Softwarica
College of IT & E-commerce

Security implementation on FALCOM
ST6005CEM Security

Submitted By

Rushmit Karki

Coventry Id:12980532

Module Leader

Arya Pokharel

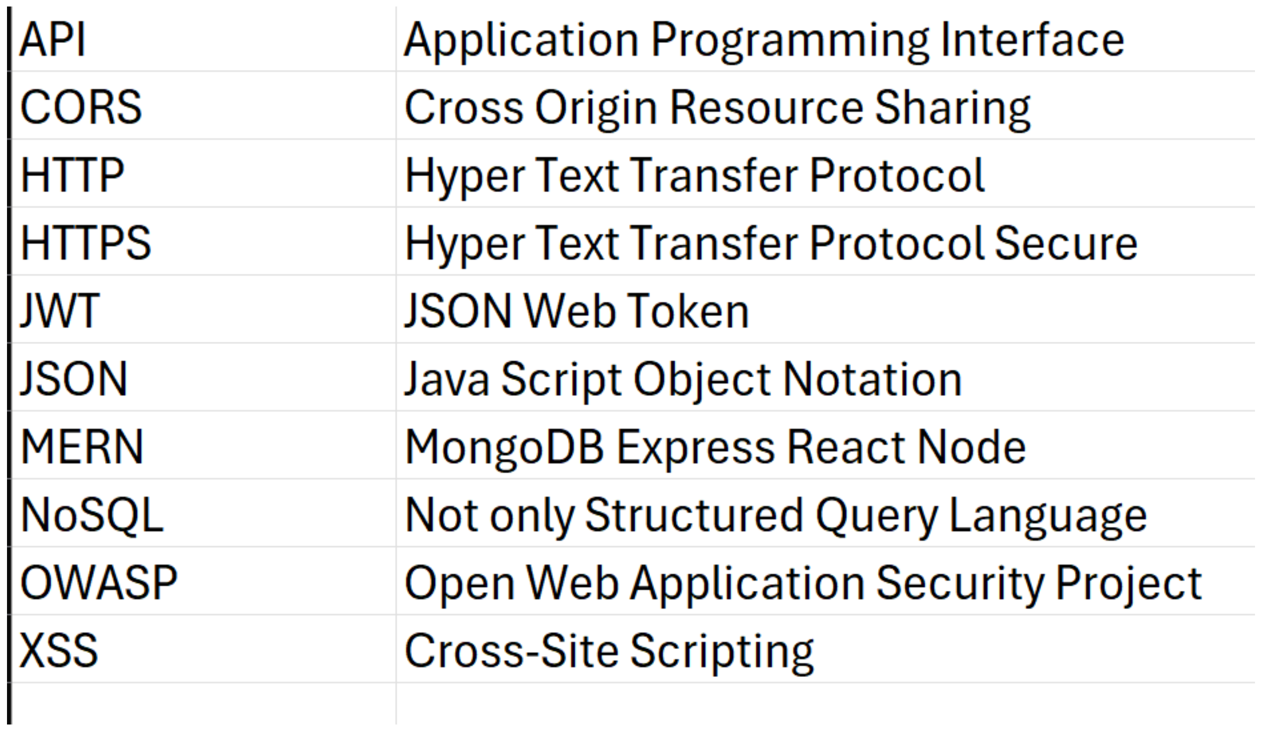
ABSTRACT

Falcom utilizes a client-server design, mixing client-side and server-side aspects to create a seamless and secure user experience. The front end, developed using React, provides a straightforward and responsive layout that adjusts to multiple devices, ensuring accessibility for all users. The backend, constructed with Node.js and Express, provides extensive server-side capabilities. Advanced security methods, such as HTTPS, JWT for secure authentication, and data encryption, are implemented to offer optimum protection against potential threats. Falcom is designed to provide high performance and scalability, allowing future expansion to meet expanding user demands. The website caters to a varied clientele, offering a complete assortment of automotive products such as bike, automobile, and truck tires, tubes, tire fluid, and accessories. Administrators control the system's correctness and security, merchants update product data, and consumers explore the platform to find solutions tailored to their needs.

