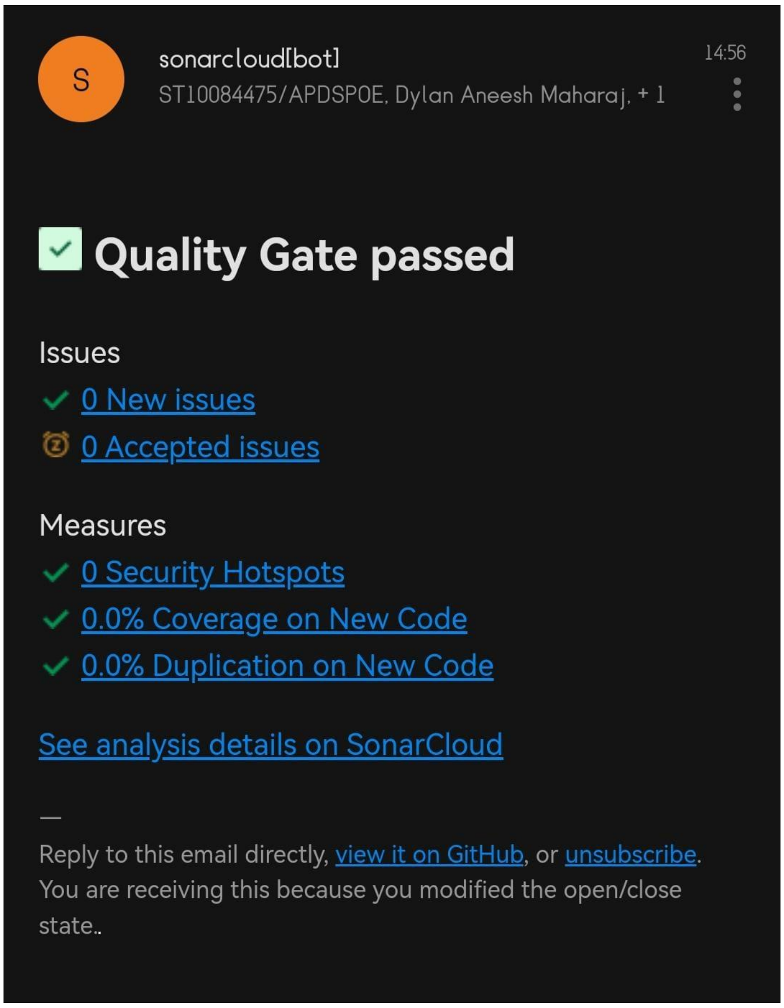
Passed

Enjoy your sparkling clean code!

Main Branch Evolution since 9 minutes ago

0 Issues

Scan results


sonarcloud[bot]
ST10084475/APDSPOE, Dylan Aneesh Maharaj, + 1
14:56

✓ Quality Gate passed

Issues
✓ 0 New issues
0 Accepted issues

Measures
✓ 0 Security Hotspots
✓ 0.0% Coverage on New Code
✓ 0.0% Duplication on New Code

See analysis details on SonarCloud

—
Reply to this email directly, view it on GitHub, or unsubscribe.
You are receiving this because you modified the open/close state..

Evidence

Project code meeting rubric requirements.

Password Security **[20 Marks]**	• Lack of general security needed for both portals	• Basic security is applied to both portals.	• The provided software shows additional research to provide an exceptional implementation
	0 – 9 Marks	**10 – 14 Marks**	**15 – 20 Marks**

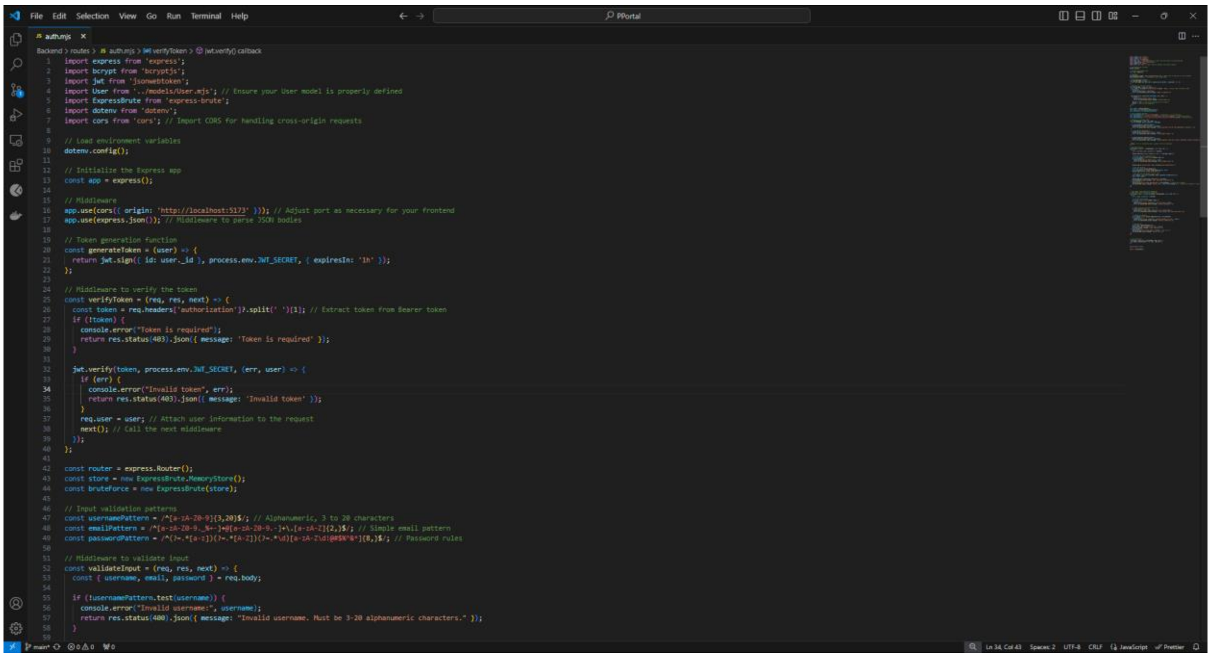
Password Security

Customer Portal

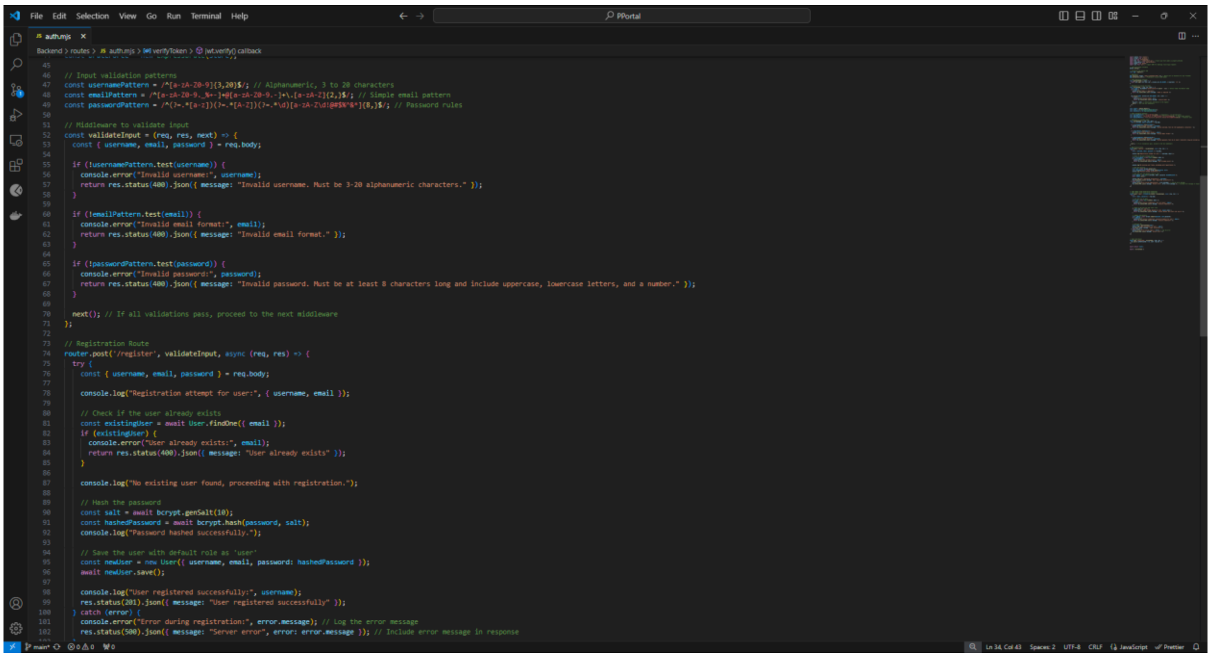
1. Password Strength Validation:
- Implemented in the validateInput middleware.
- Enforces strong password rules: at least 8 characters, one uppercase letter, one lowercase letter, one number.
2. Secure Storage:
- Passwords are hashed using bcrypt with a salt before being stored in the database.
3. Brute Force Protection:
- Implemented express-brute middleware to limit login attempts and prevent brute force attacks.
4. Secure Authentication:
- Passwords are securely verified during login using bcrypt.
- JWT tokens are signed with a secret and expire in 1 hour to reduce risk.
5. Error Handling:
- Error messages are generic to prevent user enumeration attacks.

