NAME	RAMENDRA SHUKLA
UID	23BCS12146
SECTION	622-B

EXPERIMENT 6.1

Title

Middleware Implementation for Logging and Bearer Token Authentication

Objective

Learn how to build and integrate middleware functions in an Express.js application to handle request logging and protect routes using a Bearer token. This task helps you understand middleware flow, request validation, and how to enforce secure access to specific endpoints in a Node.js backend.

Task Description

Create an Express.js server and implement two custom middleware functions. The first middleware should log the HTTP method, request URL, and timestamp for every incoming request. The second middleware should check for an Authorization header that includes a Bearer token. Only requests that include the token mysecrettoken should be allowed to access the protected route; all other requests should be denied with appropriate error messages. Apply the logging middleware globally to all routes. Create at least two routes: one public route accessible without authentication, and one protected route that requires the correct Bearer token (mysecrettoken) to access. Test both routes using curl or Postman to demonstrate how logging and token-based authentication work together.

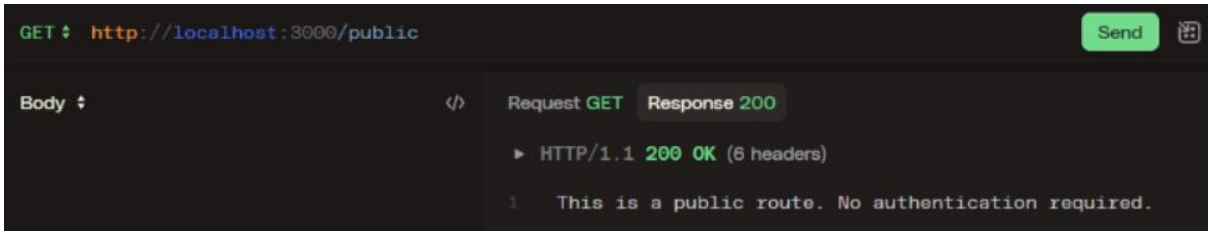
CODE :

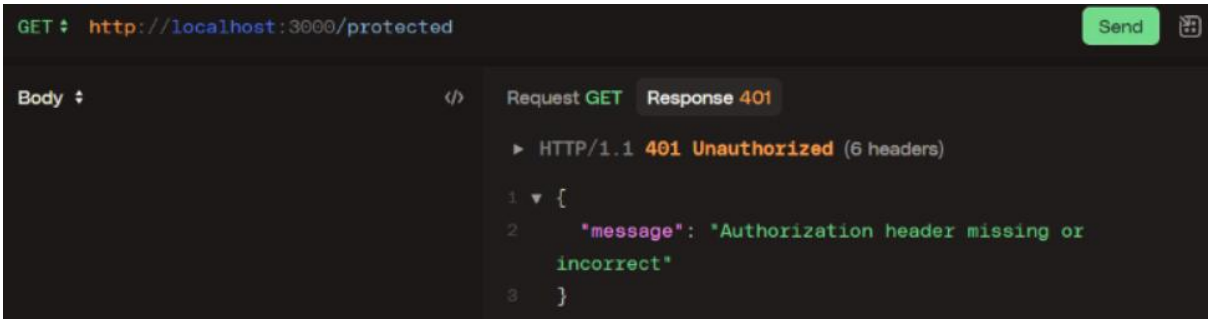
```
const express = require("express");

const app = express();
```