TABLE OF CONTENTS

Table Of Figure

TABLE OF ABBREVIATIONS



API	Application Programming Interface
CORS	Cross Origin Resource Sharing
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
JWT	JSON Web Token
JSON	Java Script Object Notation
MERN	MongoDB Express React Node
NoSQL	Not only Structured Query Language
OWASP	Open Web Application Security Project
XSS	Cross-Site Scripting

INTRODUCTION

Falcom is an innovative platform dedicated to providing high-quality automotive products and services that cater to drivers from all walks of life. Designed with seamless navigation, robust security measures, and user-friendly search and filtering options, Falcom makes it easy to locate the appropriate automobile solutions for your needs. More than just a store, Falcom delivers thorough insights into the reliability, safety, and performance of its products, helping customers to make informed selections. With options such as premium tires, lubricants, and accessories obtained from industry-leading suppliers, Falcom ensures longevity, safety, and better vehicle performance.

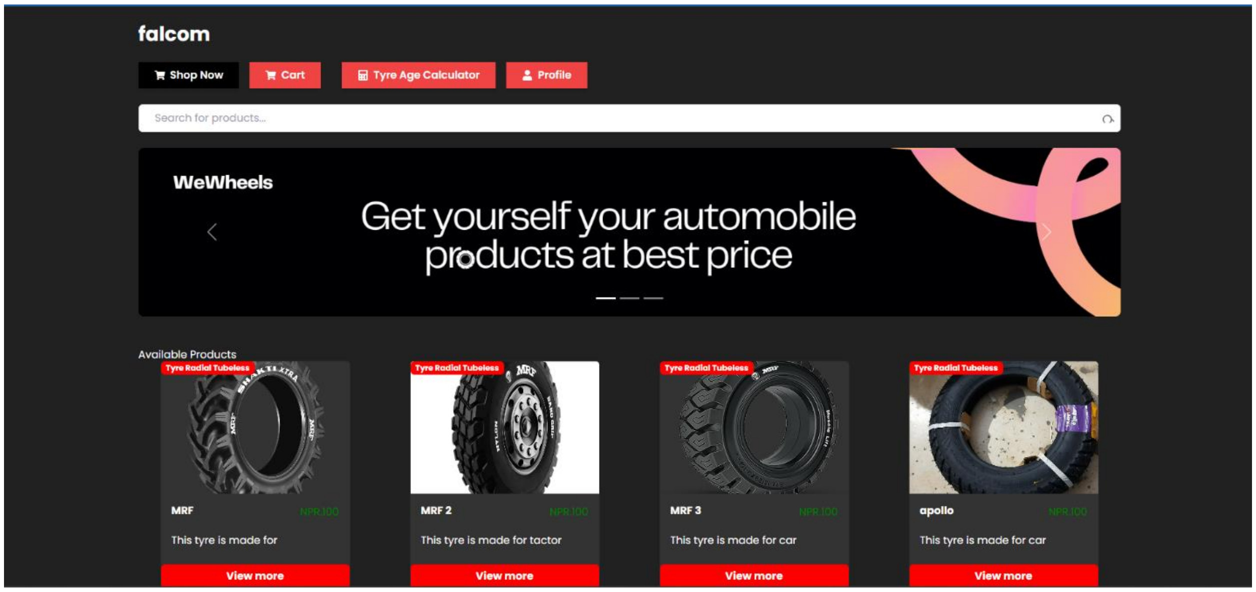


Figure 1:HomePage of the System

The platform targets a diverse audience, including car enthusiasts, professional drivers, businesses, and new drivers, offering tailored expert guidance for every customer. Falcom also prioritizes the security of its users by implementing advanced data encryption, secure payment gateways, and regular security audits to protect customer information and ensure a safe shopping experience. Additionally, Falcom fosters a sense of community by building long-lasting relationships with its customers based on trust, transparency, and shared passion for the road. Falcom is the ideal destination for anyone seeking to upgrade their driving experience, enhance vehicle performance, or stay connected with the latest automotive trends. By choosing Falcom, customers become part of a dedicated network that prioritizes quality, customer satisfaction, and innovation, ensuring that every journey is safe, efficient, and memorable. With a commitment to excellence and a focus on security, Falcom not only meets but exceeds the expectations of modern drivers, providing peace of mind alongside top-tier automotive solutions. Whether you're looking for performance upgrades, maintenance essentials, or the latest in automotive technology, Falcom is your trusted partner on the road ahead.