Code Example:

The main implementation file auth.mjs:



```javascript
import express from 'express';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import User from '../models/User.mjs'; // Ensure your User model is properly defined
import ExpressBrute from 'express-brute';
import dotenv from 'dotenv';
import cors from 'cors'; // Import CORS for handling cross-origin requests

// Load environment variables
dotenv.config();

// Initialize the Express app
const app = express();

// Middleware
app.use(cors({ origin: 'http://localhost:5173' })); // Adjust port as necessary for your frontend
app.use(express.json()); // Middleware to parse JSON bodies

// Token generation function
const generateToken = (user) => {
  return jwt.sign({ id: user._id }, process.env.JWT_SECRET, { expiresIn: '1h' });
};

// Middleware to verify the token
const verifyToken = (req, res, next) => {
  const token = req.headers['authorization']?.split(' ')[1]; // Extract token from Bearer token
  if (!token) {
    console.error("Token is required");
    return res.status(403).json({ message: 'Token is required' });
  }

  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) {
      console.error("Invalid token", err);
      return res.status(403).json({ message: 'Invalid token' });
    }
    req.user = user; // Attach user information to the request
    next(); // Call the next middleware
  });
};

const router = express.Router();
const store = new ExpressBrute.MemoryStore();
const bruteForce = new ExpressBrute(store);

// Input validation patterns
const usernamePattern = /^[a-zA-Z0-9]{3,20}$/; // Alphanumeric, 3 to 20 characters
const emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/; // Simple email pattern
const passwordPattern = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d!@#$%^&*]{8,}$/; // Password rules

// Middleware to validate input
const validateInput = (req, res, next) => {
  const { username, email, password } = req.body;

  if (!usernamePattern.test(username)) {
    console.error("Invalid username:", username);
    return res.status(400).json({ message: "Invalid username. Must be 3-20 alphanumeric characters." });
  }

```



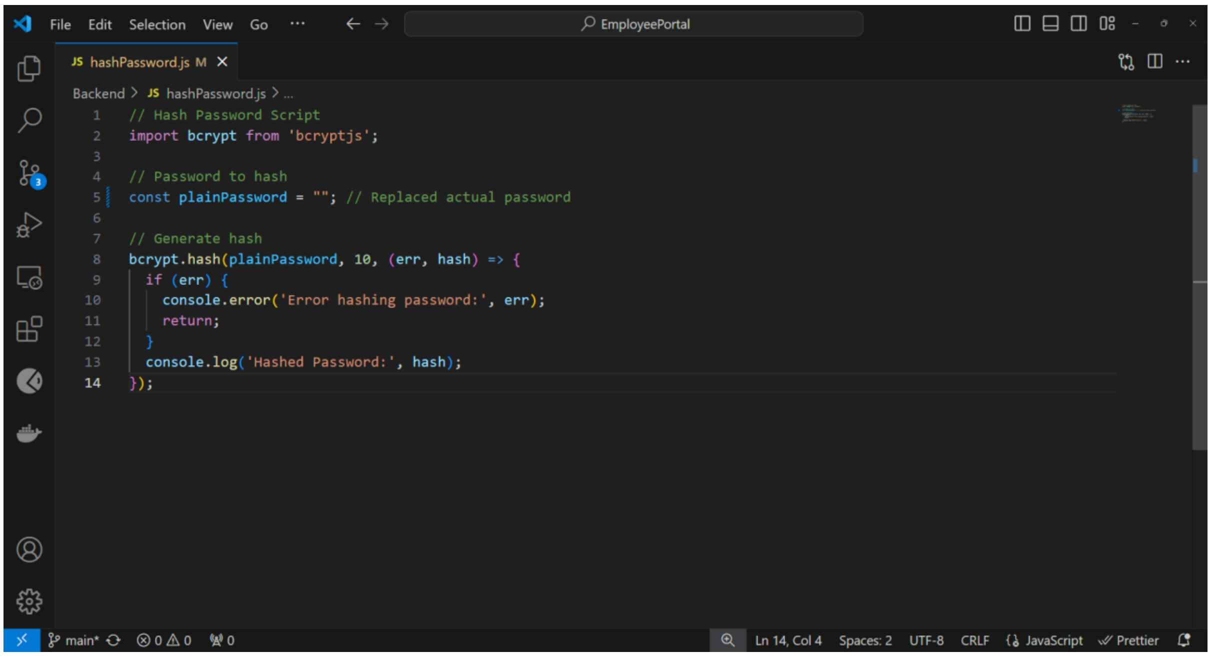
```javascript

// Input validation patterns
const usernamePattern = /^[a-zA-Z0-9]{3,20}$/; // Alphanumeric, 3 to 20 characters
const emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/; // Simple email pattern
const passwordPattern = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d!@#$%^&*]{8,}$/; // Password rules

// Middleware to validate input
const validateInput = (req, res, next) => {
  const { username, email, password } = req.body;

  if (!usernamePattern.test(username)) {
    console.error("Invalid username:", username);
    return res.status(400).json({ message: "Invalid username. Must be 3-20 alphanumeric characters." });
  }

  if (!emailPattern.test(email)) {
    console.error("Invalid email format:", email);
    return res.status(400).json({ message: "Invalid email format." });
  }

  if (!passwordPattern.test(password)) {
    console.error("Invalid password:", password);
    return res.status(400).json({ message: "Invalid password. Must be at least 8 characters long and include uppercase, lowercase letters, and a number." });
  }

  next(); // If all validations pass, proceed to the next middleware
};

// Registration Route
router.post('/register', validateInput, async (req, res) => {
  try {
    const { username, email, password } = req.body;

    console.log("Registration attempt for user:", { username, email });

    // Check if the user already exists
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      console.error("User already exists:", email);
      return res.status(400).json({ message: "User already exists" });
    }

    console.log("No existing user found, proceeding with registration.");

    // Hash the password
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);
    console.log("Password hashed successfully.");

    // Save the user with default role as 'user'
    const newUser = new User({ username, email, password: hashedPassword });
    await newUser.save();

    console.log("User registered successfully:", username);
    res.status(201).json({ message: "User registered successfully" });
  } catch (error) {
    console.error("Error during registration:", error.message); // Log the error message
    res.status(500).json({ message: "Server error", error: error.message }); // Include error message in response
```

Employee Portal

1. **Password Strength Validation:**

- The Employee Portal does not have a registration feature. Employee accounts were pre-configured and added directly to the database.
- Passwords were hashed using the hashPassword.js script, which utilizes bcrypt with a salt of 10 to ensure secure password storage.

Evidence: The following script was used for hashing passwords: Backend\hashPassword.js:


File Edit Selection View Go ···
EmployeePortal
hashPassword.js M
Backend > hashPassword.js > ...
main* 0 0 0
Ln 14, Col 4 Spaces: 2 UTF-8 CRLF JavaScript Prettier

```javascript
// Hash Password Script
import bcrypt from 'bcryptjs';

// Password to hash
const plainPassword = ""; // Replaced actual password

// Generate hash
bcrypt.hash(plainPassword, 10, (err, hash) => {
  if (err) {
    console.error('Error hashing password:', err);
    return;
  }
  console.log('Hashed Password:', hash);
});
```

2. Secure Storage:
- Passwords are stored in hashed format using bcrypt with salting to enhance security.
3. Brute Force Protection:
- A rate limiter restricts login attempts to 5 per 15 minutes to prevent brute force attacks.
- Evidence: auth.mjs implements this via the loginLimiter middleware.

Transition from express-brute to express-rate-limit:

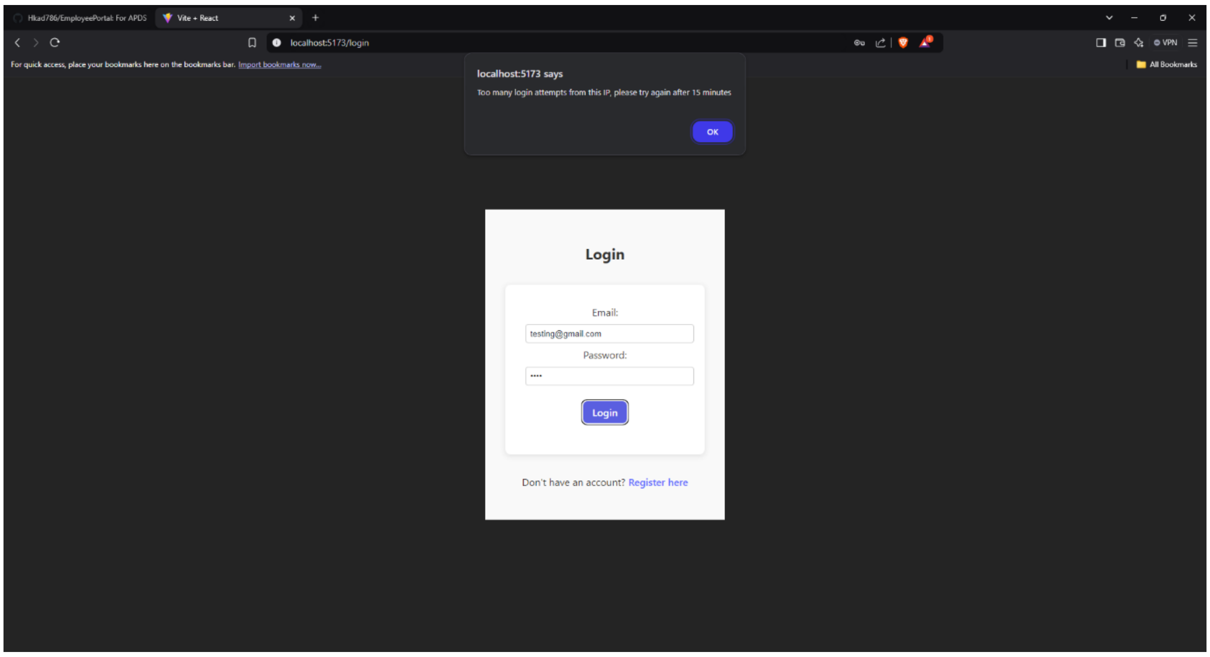
- Initial Approach: The portal initially used express-brute for brute force protection.
- Limitations Identified:
  - After receiving GitHub Dependabot warnings about outdated dependencies and potential security vulnerabilities in express-brute, I reviewed its maintenance status and security risks.
  - Research revealed that express-brute has limited active development and lacks modern security features (e.g., robust customization for attack patterns).
- Research and Decision:
  - Explored community discussions on GitHub, Stack Overflow, and security blogs, which recommended express-rate-limit as an actively maintained and widely used alternative.
- Chose express-rate-limit because it offers:
  - Simple configuration and robust performance.
  - Active development and regular updates.

- More flexibility to adapt to modern attack patterns (e.g., IP-based or route-specific limits).

