Bharatiya Vidya Bhavan's

SARDAR PATEL INSTITUTE OF TECHNOLOGY

An Autonomous Institute Affiliated To University Of Mumbai

Munshi Nagar,Andheri (W) Mumbai 400 058

C.I.T.L. EXPERIMENT 6

Topic : Inventory Management System

Submitted By:

Akash Panicker	2021300089
Mahesh Patil	2021300095
Rohit Phalke	2021300100
Adwait Purao	2021300101

Submitted To:

Prof. Sunil Ghane

Aim:
Design secured Web application using web token

Problem Statement:
Develop an inventory management system for a retail store that efficiently tracks and manages the inventory of products. The system should provide real-time updates on stock levels, generate alerts for low stock items, enable easy addition and removal of products, and offer insights into sales trends to optimize restocking decisions.

Theory:
Designing a secure web application using JSON Web Tokens (JWTs) involves implementing several security measures to protect user data and prevent unauthorized access. Below is a comprehensive overview of the theoretical aspects and practical implementation of JWT-based authentication:

- **What is JWT?**
  JWTs are an open standard (RFC 7519) for securely transmitting information between two parties as a JSON object. They are self-contained, meaning they contain all the necessary information to be verified and trusted without requiring additional server-side calls.

- **Structure of a JWT:**
  A JWT consists of three parts separated by periods:
  - **Header:** Contains information about the signing algorithm and token type.
  - **Payload:** Contains claims, which are statements about the user or application. Claims can be any key-value pair, such as user ID, role, or expiration time.
  - **Signature:** Ensures the integrity of the header and payload. It is generated using a secret key or a public/private key pair.

- **JWT Authentication Flow:**
  - User Login: The user enters their credentials (username and password) into the login form.
  - Credential Validation: The server validates the user's credentials against a database or authentication service.
  - JWT Generation: Upon successful validation, the server generates a JWT containing user claims.
  - JWT Response: The server sends the JWT back to the client, typically via an HTTP response header.
  - JWT Storage: The client stores the JWT securely in local storage or a cookie.

    - JWT Authorization: Subsequent requests from the client include the JWT in the Authorization header.
    - JWT Validation: The server verifies the signature and validity of the JWT to authenticate the user.
    - Access Control: Based on the user's claims, the server grants or denies access to resources.

- **Security Measures:**
    - HTTPS Enforcement: Use HTTPS to encrypt all communication between the client and server to prevent interception and data tampering.
    - Secret Key Protection: Keep the secret key used for signing JWTs confidential and secure to prevent unauthorized token generation.
    - Token Expiration: Set an appropriate expiration time for JWTs to limit their validity and prevent unauthorized access after a certain period.
    - Token Blacklist: Implement a mechanism to invalidate and blacklist compromised or expired tokens to prevent their reuse.
    - Regular Security Audits: Conduct regular security audits to identify and address potential vulnerabilities in the authentication system.

- **Practical Implementation:**
    - Choose a JWT Library: Select a well-maintained and trusted JWT library for your programming language or framework.
    - Generate and Sign JWTs: Use the library's functions to generate and sign JWTs with appropriate claims and expiration times.
    - Validate JWTs: Implement logic on the server to validate incoming JWTs, ensuring their signature integrity and validity.
    - Protect JWTs: Store JWTs securely in local storage or cookies with appropriate access restrictions and encryption mechanisms.
    - Handle Token Invalidation: Implement mechanisms to invalidate and blacklist compromised or expired tokens.
    - Integrate with User Management: Integrate JWT-based authentication with your user management system to manage user sessions and permissions.
    - Monitor and Audit: Implement logging and monitoring tools to track JWT usage and identify potential security anomalies.

Screenshots:

Controller for Register and Login:

1. **Vendor**

```
router.post("/vendorregister", async (req, res) => {
  const { name, email, phone, role, password, cpassword } = req.body;
  if (!name || !email || !phone || !role || !password || !cpassword) {
    return res.status(422).json({ error: "All fields need to be filled"
});
  }

  try {
    const vendorExist = await Vendor.findOne({ email: email });
    if (vendorExist) {
      return res.status(409).json({ error: "Email already registered" });
    } else if (password != cpassword) {
      return res.status(422).json({ error: "Passwords do not match" });
    }
    const ven = new Vendor({ name, email, phone, password, cpassword });
    await ven.save();
    const pro = new Profile({ name: name, email: email, phone: phone,
Grole: role })
    await pro.save()
    return res.status(200).json({ msg: "Vendor registered successfully"
});
  } catch (error) {
    console.log(error);
    return res.status(500).json({ error: "Some unexpected error occured"
});
  }
});

router.post("/vendorsignin", async (req, res) => {
  const { email, password } = req.body;
  if (!email || !password) {
    return res.status(400).json({ error: "Please fill all required fields"
});
  }
  try {
    const emailExist = await Vendor.findOne({ email: email });
    if (emailExist) {
      const isMatch = await bcrypt.compare(password, emailExist.password);
      if (isMatch) {
        token = await emailExist.generateAuthToken();
        res.cookie(
          "inv_man",
          { token, role: "vendor", email: email },
          {
            expires: new Date(Date.now() + 604800),
            httpOnly: true,
          }
        );
        return res.status(200).json({ msg: "Login successful" });
      } else {
        return res.status(400).json({ error: "Login failed" });
      }
    } else {
      return res.status(400).json({ error: "Invalid credentials" });
    }
  } catch (error) {
    return res.status(500).json({ error: "Some unexpected error occured"
});
  }
});
```