SOFTWARE DETAILS

The MERN stack is a robust and efficient web development stack combining four key technologies: MongoDB, Express.js, React, and Node.js. This stack allows developers to create full-stack JavaScript applications, with both the frontend and backend written in JavaScript (*MERN Stack Explained*, n.d.). Here's a detailed breakdown of each component of the MERN stack for developing Falcom:

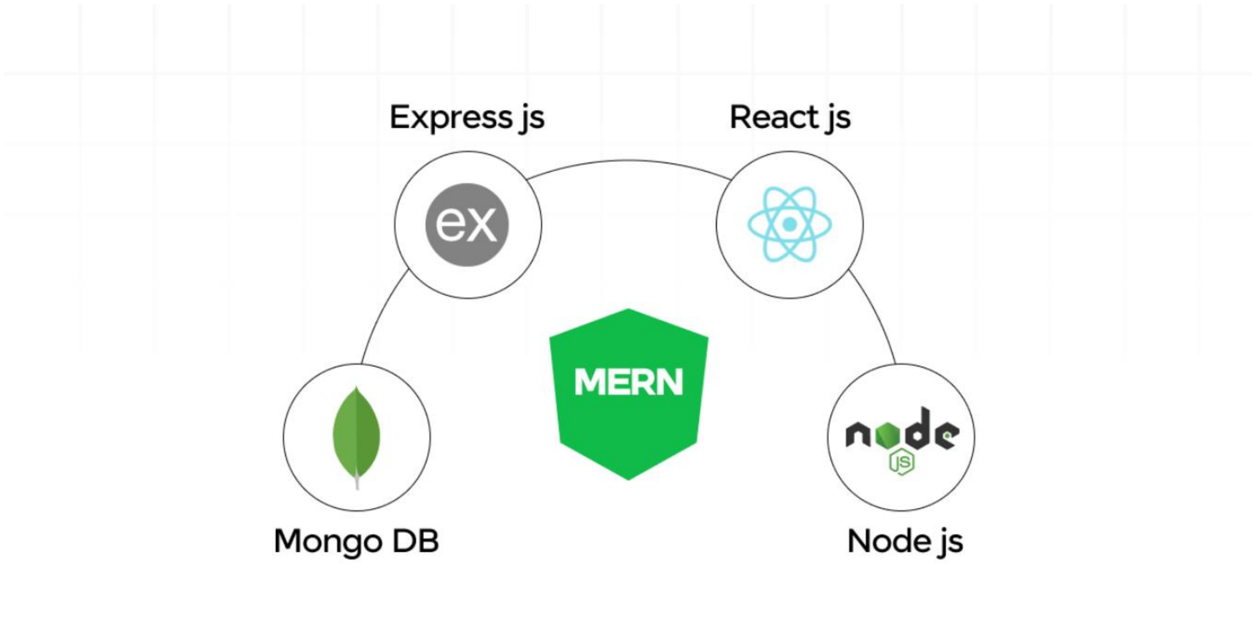


Figure 2: Framework used

MONGODB

- **Type**: NoSQL database
- **Role**: Database layer

  MongoDB provides a scalable and flexible database solution for Falcom, capable of storing diverse data types such as product catalogs, customer information, and transaction records. Its schema-less nature supports rapid iteration, allowing for easy updates and changes to the data model as the application grows. MongoDB's high-performance read and write operations ensure efficient data handling, making it ideal for a dynamic e-commerce platform like Falcom.

EXPRESS.JS

- **Type**: Web application framework
- **Role**: Server-side framework

  Express.js simplifies backend development for Falcom by providing a robust set of tools for creating RESTful APIs. It enables efficient routing, middleware integration, and seamless handling of HTTP requests and responses. Its unopinionated nature gives developers the flexibility to design the backend architecture tailored to the platform's specific needs, such as secure user authentication, product management, and order processing.