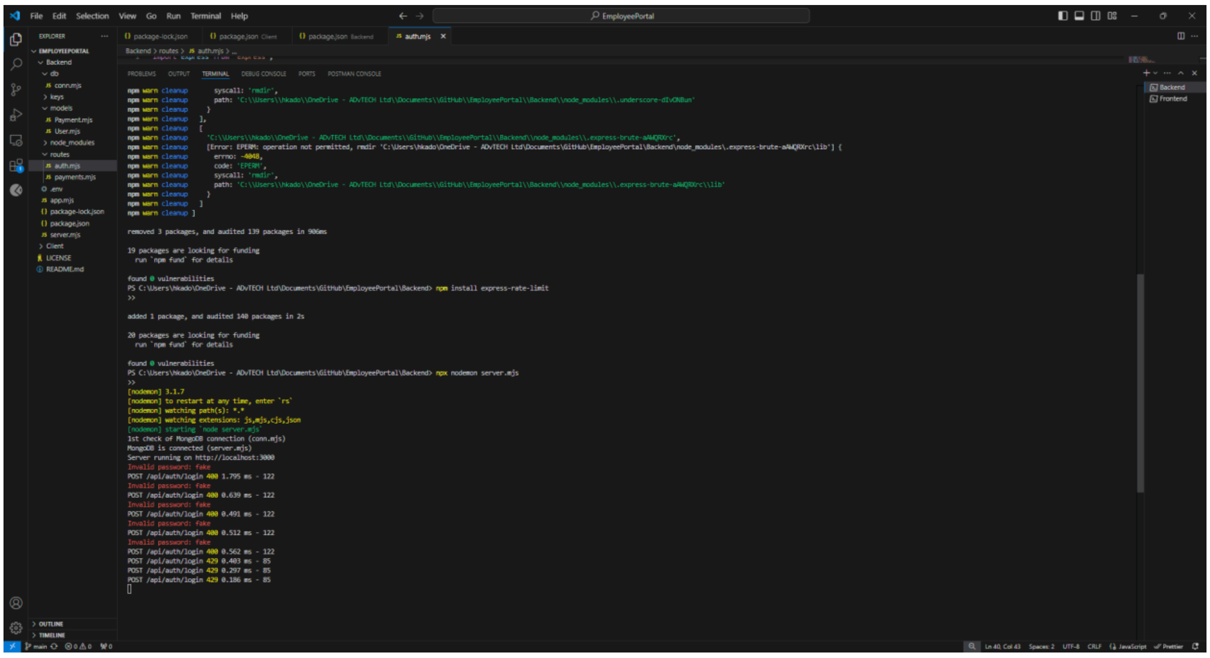
- Implementation:

Replaced express-brute with express-rate-limit in the authentication route to address these concerns.

Pics Showing implementation:
(Using an incorrect account multiple times locks it out)


localhost:5173 says
Too many login attempts from this IP, please try again after 15 minutes

(backend View):


```
npm warn cleanup      syscall: 'rmdir',
npm warn cleanup      path: 'C:\\Users\\hkado\\OneDrive - ADvTECH Ltd\\Documents\\GitHub\\EmployeePortal\\Backend\\node_modules\\.underscore-dIvONBun'
npm warn cleanup    }
npm warn cleanup  ],
npm warn cleanup  [
npm warn cleanup    'C:\\Users\\hkado\\OneDrive - ADvTECH Ltd\\Documents\\GitHub\\EmployeePortal\\Backend\\node_modules\\.express-brute-aAWQRXrc',
npm warn cleanup    [Error: EPERM: operation not permitted, rmdir 'C:\Users\hkado\OneDrive - ADvTECH Ltd\Documents\GitHub\EmployeePortal\Backend\node_modules\.express-brute-aAWQRXrc\lib'] {
npm warn cleanup      errno: -4048,
npm warn cleanup      code: 'EPERM',
npm warn cleanup      syscall: 'rmdir',
npm warn cleanup      path: 'C:\\Users\\hkado\\OneDrive - ADvTECH Ltd\\Documents\\GitHub\\EmployeePortal\\Backend\\node_modules\\.express-brute-aAWQRXrc\\lib'
npm warn cleanup    }
npm warn cleanup  ]
npm warn cleanup ]

removed 3 packages, and audited 139 packages in 906ms

19 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\hkado\OneDrive - ADvTECH Ltd\Documents\GitHub\EmployeePortal\Backend> npm install express-rate-limit
>>

added 1 package, and audited 140 packages in 2s

20 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\hkado\OneDrive - ADvTECH Ltd\Documents\GitHub\EmployeePortal\Backend> npx nodemon server.mjs
>>
[nodemon] 3.1.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.mjs`
1st check of MongoDB connection (conn.mjs)
MongoDB is connected (server.mjs)
Server running on http://localhost:3000
Invalid password: fake
POST /api/auth/login 400 1.795 ms - 122
Invalid password: fake
POST /api/auth/login 400 0.639 ms - 122
Invalid password: fake
POST /api/auth/login 400 0.491 ms - 122
Invalid password: fake
POST /api/auth/login 400 0.512 ms - 122
Invalid password: fake
POST /api/auth/login 400 0.562 ms - 122
POST /api/auth/login 429 0.403 ms - 85
POST /api/auth/login 429 0.297 ms - 85
POST /api/auth/login 429 0.186 ms - 85
```

4. Secure Authentication:

- Passwords are securely verified using bcrypt.compare during login.
- JWT tokens are signed with a secret key and expire in 1 hour, reducing exposure risk in case of token theft.

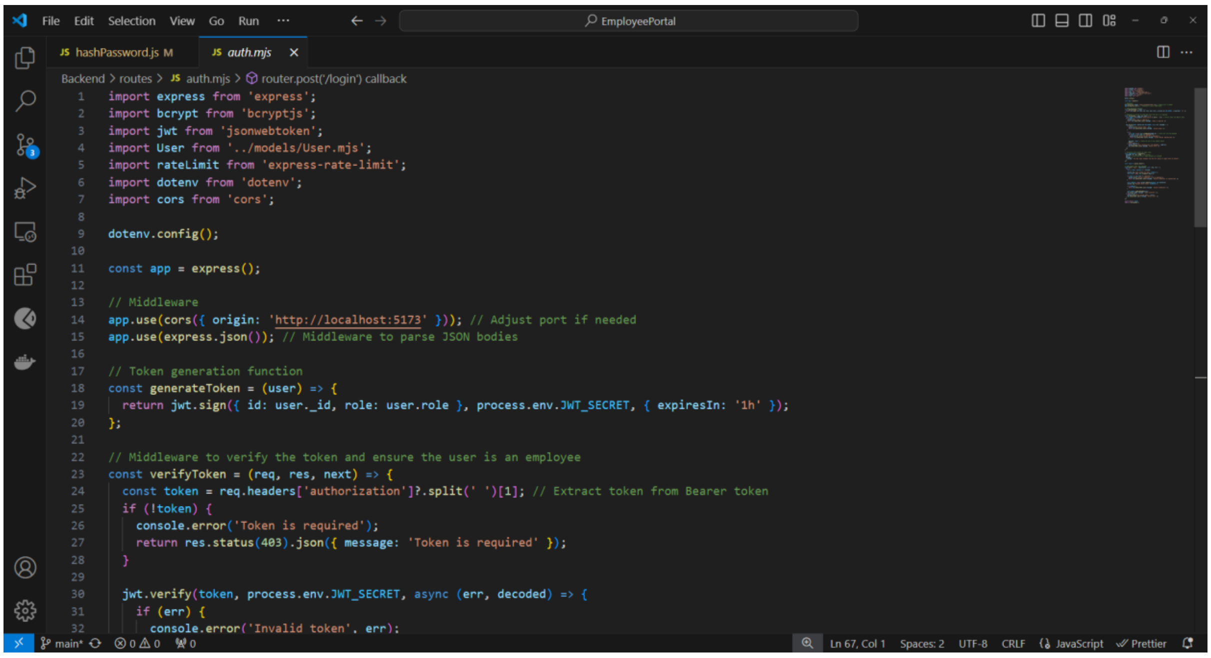
5. Error Handling:

- Generic error messages prevent user enumeration attacks.
- Example: "Invalid credentials" is displayed regardless of whether the email or password is incorrect.

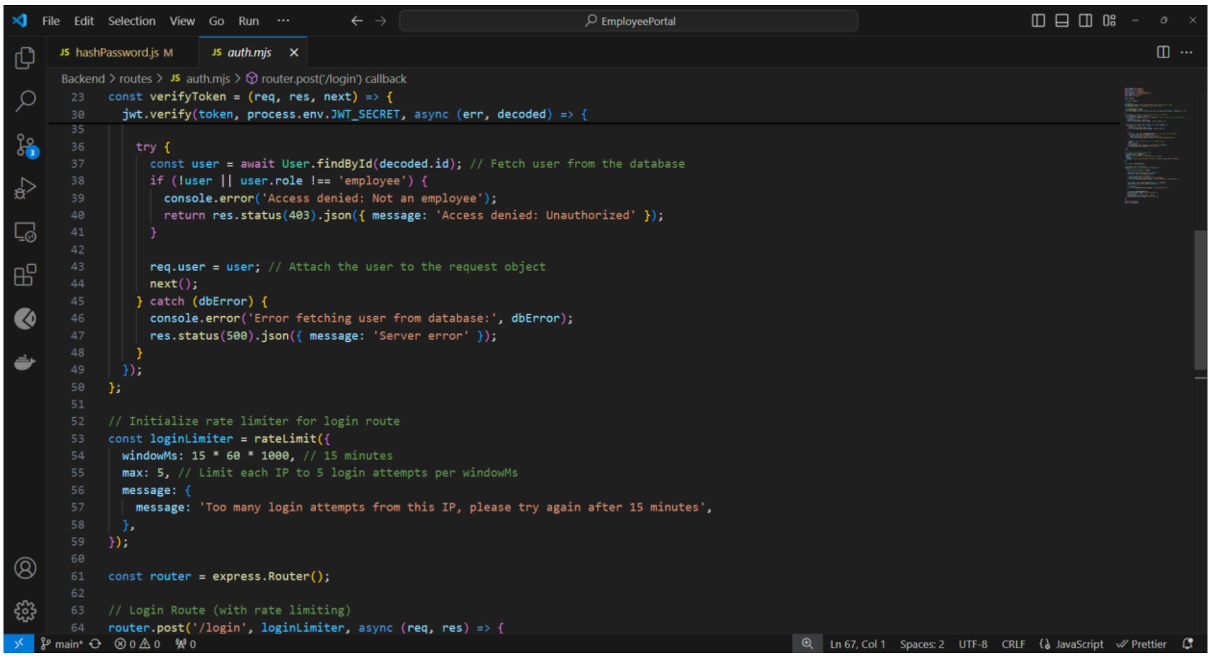
6. Additional Measures:

- Session Hijacking Protection: HTTPS will be used during deployment to encrypt all communication between client and server.
- Clickjacking Protection: Implemented via helmet middleware, which sets the X-Frame-Options header.
- Cross-Site Scripting (XSS) Prevention: Input validation sanitizes user inputs to prevent script injection.
- Man-in-the-Middle Attack Prevention: HTTPS ensures secure communication during data transmission.