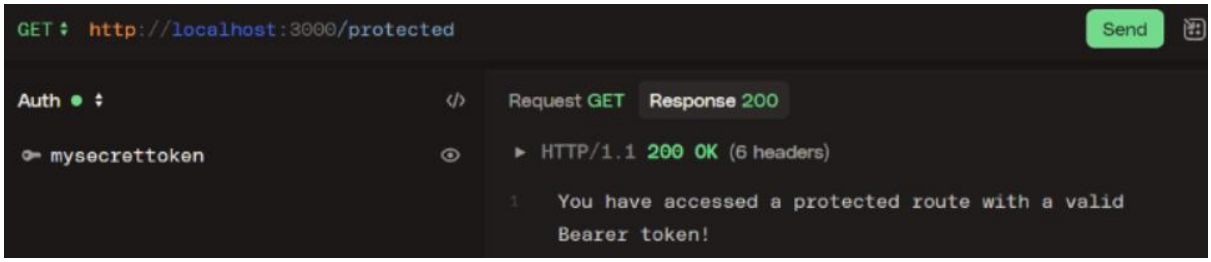
```
const loggerMiddleware = (req, res, next) => {
  const currentTime = new Date().toISOString();
  console.log(`[${currentTime}] ${req.method} ${req.originalUrl}`);
  next();
};

app.use(loggerMiddleware);

const authMiddleware = (req, res, next) => {
  const authHeader = req.headers["authorization"];
  if (!authHeader) {
    return res.status(401).json({ error: "Authorization header missing" });
  }
  const parts = authHeader.split(" ");
  if (parts.length !== 2 || parts[0] !== "Bearer") {
    return res.status(400).json({ error: "Invalid token format. Use Bearer <token>" });
  }
  const token = parts[1];
  if (token !== "mysecrettoken") {
    return res.status(403).json({ error: "Access denied: Invalid token" });
  }
  next();
};

app.get("/public", (req, res) => {
  res.json({
    message: "Welcome to the public route! No authentication needed.",
```

```
  });
});

app.get("/protected", authMiddleware, (req, res) => {
  res.json({
    message: "Access granted! You have reached the protected route.",
  });
});

const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

OUTPUT:







EXPERIMENT 6.2

Title

JWT Authentication for Secure Banking API Endpoints

Objective

Learn how to implement secure authentication in an Express.js application using JSON Web Tokens (JWT). This task helps you understand how to generate tokens, verify them in middleware, and protect sensitive API routes to ensure only authorized users can access banking operations.

Task Description

Create an Express.js banking API with endpoints for viewing account balance (/balance), depositing money (/deposit), and withdrawing money (/withdraw). Implement a /login route that accepts a username and password (hardcoded for this exercise) and returns a signed JWT token upon successful login. Use a middleware function to verify the JWT token in the Authorization header before allowing access to the protected banking routes. Handle common errors, such as missing or invalid tokens and insufficient balance for withdrawals. Test the API by first logging in to obtain a token, then sending it as a Bearer token in requests to protected endpoints to demonstrate secure access control.

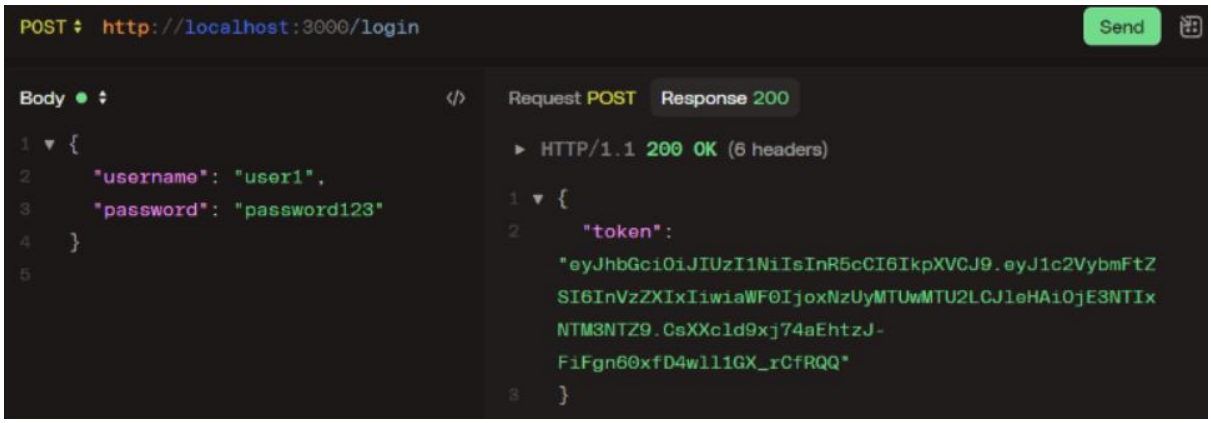
CODE

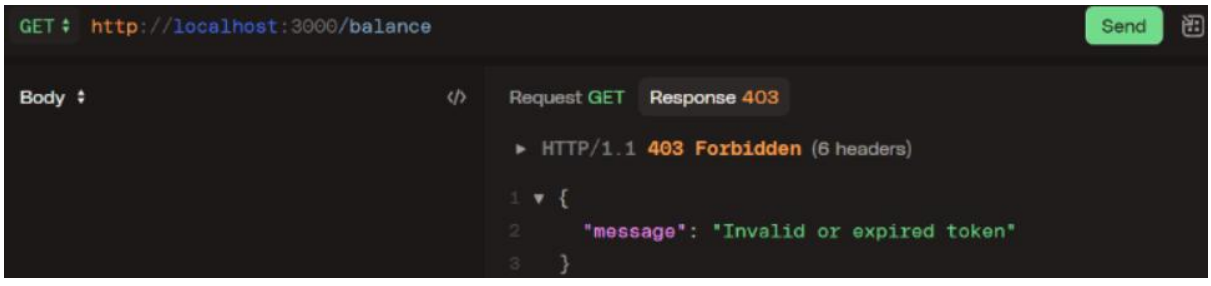
```
const express = require("express");

