Name: Rishabh Singh

Assignment Name: Module 3 Activity 3: Middleware

PART 1

**Overview:**

The proposed middleware is a critical component in our application architecture and handles user authentication and authorization. Its primary function is to abstract the complexities of user security, reducing the burden on developers and ensuring consistent and secure authentication and authorization processes.

**Key Functions:**

Authentication: The middleware validates user credentials, ensuring the user is who they claim to be. It securely handles user logins and token generation.

Authorization: It enforces role-based access control, allowing or denying access to specific resources or actions based on user roles and permissions.

Logging: The middleware logs authentication and authorization events for security and auditing purposes, providing insights into user activities.

**Interaction with Other Systems:**

The middleware interacts with the "Web UI," where users interact with the application.

It communicates with an "Authentication Service" to validate user credentials.

It queries an "Authorization Service" to check user roles and permissions.

It retrieves user information from a "User Database" to perform authentication and authorization tasks.

Middleware Interaction Diagram

[Web UI] <--> [Middleware] <--> [Authentication Service]

|

v

[Authorization Service]

|

v

[User Database]

**Benefits Analysis**

The middleware offers several benefits for the development process:

Modularity: The middleware promotes modular design by abstracting authentication and authorization logic. Developers can focus on building application-specific features without the need to handle security intricacies.

Reduced Complexity: Developers are relieved from the complexities of implementing secure user authentication and authorization within the application's core code. This results in cleaner, more maintainable, and efficient code.

Scalability: As the application grows, the middleware can scale independently to handle increased authentication and authorization traffic. It does not impose limitations on application scalability.

Security: Centralizing security logic in the middleware reduces the risk of security vulnerabilities. Developers can rely on well-established security practices implemented in middleware.

Ease of Integration: The middleware provides well-defined APIs or libraries for integration, saving development time and effort. Developers can seamlessly incorporate it into their applications.

**Reflection**

This assignment provided valuable insights into the pivotal role middleware plays in information processing, particularly in the context of user authentication and authorization. I found the task of designing a middleware system to be both challenging and intellectually rewarding.

**Challenges:**

Complexity: Creating a robust middleware system involves handling intricate security processes. Ensuring secure user authentication and authorization requires careful consideration of various scenarios and potential vulnerabilities.

**Outcome:**

Middleware's Importance: This assignment highlighted middleware's critical role in separating concerns, improving security, and enhancing application scalability and maintainability.

PART 2

Bookhub code:

const express=require("express");

const app=express();

const bodyParser=require("body-parser");

app.use(bodyParser.json());

books=[]

const logInfo=(*req*,*res*,*next*)=>{

    const timestamp = **new** *Date*().toISOString();

    const method = *req*.method;

    const url = *req*.originalUrl;

    console.log(`[${timestamp}] ${method} ${url}`);

    next();

};

const checkAuth=(*req*,*res*,*next*)=>{

    if (*req*.query.auth==='true'){

        return next();

    }else{

*res*.send("not Authorized");

    }

};

app.use(logInfo);

app.get("/",(*req*,*res*)=>{

*res*.send("welcome to bookHub");

});

app.post("/cart/add",checkAuth,(*req*,*res*)=>{

    const book=*req*.body;

    // console

    // books.push(book.name);

*res*.send(`book with the name ${book.name} added to the database`);

})

app.delete("/cart/remove/:id",checkAuth,(*req*,*res*)=>{

    const id=*req*.params.id;

    books.splice(id,1);

*res*.send(`deleted the book with id ${id}`);

})

app.listen(3000,()=>{

    console.log("Server is running on port 3000");

});

Screenshots:

A screenshot of a computer screen

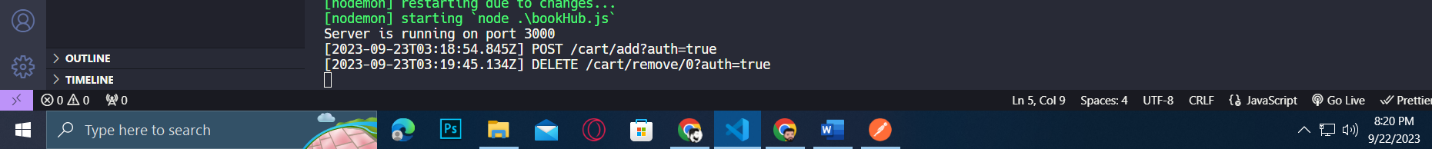
Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generated

Lofinfo global middleware:



**Code explanation:**

**Import Dependencies and Create an Express App:**

It imports the required modules: express, body-parser, and creates an Express application.

const express = require("express");

const app = express();

const bodyParser = require("body-parser");

app.use(bodyParser.json());

**Initialize an Array for Books:**

It initializes an empty array called books to store book data.

books = [];

**Define Middleware Functions:**

**logInfo middleware:** This middleware logs incoming requests to the console. It is applied globally to all routes in the application.

**checkAuth middleware:** This middleware checks if the auth query parameter is set to 'true'. If it is, it allows the request to continue to the next middleware or route; otherwise, it sends a "not Authorized" response.

const logInfo=(req,res,next)=>{

const timestamp = new Date().toISOString();

const method = req.method;

const url = req.originalUrl;

console.log(`[${timestamp}] ${method} ${url}`);

next();

};

const checkAuth = (req, res, next) => {

if (req.query.auth === 'true') {

return next();

} else {

res.send("not Authorized");

}

};

Apply the logInfo Middleware Globally:

The app.use(logInfo) line applies the logInfo middleware globally to all routes, so every incoming request will be logged to the console.

Define Routes:

/: This route responds with a "welcome to bookHub" message when users access the root URL.

/cart/add: This route allows users to add books to the cart. It checks for authentication using the checkAuth middleware. It parses the request body to extract book data and adds it to the books array if authenticated.

/cart/remove/:id: This route allows users to remove books from the cart by specifying an id parameter. It also checks for authentication using the checkAuth middleware and removes the specified book from the books array.

app.get("/", (req, res) => {

res.send("welcome to bookHub");

});

app.post("/cart/add", checkAuth, (req, res) => {

const book = req.body;

books.push(book.name);

res.send(`book with the name ${book.name} added to the database`);

});

app.delete("/cart/remove/:id", checkAuth, (req, res) => {

const id = req.params.id;

books.splice(id, 1);

res.send(`deleted the book with id ${id}`);

});

Start the Express Server:

It starts the Express server on port 3000 and logs a message indicating that the server is running.

app.listen(3000, () => {

console.log("Server is running on port 3000");

});

In summary, this code sets up a basic Express.js server that handles adding and removing books from a cart. It includes middleware for logging requests and checking for authentication based on a auth query parameter. Please note that this is a simplified example, and for a real-world application, you would typically use more robust authentication mechanisms and possibly a database to store book data.