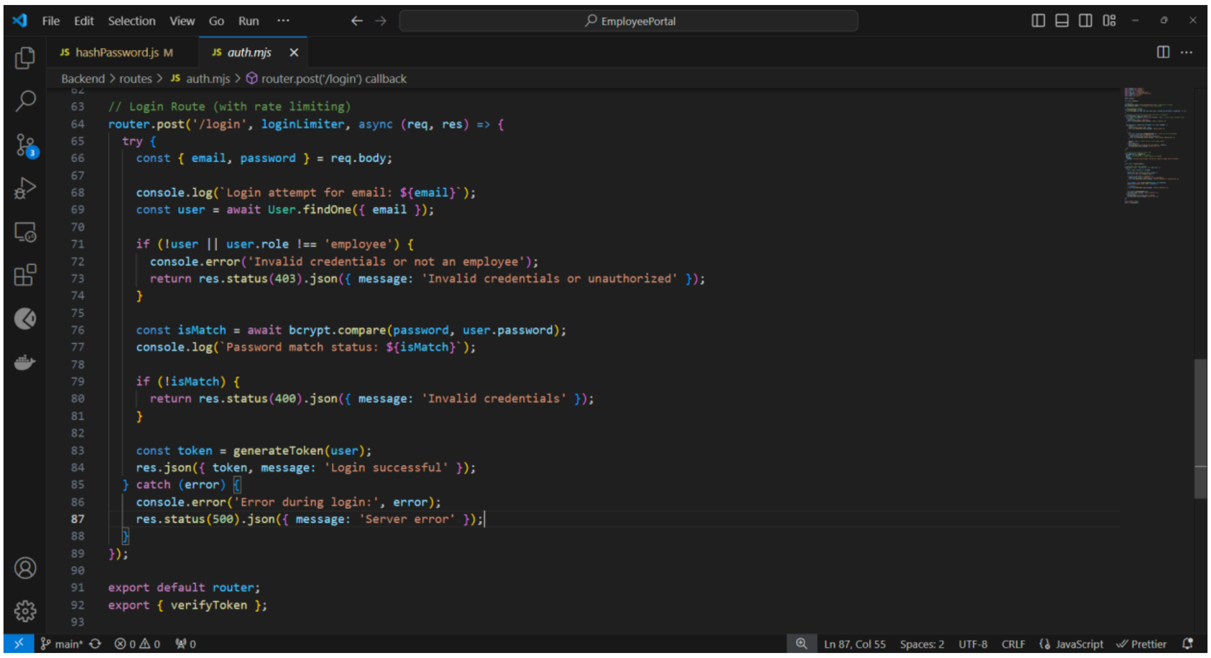
Main Code:


File Edit Selection View Go Run ···
EmployeePortal
hashPassword.js M
auth.mjs
Backend > routes > auth.mjs > router.post('/login') callback
main* ⊗ 0 ⚠ 0 0
Ln 67, Col 1 Spaces: 2 UTF-8 CRLF JavaScript Prettier

```javascript
import express from 'express';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import User from '../models/User.mjs';
import rateLimit from 'express-rate-limit';
import dotenv from 'dotenv';
import cors from 'cors';

dotenv.config();

const app = express();

// Middleware
app.use(cors({ origin: 'http://localhost:5173' })); // Adjust port if needed
app.use(express.json()); // Middleware to parse JSON bodies

// Token generation function
const generateToken = (user) => {
  return jwt.sign({ id: user._id, role: user.role }, process.env.JWT_SECRET, { expiresIn: '1h' });
};

// Middleware to verify the token and ensure the user is an employee
const verifyToken = (req, res, next) => {
  const token = req.headers['authorization']?.split(' ')[1]; // Extract token from Bearer token
  if (!token) {
    console.error('Token is required');
    return res.status(403).json({ message: 'Token is required' });
  }

  jwt.verify(token, process.env.JWT_SECRET, async (err, decoded) => {
    if (err) {
      console.error('Invalid token', err);
```



```javascript
const verifyToken = (req, res, next) => {
  jwt.verify(token, process.env.JWT_SECRET, async (err, decoded) => {

    try {
      const user = await User.findById(decoded.id); // Fetch user from the database
      if (!user || user.role !== 'employee') {
        console.error('Access denied: Not an employee');
        return res.status(403).json({ message: 'Access denied: Unauthorized' });
      }

      req.user = user; // Attach the user to the request object
      next();
    } catch (dbError) {
      console.error('Error fetching user from database:', dbError);
      res.status(500).json({ message: 'Server error' });
    }
  });
};

// Initialize rate limiter for login route
const loginLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 5, // Limit each IP to 5 login attempts per windowMs
  message: {
    message: 'Too many login attempts from this IP, please try again after 15 minutes',
  },
});

const router = express.Router();

// Login Route (with rate limiting)
router.post('/login', loginLimiter, async (req, res) => {
```



```javascript
// Login Route (with rate limiting)
router.post('/login', loginLimiter, async (req, res) => {
  try {
    const { email, password } = req.body;

    console.log(`Login attempt for email: ${email}`);
    const user = await User.findOne({ email });

    if (!user || user.role !== 'employee') {
      console.error('Invalid credentials or not an employee');
      return res.status(403).json({ message: 'Invalid credentials or unauthorized' });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    console.log(`Password match status: ${isMatch}`);

    if (!isMatch) {
      return res.status(400).json({ message: 'Invalid credentials' });
    }

    const token = generateToken(user);
    res.json({ token, message: 'Login successful' });
  } catch (error) {
    console.error('Error during login:', error);
    res.status(500).json({ message: 'Server error' });
  }
});

export default router;
export { verifyToken };
```

Reference links:

OWASP Password Storage Cheat Sheet:

https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Express Rate Limit GitHub:

https://github.com/nfriedly/express-rate-limit

bcrypt GitHub Repository:

https://github.com/kelektiv/node.bcrypt.js/

Static login [10 Marks]	• No or limited static login information is applied	• Accounts are preconfigured and functional; no registration process is possible.	• The provided software shows additional research to provide an exceptional implementation
	0 – 4 Marks	**5 – 7 Marks**	**8 – 10 Marks**