2. **Company**

```
router.post("/companyregister", async (req, res) => {
  const { name, email, phone, role, password, cpassword } = req.body;

  if (!name || !email || !phone || !role || !password || !cpassword) {
    return res.status(422).json({ error: "All fields need to be filled"
});
  }

  try {
    const companyExist = await Company.findOne({ email: email });
    if (companyExist) {
      return res.status(409).json({ error: "Email already registered" });
    } else if (password != cpassword) {
      return res.status(422).json({ error: "Passwords do not match" });
    }
    const comp = new Company({ name, email, phone, password, cpassword });
    await comp.save();
    const pro = new Profile({ name: name, email: email, phone: phone,
Grole: role })
    await pro.save()
    return res.status(200).json({ msg: "Company registered successfully"
});
  } catch (error) {
    console.log(error);
    return res.status(500).json({ error: "Some unexpected error occured"
});
  }
});

router.post("/companysignin", async (req, res) => {
  const { email, password } = req.body;
  if (!email || !password) {
    return res.status(400).json({ error: "Please fill all required fields"
});
  }
  try {
    const emailExist = await Company.findOne({ email: email });
    if (emailExist) {
      const isMatch = await bcrypt.compare(password, emailExist.password);
      if (isMatch) {
        token = await emailExist.generateAuthToken();
        res.cookie(
          "inv_man",
          { token, role: "company", email: email },
          {
            expires: new Date(Date.now() + 604800),
            httpOnly: true,
          }
        );
        return res.status(200).json({ msg: "Login successful" });
      } else {
        return res.status(400).json({ error: "Login failed" });
      }
    } else {
      return res.status(400).json({ error: "Invalid credentials" });
    }
  } catch (error) {
    return res.status(500).json({ error: "Some unexpected error occured"
});
  }
});
```

Middleware for authentication:

1. **Vendor**

```
const jwt = require('jsonwebtoken')
const Vendor = require('../models/Vendor')


const vendorAuthenticate = async (req, res, next) => {
    try {
        const token = req.cookies.inv_man.token;
        const role = req.cookies.inv_man.role;
        const verifyToken = jwt.verify(token, process.env.SECRET_KEY)

        const findVendor = await Vendor.findOne({_id:verifyToken._id,
"tokens.token":token})

        if(!findVendor){
            throw new Error("Login Expired")
        }

        if(role!=="vendor"){
            res.status(401).json({msg:'Unauthorized access'})
        }

        req.token=token
        req.findVendor=findVendor
        req.userID=findVendor._id
        next()
    } catch (error) {
        res.status(401).json({msg:'Unauthorized access'})
    }
}
```

```
module.exports = vendorAuthenticate
```

2. **Company**

```
const jwt = require('jsonwebtoken')
const Company = require('../models/Company')


const companyAuthenticate = async (req, res, next) => {
    try {
        const token = req.cookies.inv_man.token;
        const role = req.cookies.inv_man.role;
        const verifyToken = jwt.verify(token, process.env.SECRET_KEY)

        const findCompany = await Company.findOne({_id:verifyToken._id,
"tokens.token":token})

        if(!findCompany){
            throw new Error("Login Expired")
        }

        if(role!=="company"){
            res.status(401).json({msg:'Unauthorized access'})
        }

        req.token=token
        req.findCompany=findCompany
        req.userID=findCompany._id
        next()
    } catch (error) {
        res.status(401).json({msg:'Unauthorized access'})
    }
}

module.exports = companyAuthenticate
```

Creation of JWT token

1. **Vendor**

```
vendorSchema.methods.generateAuthToken = async function(){
    try {
        const token = jwt.sign({_id: this._id}, process.env.SECRET_KEY);
        this.tokens = this.tokens.concat({token: token});
        await this.save();
        return token;
    } catch (error) {
        console.log(error)
    }
}
```

2. **Company**

```
companySchema.methods.generateAuthToken = async function(){
    try {
        const token = jwt.sign({_id: this._id}, process.env.SECRET_KEY);
        this.tokens = this.tokens.concat({token: token});
        await this.save();
        return token;
    } catch (error) {
        console.log(error)
    }
}
```

Conclusion:

JSON Web Tokens (JWTs) provide a secure and efficient mechanism for authentication in web applications. By utilizing JWTs, developers can implement robust authentication protocols while maintaining a stateless design. The self-contained nature of JWTs eliminates the need for constant server-side calls, reducing server load and improving performance. Additionally, the use of cryptographic signatures ensures the integrity and authenticity of the information contained within the tokens.