REACT

- **Type**: Frontend library
- **Role**: Client-side framework

React is used to build Falcon's interactive and responsive user interface. It allows the creation of reusable components for features like product displays, search filters, and customer dashboards. With its virtual DOM, React ensures high performance and smooth user interactions, even in data-intensive scenarios. Its component-based architecture promotes modularity and reusability, ensuring a maintainable and scalable frontend.

NODE.JS

- **Type**: JavaScript runtime
- **Role**: Server-side runtime environment

Node.js serves as the backbone of the backend, enabling real-time data processing and handling multiple concurrent user requests. Its event-driven, non-blocking I/O model ensures lightweight and efficient server operations. For Falcom, Node.js is essential for managing data transactions, integrating payment gateways, and supporting features like real-time stock updates and notifications.

SSL SECURITY

The process of security on a website comprises of a series of critical processes, including deployment of specialized tools to prevent the website from growing susceptible. First, one should substitute HTTP with HTTPS to guarantee that the information sent between the client and the server is encrypted and not exposed to interception.

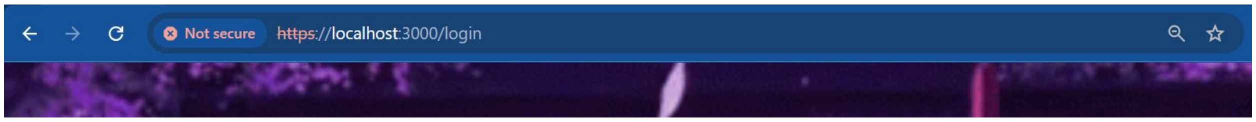
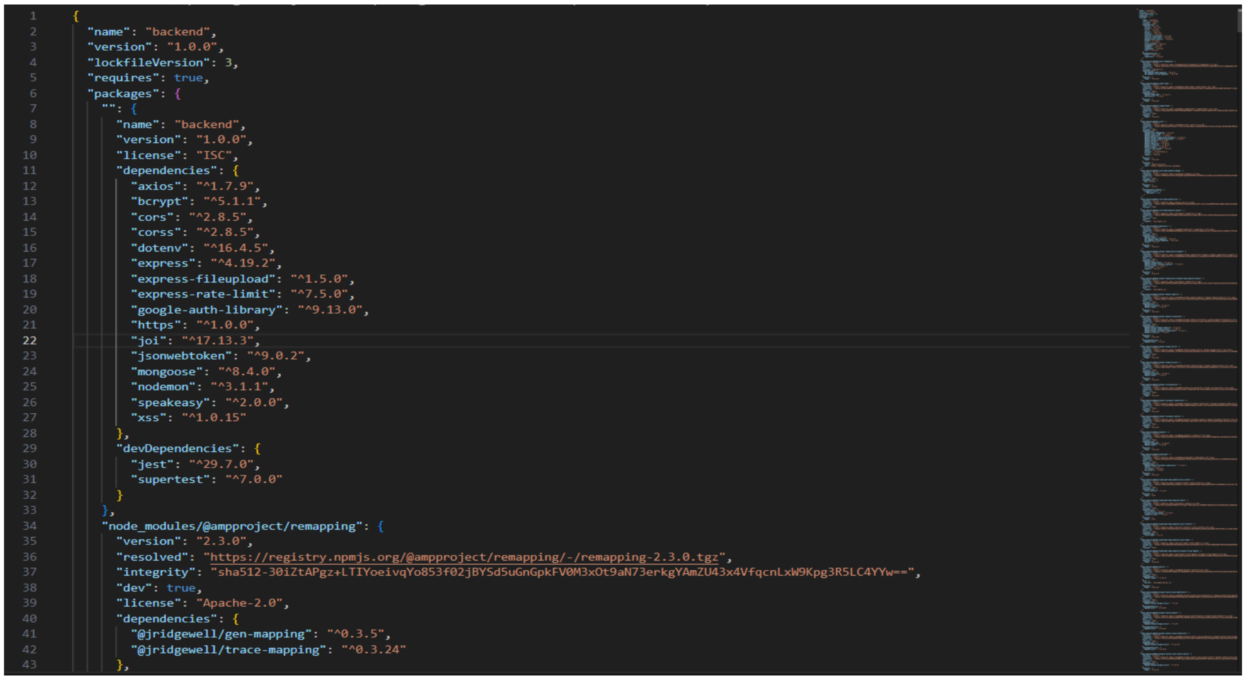