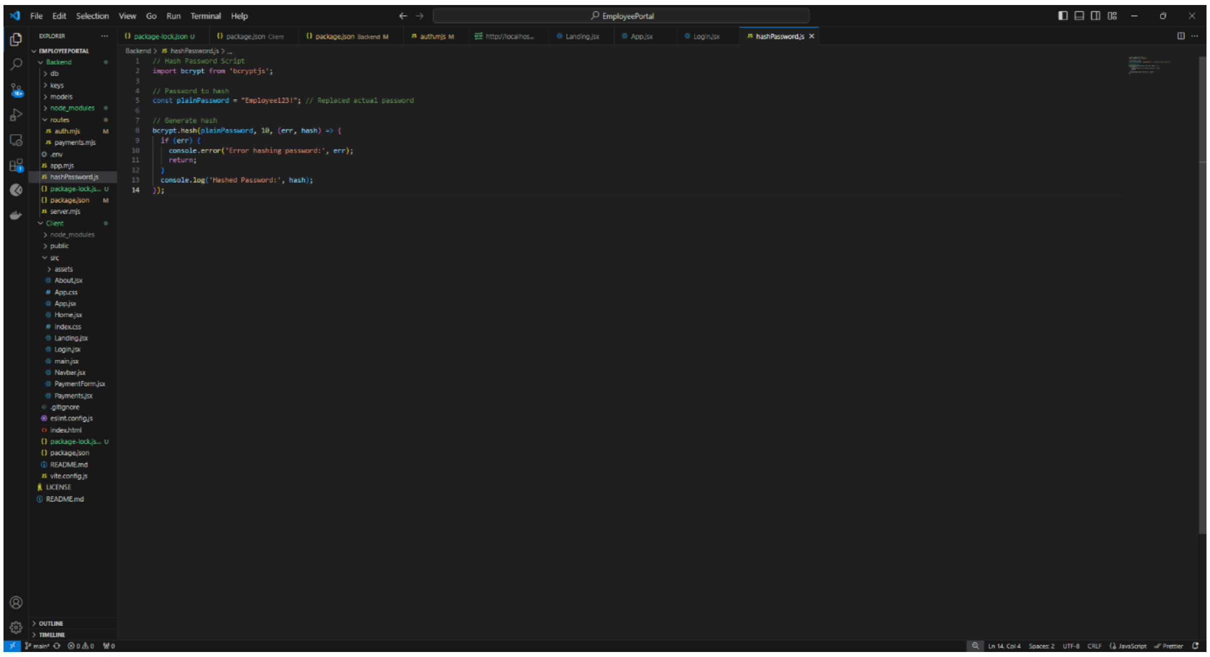
Static Login

Employee Accounts

1. Preconfigured Accounts:

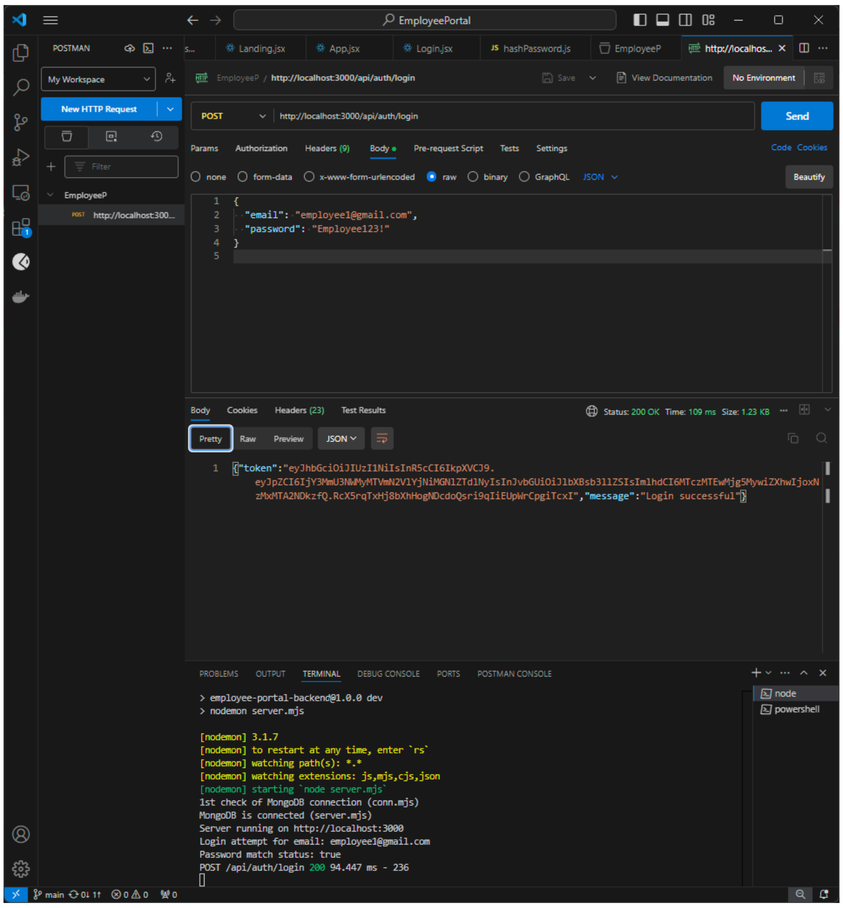
- Employee accounts are preconfigured directly in MongoDB with no registration feature, ensuring static login functionality.
- Passwords are securely hashed using the hashPassword.js script, leveraging bcrypt with salting for strong password storage.

Pics showing the setup:



Backend > hashPassword.js > ...

```javascript
// Hash Password Script
import bcrypt from 'bcryptjs';

// Password to hash
const plainPassword = "Employee123!"; // Replaced actual password

// Generate hash
bcrypt.hash(plainPassword, 10, (err, hash) => {
  if (err) {
    console.error('Error hashing password:', err);
    return;
  }
  console.log('Hashed Password:', hash);
});
```



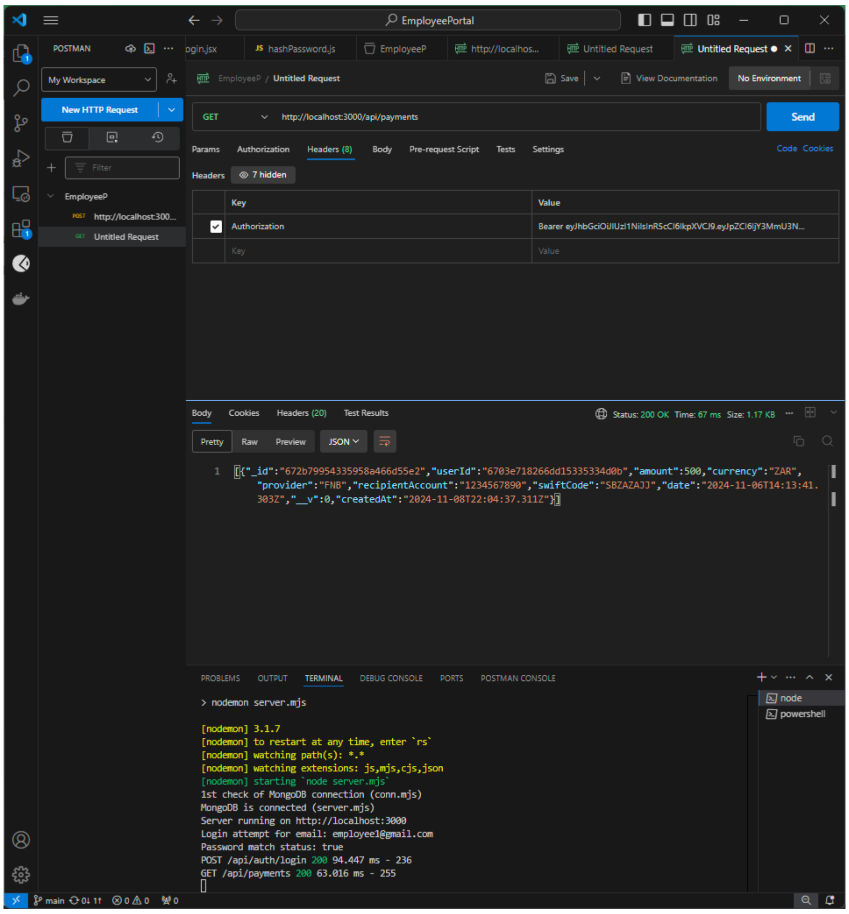
POST http://localhost:3000/api/auth/login

```
{
  "email": "employee1@gmail.com",
  "password": "Employee123!"
}
```

Status: 200 OK Time: 109 ms Size: 1.23 KB

```
{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpZCI6IjY3MmU3NWMyMTVmN2V1YjNiMGN1ZTd1NyIsInJvbGUiOiJ1bXBsb311ZSIsImlhdCI6MTczMTEwMjg5MywiZXhwIjoxN
zMxMTA2NDkzfQ.RcX5rqTxHj8bXhHogNDcdoQsri9qIiEUpWrCpgiTcxI","message":"Login successful"}
```

```
> employee-portal-backend@1.0.0 dev
> nodemon server.mjs

[nodemon] 3.1.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.mjs`
1st check of MongoDB connection (conn.mjs)
MongoDB is connected (server.mjs)
Server running on http://localhost:3000
Login attempt for email: employee1@gmail.com
Password match status: true
POST /api/auth/login 200 94.447 ms - 236
```



GET http://localhost:3000/api/payments

Key	Value
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MmU3N...

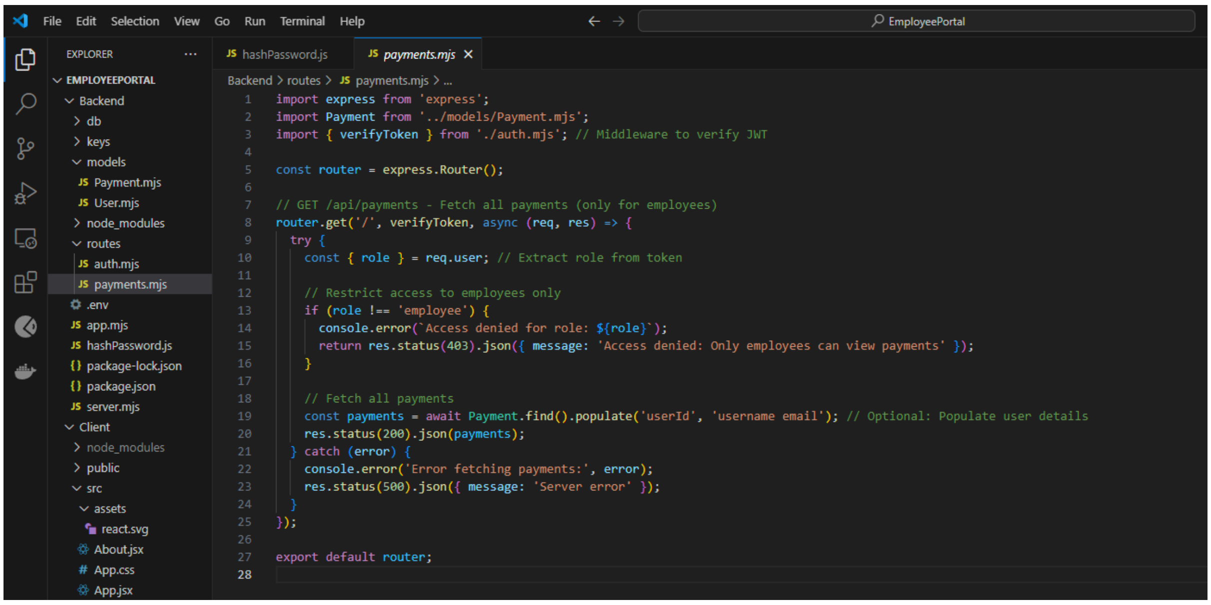
Status: 200 OK Time: 67 ms Size: 1.17 KB

```
[{"_id":"672b79954335958a466d55e2","userId":"6703e718266dd15335334d0b","amount":500,"currency":"ZAR",
"provider":"FNB","recipientAccount":"1234567890","swiftCode":"SBZAZAJJ","date":"2024-11-06T14:13:41.
303Z","__v":0,"createdAt":"2024-11-08T22:04:37.311Z"}]
```

```
> nodemon server.mjs

[nodemon] 3.1.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.mjs`
1st check of MongoDB connection (conn.mjs)
MongoDB is connected (server.mjs)
Server running on http://localhost:3000
Login attempt for email: employee1@gmail.com
Password match status: true
POST /api/auth/login 200 94.447 ms - 236
GET /api/payments 200 63.016 ms - 255
```