const jwt = require("jsonwebtoken");


const app = express();

app.use(express.json());
```

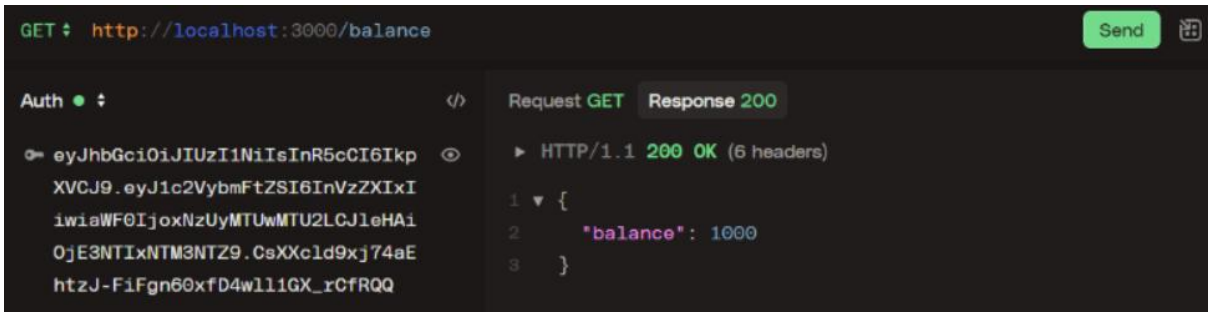
```
const SECRET_KEY = "myjwtsecretkey";

let accountBalance = 1000;

app.post("/login", (req, res) => {
  const { username, password } = req.body;

  if (username === "user123" && password === "pass123") {
    const token = jwt.sign({ username }, SECRET_KEY, { expiresIn: "1h" });
    return res.json({ message: "Login successful", token });
  } else {
    return res.status(401).json({ error: "Invalid username or password" });
  }
});

const authMiddleware = (req, res, next) => {
  const authHeader = req.headers["authorization"];
  if (!authHeader) {
    return res.status(401).json({ error: "Authorization header missing" });
  }

  const token = authHeader.split(" ")[1];
  if (!token) {
    return res.status(401).json({ error: "Token missing" });
  }