Figure 3: Server Started on HTTPS

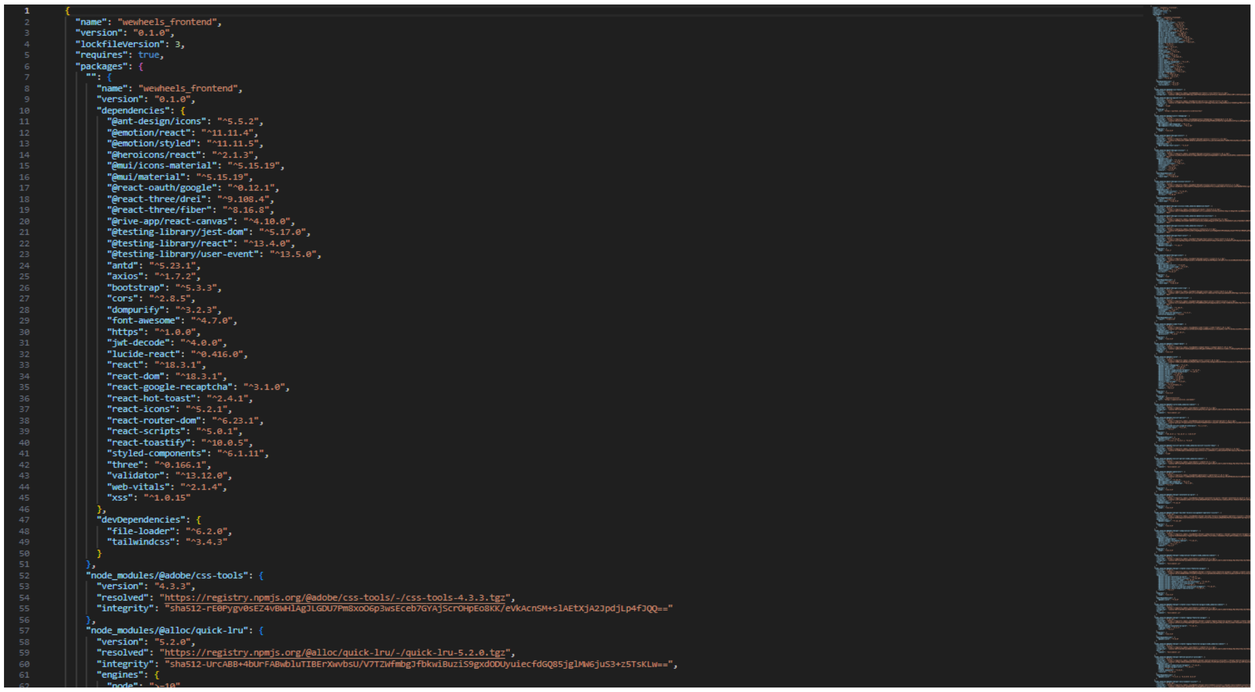
Generated SSL certificated was implemented to the system to secure more.