2. Role-Based Access Control (RBAC):

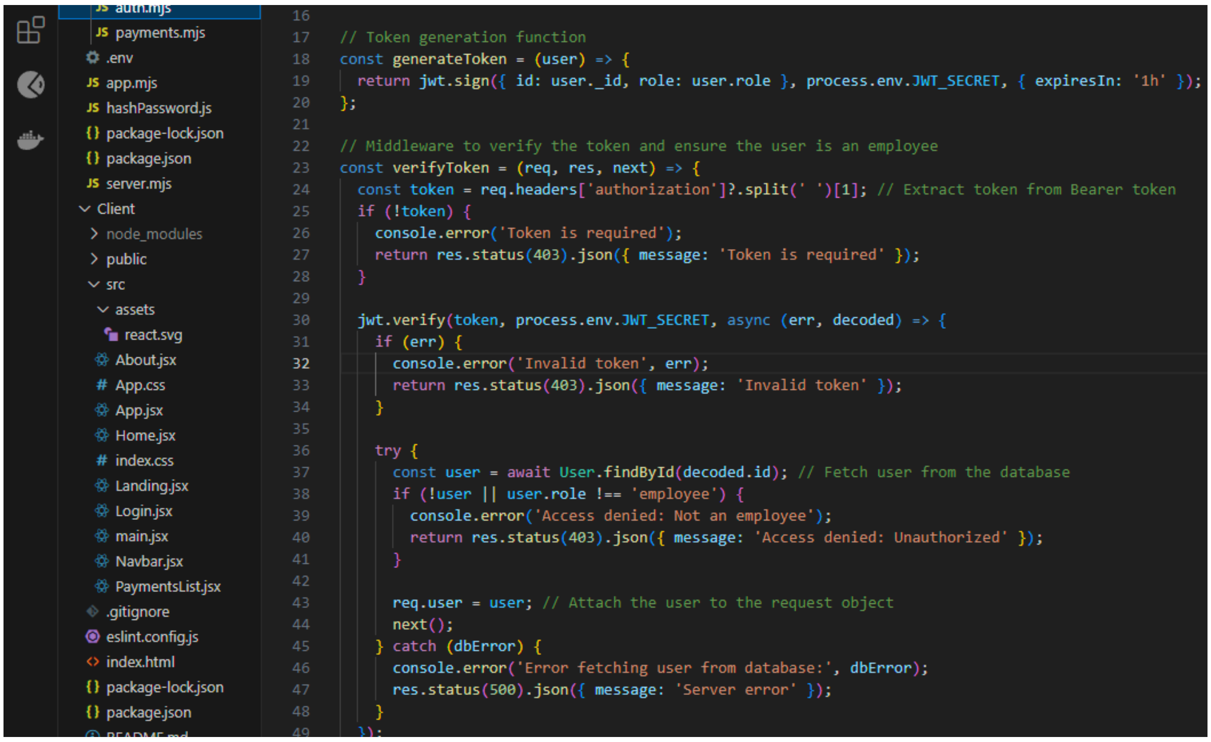
- Employee roles are enforced during login and API access.
- Only accounts with the role of employee can access sensitive endpoints like payment data.



```javascript
import express from 'express';
import Payment from '../models/Payment.mjs';
import { verifyToken } from './auth.mjs'; // Middleware to verify JWT

const router = express.Router();

// GET /api/payments - Fetch all payments (only for employees)
router.get('/', verifyToken, async (req, res) => {
  try {
    const { role } = req.user; // Extract role from token

    // Restrict access to employees only
    if (role !== 'employee') {
      console.error(`Access denied for role: ${role}`);
      return res.status(403).json({ message: 'Access denied: Only employees can view payments' });
    }

    // Fetch all payments
    const payments = await Payment.find().populate('userId', 'username email'); // Optional: Populate user details
    res.status(200).json(payments);
  } catch (error) {
    console.error('Error fetching payments:', error);
    res.status(500).json({ message: 'Server error' });
  }
});

export default router;
```

3. Secure Authentication:

- JWTs are generated upon login, ensuring secure stateless authentication.
- Tokens are validated for both identity and role, restricting access based on preconfigured roles.



```javascript

// Token generation function
const generateToken = (user) => {
  return jwt.sign({ id: user._id, role: user.role }, process.env.JWT_SECRET, { expiresIn: '1h' });
};

// Middleware to verify the token and ensure the user is an employee
const verifyToken = (req, res, next) => {
  const token = req.headers['authorization']?.split(' ')[1]; // Extract token from Bearer token
  if (!token) {
    console.error('Token is required');
    return res.status(403).json({ message: 'Token is required' });
  }

  jwt.verify(token, process.env.JWT_SECRET, async (err, decoded) => {
    if (err) {
      console.error('Invalid token', err);
      return res.status(403).json({ message: 'Invalid token' });
    }

    try {
      const user = await User.findById(decoded.id); // Fetch user from the database
      if (!user || user.role !== 'employee') {
        console.error('Access denied: Not an employee');
        return res.status(403).json({ message: 'Access denied: Unauthorized' });
      }

      req.user = user; // Attach the user to the request object
      next();
    } catch (dbError) {
      console.error('Error fetching user from database:', dbError);
      res.status(500).json({ message: 'Server error' });
    }
  });
```

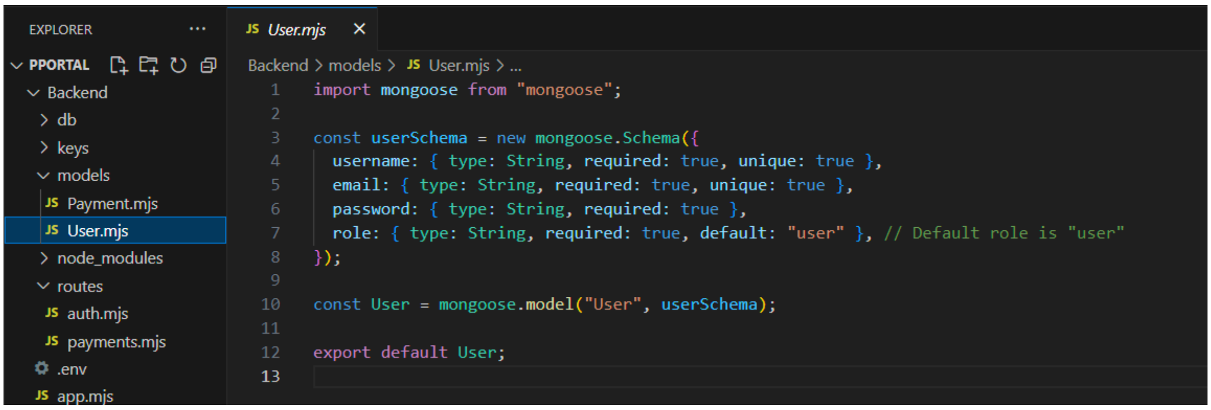
Customer Portal

Also has use roles implemented to prevent employees from accessing the user portal and vice-cersa.

1. Preconfigured Role Validation (User Role):
- Registration functionality assigns the user role by default.

The role field in the User schema for BOTH THE EMPLOYEE AND CUSTOMER PORTAL is assigned as "user" during registration so that default/non-employee accounts have the user role.

Backend\models\User.mjs:



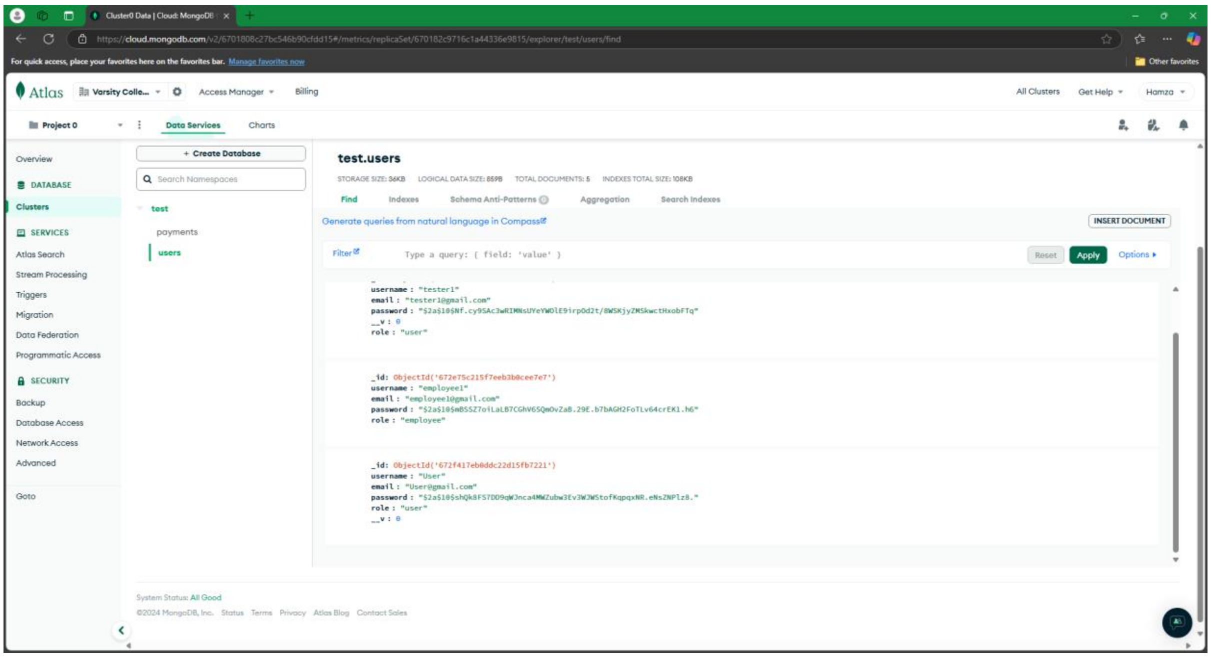
```javascript
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, required: true, default: "user" }, // Default role is "user"
});

const User = mongoose.model("User", userSchema);

export default User;
```

2. Role-Based Access for Payment Features:
- Role validation ensures that only accounts with the role user can access payment forms and associated routes.

3. JWT for Secure Authentication:

- JWT tokens are required for accessing all secured pages, ensuring users cannot access routes without proper authentication.