  try {
```

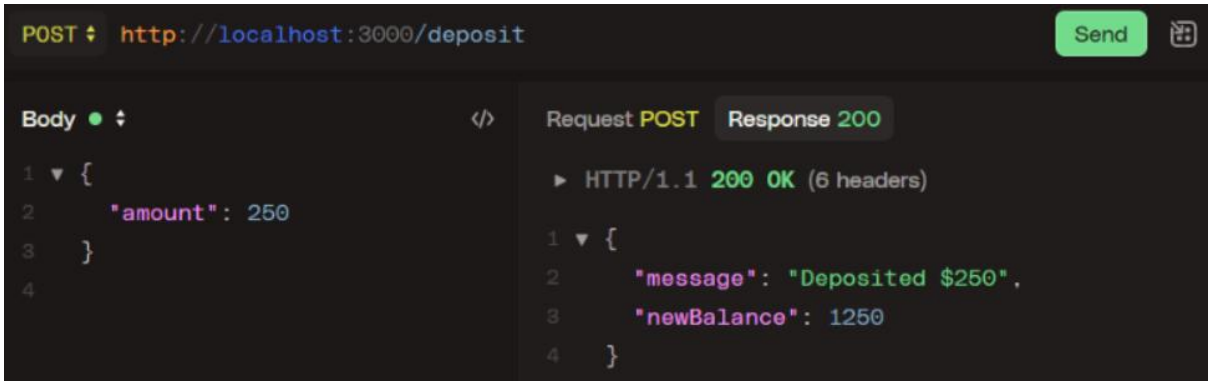
```
    const decoded = jwt.verify(token, SECRET_KEY);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(403).json({ error: "Invalid or expired token" });
  }
};

app.get("/balance", authMiddleware, (req, res) => {
  res.json({ username: req.user.username, balance: accountBalance });
});

app.post("/deposit", authMiddleware, (req, res) => {
  const { amount } = req.body;
  if (!amount || amount <= 0) {
    return res.status(400).json({ error: "Invalid deposit amount" });
  }
  accountBalance += amount;
  res.json({ message: "Deposit successful", balance: accountBalance });
});

app.post("/withdraw", authMiddleware, (req, res) => {
  const { amount } = req.body;
  if (!amount || amount <= 0) {
    return res.status(400).json({ error: "Invalid withdrawal amount" });
  }
  if (amount > accountBalance) {
    return res.status(400).json({ error: "Insufficient balance" });
```

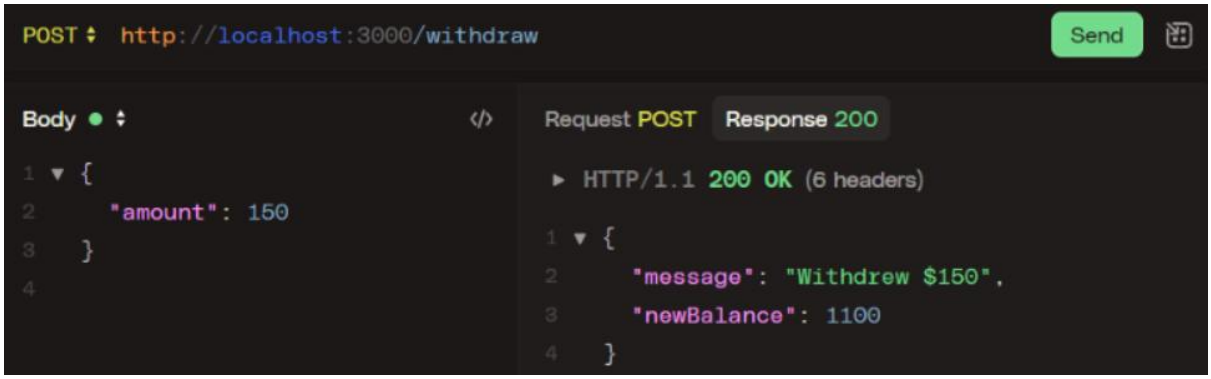
```
  }
  accountBalance -= amount;
  res.json({ message: "Withdrawal successful", balance: accountBalance });
});

const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```











EXPERIMENT 6.3

Title

Account Transfer System with Balance Validation in Node.js

Objective

Learn how to implement a secure money transfer API in Node.js and MongoDB without using database transactions. This task helps you understand how to perform dependent multi-document updates by carefully validating balances and handling errors in application logic to ensure consistent and correct results.

Task Description

Create a Node.js and Express.js application with a MongoDB database to simulate a bank account transfer system. Implement an API endpoint to transfer money from one user account to another. Before updating balances, check if the

sender has enough balance and proceed only if the condition is satisfied. Even though database-level transactions are not used, ensure logical correctness through proper validation and sequential updates. Return meaningful error messages when the sender's balance is insufficient or if either account does not exist. Test the API using sample user accounts, and demonstrate both successful and failed transfer scenarios to show how logical checks prevent invalid state updates.

CODE:

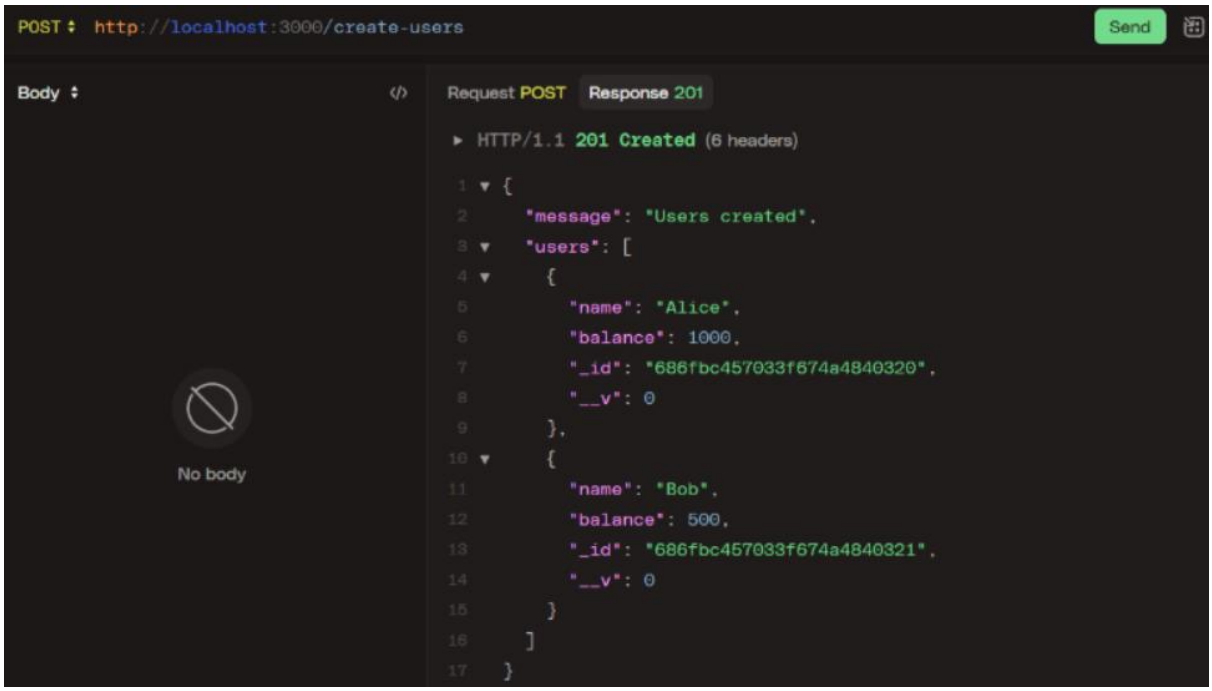
```
const express = require("express");
const mongoose = require("mongoose");

const app = express();
app.use(express.json());

mongoose
  .connect("mongodb://127.0.0.1:27017/bank_transfer_system", {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => console.log("✅ Connected to MongoDB"))
  .catch((err) => console.error("❌ MongoDB connection error:", err));

const accountSchema = new mongoose.Schema({
  name: String,
  balance: Number,
```