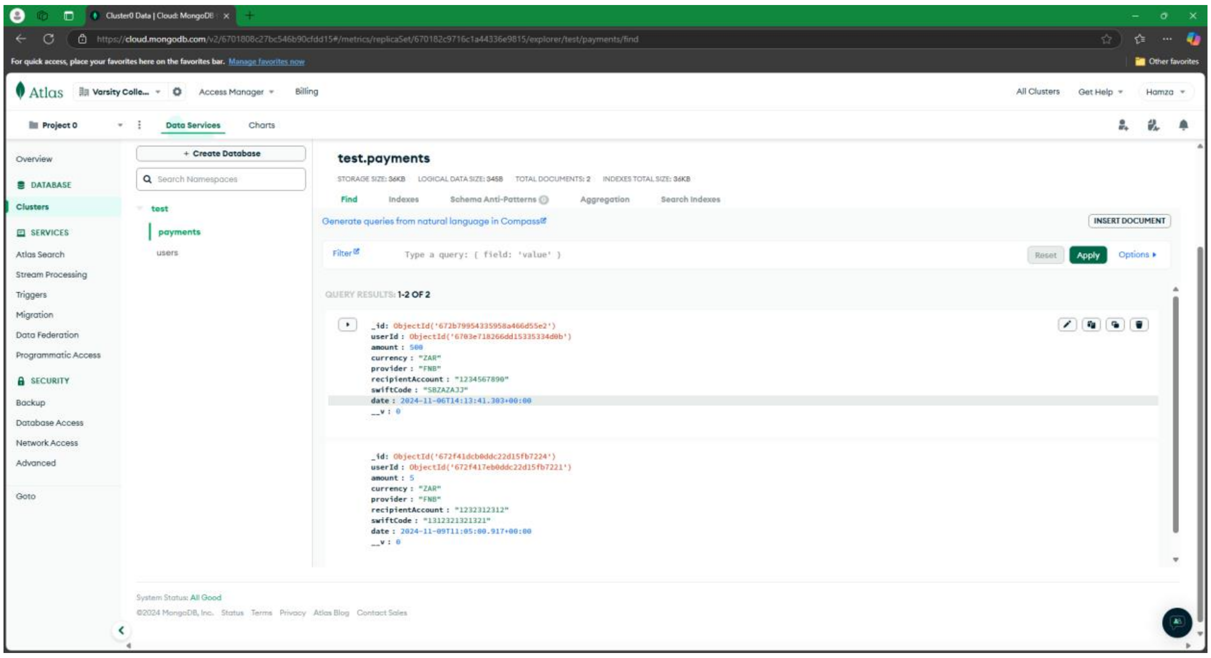
Mongo DB



test.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 859B TOTAL DOCUMENTS: 5 INDEXES TOTAL SIZE: 108KB

```
username : "tester1"
email : "tester1@gmail.com"
password : "$2a$10$Nf.cy9SAc3wRIMNsUYeYWDlE9irpOd2t/8WSKjyZMSkwctHxobFTq"
__v : 0
role : "user"

_id: ObjectId('672e75c215f7eeb3b0cee7e7')
username : "employee1"
email : "employee1@gmail.com"
password : "$2a$10$mBSSZ7oiLaLB7CGhV6SQmOvZaB.29E.b7bAGH2FoTLv64crEK1.h6"
role : "employee"

_id: ObjectId('672f417eb0ddc22d15fb7221')
username : "User"
email : "User@gmail.com"
password : "$2a$10$shQk8FS7DD9qWJnca4MWZubw3Ev3WJWStofKqpqxNR.eNsZNPlz8."
role : "user"
__v : 0
```



test.payments

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 345B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 36KB

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('672b79954335958a466d55e2')
userId : ObjectId('6703e718266dd15335334d0b')
amount : 500
currency : "ZAR"
provider : "FNB"
recipientAccount : "1234567890"
swiftCode : "SBZAZAJJ"
date : 2024-11-06T14:13:41.303+00:00
__v : 0

_id: ObjectId('672f41dcb0ddc22d15fb7224')
userId : ObjectId('672f417eb0ddc22d15fb7221')
amount : 5
currency : "ZAR"
provider : "FNB"
recipientAccount : "1232312312"
swiftCode : "1312321321321"
date : 2024-11-09T11:05:00.917+00:00
__v : 0
```

References

- Auth0 Blog. Implement Role-Based Access Control in Node.js with Express.

Available at: https://auth0.com/blog/role-based-access-control-rbac-and-node-js-api/

- DigitalOcean. Building a Secure Role-Based Access Control System in Express. Available at: https://www.digitalocean.com/community/tutorials/nodejs-role-based-access-control-api
- JWT.io. JSON Web Tokens Introduction. Available at: https://jwt.io/introduction

- FreeCodeCamp. Using JSON Web Tokens (JWT) for User Authentication in Node.js. Available at: https://www.freecodecamp.org/news/how-to-use-jwt-to-authenticate-and-authorize-users-in-node-js-d7c7e375d81e/

The overall functioning of the web app	• The web app is not functioning or only partially functioning.	• The web application is correctly configured and secured. Information processed on the customer portal appears in the staff portal correctly.	• The provided software shows additional research to provide an exceptional implementation
[20 Marks]	**0 – 9 Marks**	**10 – 14 Marks**	**15 – 20 Marks**

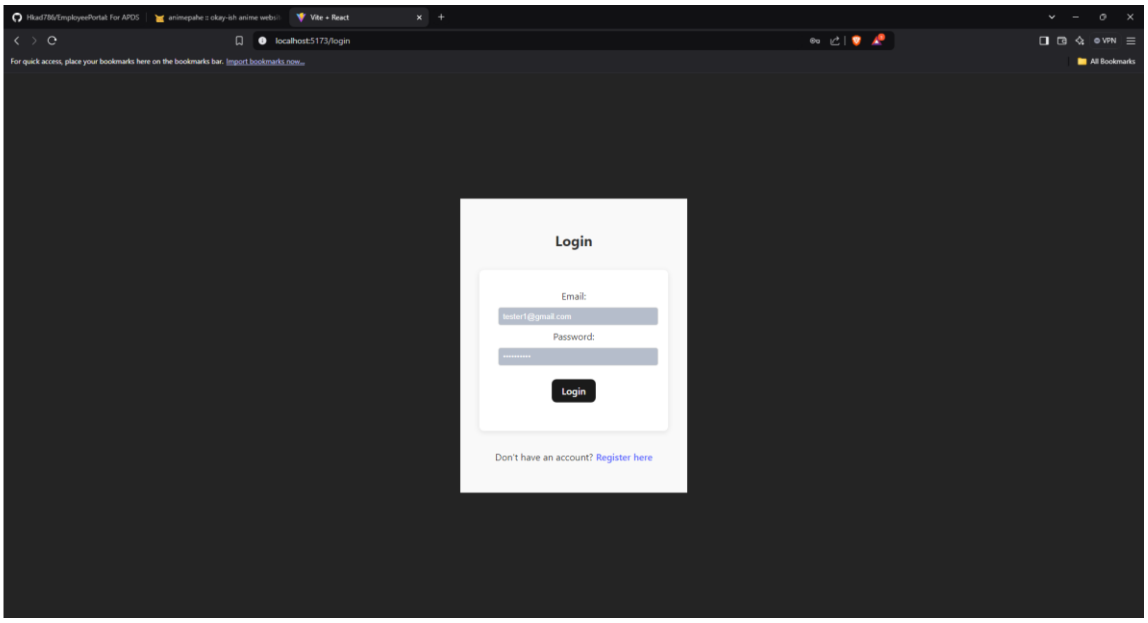
The overall functioning of the web app

(Note: The Functionality of the app show below works hand in hand with the database shown above which would show the matching info along with a demo video showing all functionality possible in demo video)

1. Logging In and Role-Based Access Control:

The Customer portal only allows customer accounts to login and supplies them with a token and vice versa with the Employee portal.

Customer Portal Functions shown.


Hkad786/EmployeePortal: For APDS | animepahe :: okay-ish anime websi | Vite + React

localhost:5173/login

For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

All Bookmarks

Login

Email:

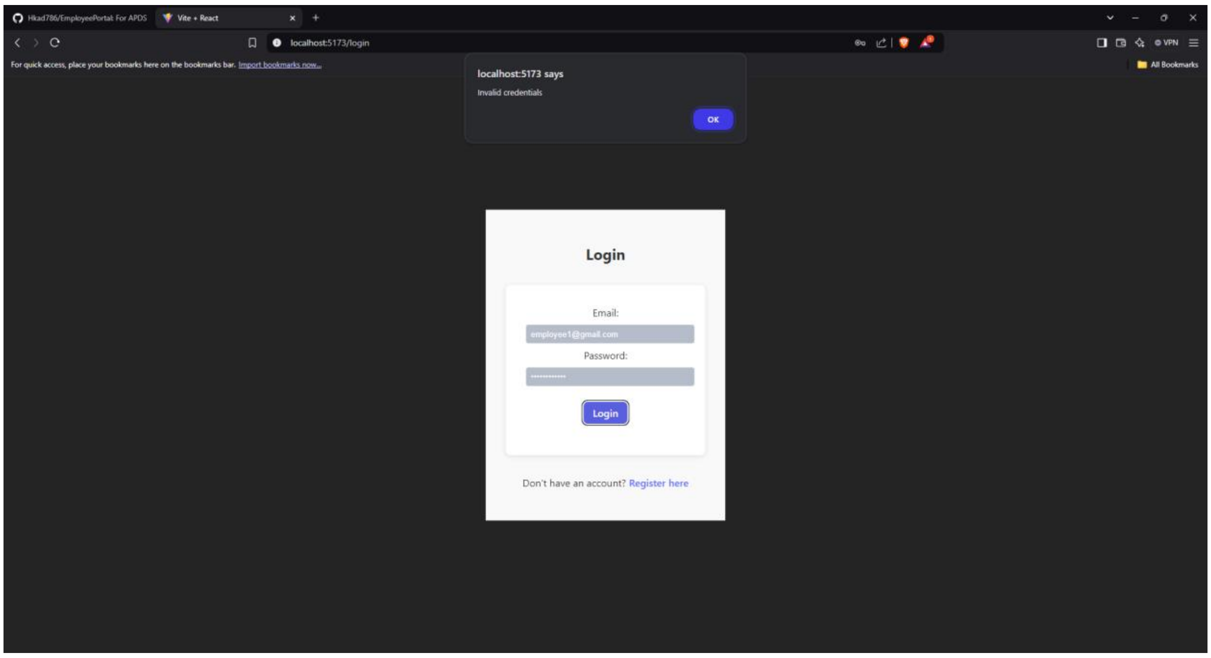
tester1@gmail.com

Password:

•••••••••••

Login

Don't have an account? Register here


Hkad786/EmployeePortal: For APDS
Vite + React
localhost:5173/login
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
All Bookmarks
OK
Login

localhost:5173 says

Invalid credentials

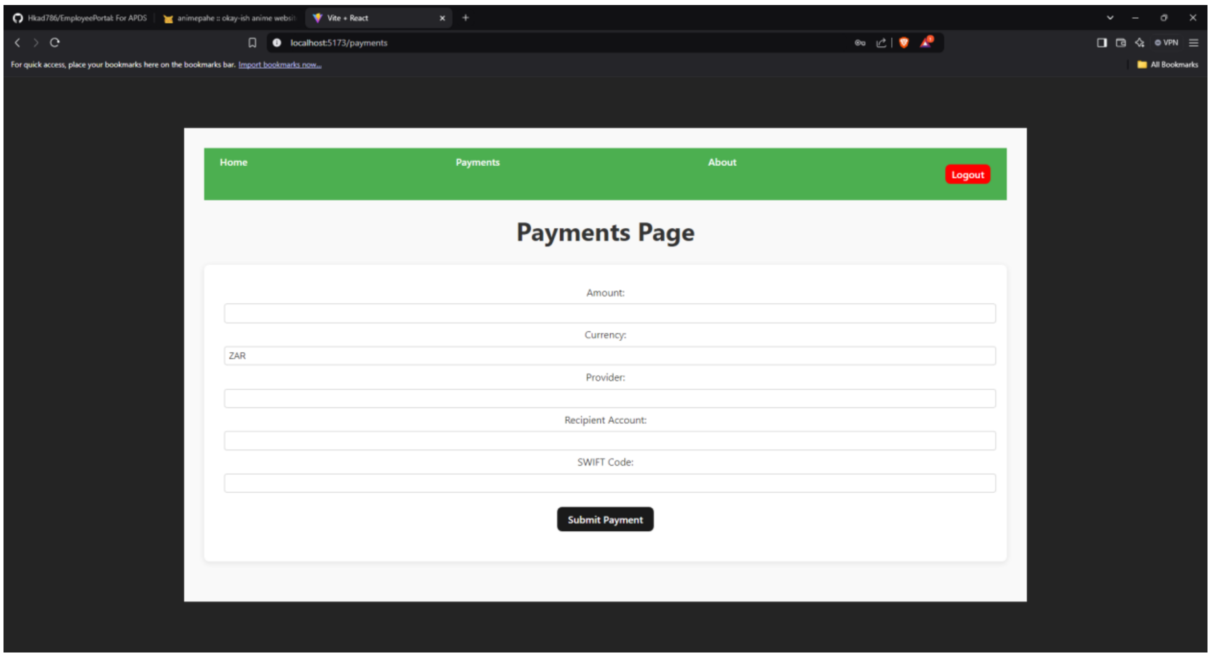
Login

Email: employee1@gmail.com

Password: ••••••••••••

Don't have an account? Register here

Only users are allowed, and employees are not allowed.


Hkad786/EmployeePortal: For APDS
animepahe :: okay-ish anime website
Vite + React
localhost:5173/payments
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
All Bookmarks
Home
Payments
About
Logout
Submit Payment

Payments Page

Amount:

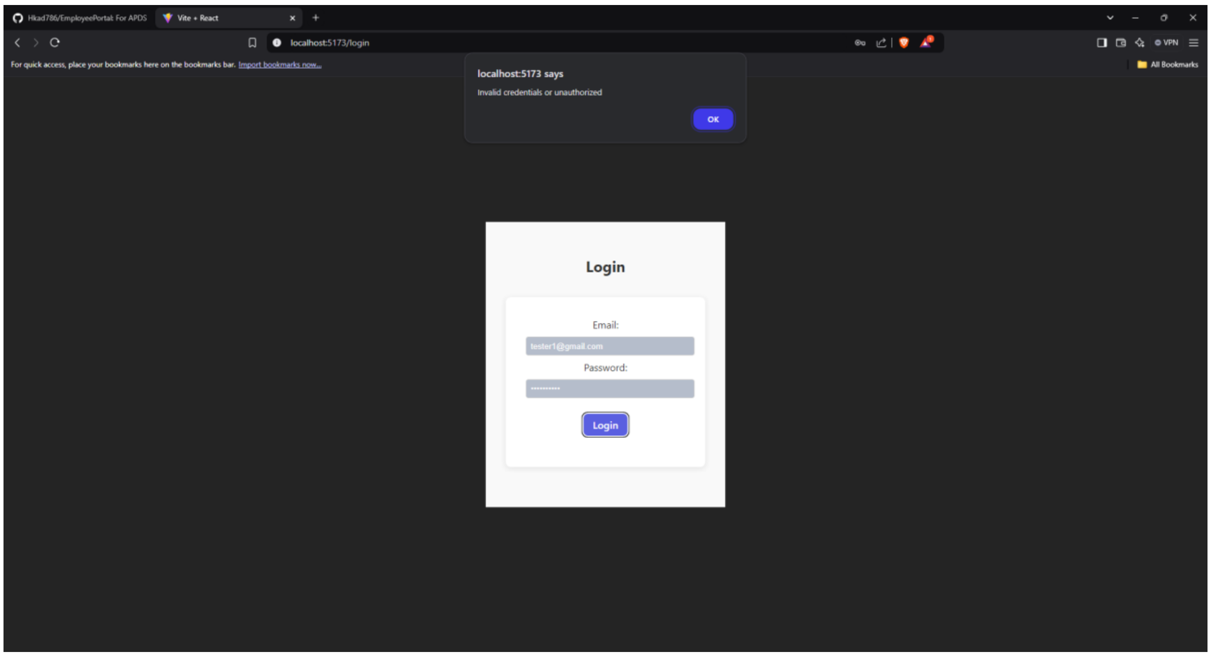
Currency: ZAR

Provider:

Recipient Account:

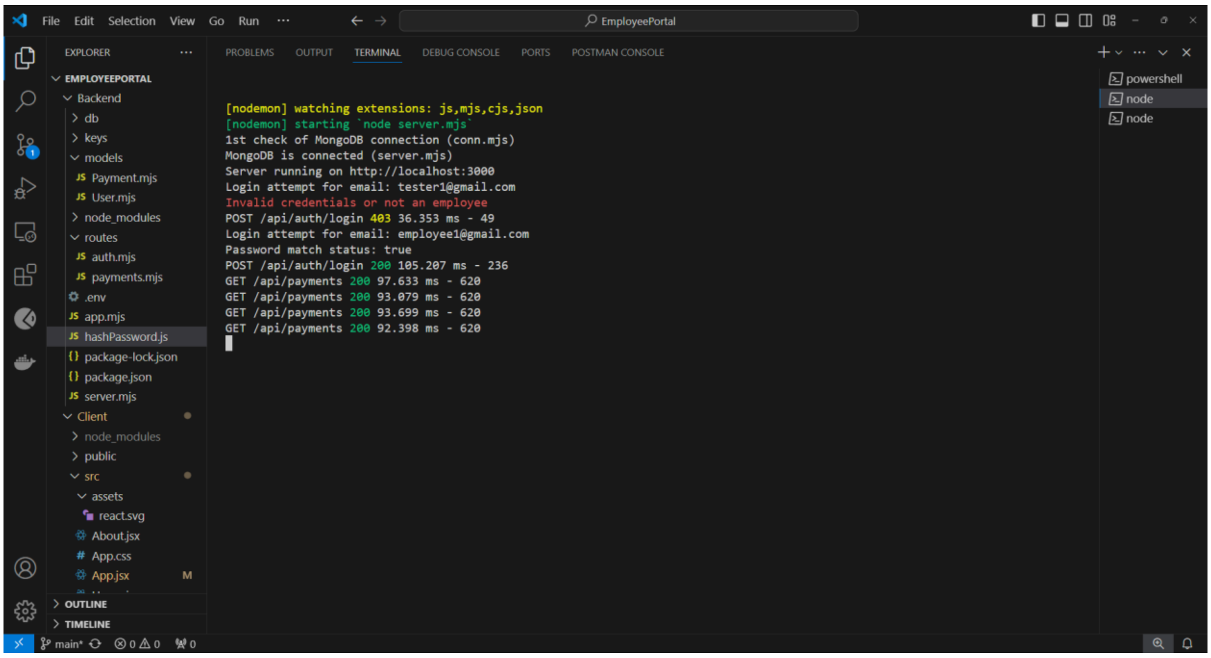
SWIFT Code:

Employee portal


localhost:5173 says

Invalid credentials or unauthorized

User accounts don’t work on the employee portal which only has login and no register functionality.


```
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.mjs`
1st check of MongoDB connection (conn.mjs)
MongoDB is connected (server.mjs)
Server running on http://localhost:3000
Login attempt for email: tester1@gmail.com
Invalid credentials or not an employee
POST /api/auth/login 403 36.353 ms - 49
Login attempt for email: employee1@gmail.com
Password match status: true
POST /api/auth/login 200 105.207 ms - 236
GET /api/payments 200 97.633 ms - 620
GET /api/payments 200 93.079 ms - 620
GET /api/payments 200 93.699 ms - 620
GET /api/payments 200 92.398 ms - 620
```

References (2)

Atlassian, 2020. *What is DevOps? | Atlassian*. [online] Atlassian. Available at: <https://www.atlassian.com/devops> [Accessed 11 November 2024].

Anon. 2024. *SonarSource/sonarqube: Continuous Inspection*. [online] GitHub. Available at: <https://github.com/SonarSource/sonarqube> [Accessed 11 November 2024].

Anon. 2024. *What Is MongoDB?* [online] MongoDB. Available at: <https://www.mongodb.com/company/what-is-mongodb> [Accessed 11 November 2024].

https://www.freecodecamp.org/news/how-to-use-jwt-to-authenticate-and-authorize-users-in-node-js-d7c7e375d81e/

https://auth0.com/blog/role-based-access-control-rbac-and-node-js-api/

https://www.digitalocean.com/community/tutorials/nodejs-role-based-access-control-api

(Used guide as well).