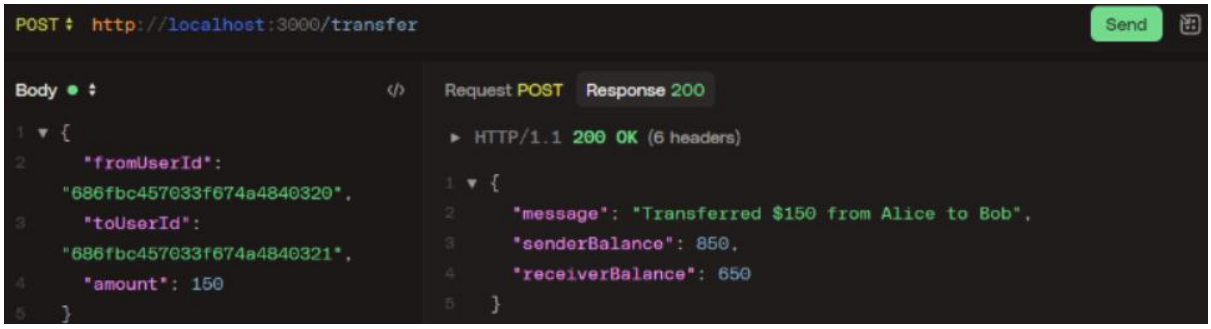
```
});

const Account = mongoose.model("Account", accountSchema);

async function seedAccounts() {
  const count = await Account.countDocuments();
  if (count > 0) return console.log("⚠️ Sample accounts already exist.");

  await Account.insertMany([
    { name: "Alice", balance: 1000 },
    { name: "Bob", balance: 500 },
  ]);

  console.log("✅ Sample accounts created successfully!");
}

app.get("/accounts", async (req, res) => {
  const accounts = await Account.find();
  res.json(accounts);
});

app.post("/transfer", async (req, res) => {
  const { from, to, amount } = req.body;

  if (!from || !to || !amount || amount <= 0) {
```

```
  return res.status(400).json({ error: "Invalid transfer details" });
}

try {
  const sender = await Account.findOne({ name: from });
  const receiver = await Account.findOne({ name: to });

  if (!sender) return res.status(404).json({ error: "Sender not found" });
  if (!receiver) return res.status(404).json({ error: "Receiver not found" });

  if (sender.balance < amount) {
    return res.status(400).json({ error: "Insufficient balance" });
  }

  sender.balance -= amount;
  receiver.balance += amount;

  await sender.save();
  await receiver.save();

  res.json({
    message: "Transfer successful",
    from: sender.name,
    to: receiver.name,
    amount,
```

```
      senderBalance: sender.balance,
      receiverBalance: receiver.balance,
    });
  } catch (err) {
    console.error("❌ Error during transfer:", err);
    res.status(500).json({ error: "Internal server error" });
  }
});

const PORT = 3000;
app.listen(PORT, async () => {
  console.log(`🚀 Server running on http://localhost:${PORT}`);
  await seedAccounts();
});
```

OUTPUT:




POST http://localhost:3000/transfer
Send
Body
Request POST
Response 200
HTTP/1.1 200 OK (6 headers)
{
"fromUserId": "686fbc457033f674a4840320",
"toUserId": "686fbc457033f674a4840321",
"amount": 150
}
{
"message": "Transferred $150 from Alice to Bob",
"senderBalance": 850,
"receiverBalance": 650
}


POST http://localhost:3000/transfer
Send
Body
Request POST
Response 400
HTTP/1.1 400 Bad Request (6 headers)
{
"fromUserId": "686fbc457033f674a4840320",
"toUserId": "686fbc457033f674a4840321",
"amount": 900
}
{
"message": "Insufficient balance"
}