PACKAGES USED



```
{
  "name": "backend",
  "version": "1.0.0",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "name": "backend",
      "version": "1.0.0",
      "license": "ISC",
      "dependencies": {
        "axios": "^1.7.9",
        "bcrypt": "^5.1.1",
        "cors": "^2.8.5",
        "corss": "^2.8.5",
        "dotenv": "^16.4.5",
        "express": "^4.19.2",
        "express-fileupload": "^1.5.0",
        "express-rate-limit": "^7.5.0",
        "google-auth-library": "^9.13.0",
        "https": "^1.0.0",
        "joi": "^17.13.3",
        "jsonwebtoken": "^9.0.2",
        "mongoose": "^8.4.0",
        "nodemon": "^3.1.1",
        "speakeasy": "^2.0.0",
        "xss": "^1.0.15"
      },
      "devDependencies": {
        "jest": "^29.7.0",
        "supertest": "^7.0.0"
      }
    },
    "node_modules/@ampproject/remapping": {
      "version": "2.3.0",
      "resolved": "https://registry.npmjs.org/@ampproject/remapping/-/remapping-2.3.0.tgz",
      "integrity": "sha512-30iZtAPgz+LTIYoeivqYo853f02jBYSd5uGnGpkFV0M3xOt9aN73erkgYAmZU43x4VfqcnLxW9Kpg3R5LC4YYw==",
      "dev": true,
      "license": "Apache-2.0",
      "dependencies": {
        "@jridgewell/gen-mapping": "^0.3.5",
        "@jridgewell/trace-mapping": "^0.3.24"
      },
```

Figure 4: Backend Package



```
{
  "name": "wewheels_frontend",
  "version": "0.1.0",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "name": "wewheels_frontend",
      "version": "0.1.0",
      "dependencies": {
        "@ant-design/icons": "^5.5.2",
        "@emotion/react": "^11.11.4",
        "@emotion/styled": "^11.11.5",
        "@heroicons/react": "^2.1.3",
        "@mui/icons-material": "^5.15.19",
        "@mui/material": "^5.15.19",
        "@react-oauth/google": "^0.12.1",
        "@react-three/drei": "^9.108.4",
        "@react-three/fiber": "^8.16.8",
        "@rive-app/react-canvas": "^4.10.0",
        "@testing-library/jest-dom": "^5.17.0",
        "@testing-library/react": "^13.4.0",
        "@testing-library/user-event": "^13.5.0",
        "antd": "^5.23.1",
        "axios": "^1.7.2",
        "bootstrap": "^5.3.3",
        "cors": "^2.8.5",
        "dompurify": "^3.2.3",
        "font-awesome": "^4.7.0",
        "https": "^1.0.0",
        "jwt-decode": "^4.0.0",
        "lucide-react": "^0.416.0",
        "react": "^18.3.1",
        "react-dom": "^18.3.1",
        "react-google-recaptcha": "^3.1.0",
        "react-hot-toast": "^2.4.1",
        "react-icons": "^5.2.1",
        "react-router-dom": "^6.23.1",
        "react-scripts": "^5.0.1",
        "react-toastify": "^10.0.5",
        "styled-components": "^6.1.11",
        "three": "^0.166.1",
        "validator": "^13.12.0",
        "web-vitals": "^2.1.4",
        "xss": "^1.0.15"
      },
      "devDependencies": {
        "file-loader": "^6.2.0",
        "tailwindcss": "^3.4.3"
      }
    },
    "node_modules/@adobe/css-tools": {
      "version": "4.3.3",
      "resolved": "https://registry.npmjs.org/@adobe/css-tools/-/css-tools-4.3.3.tgz",
      "integrity": "sha512-rE0Pygv0sEZ4vBWHlAgJLGDU7Pm8xoO6p3wsEceb7GYAjScrOHpEo8KK/eVkAcnSM+slAEtXjA2JpdjLp4fJQQ=="
    },
    "node_modules/@alloc/quick-lru": {
      "version": "5.2.0",
      "resolved": "https://registry.npmjs.org/@alloc/quick-lru/-/quick-lru-5.2.0.tgz",
      "integrity": "sha512-UrcABB+4bUrFABwbluTIBErXwvbsU/V7TZWfmbgJfbkwiBuziS9gxdODUyuiecfdGQ85jglMW6juS3+z5TsKLw==",
      "engines": {
        "node": ">=10"
```

Figure 5: Frontend Package

STRATEGIC DESIGN PRINCIPLES OF BUSINESS DEVELOPMENT

USER EXPERIENCE DESIGN

After careful research of the elements of the planned product, real users' habits and preferences, a good-looking and easy to navigate interface was built. Convenience is provided with proper layout of the design because it assists the users to conveniently and consecutively search for the best tyre. Appreciation of the employment of graphic elements and the synchronized colors improves the general outlook and makes it engaging for all the users (Stevens, 2025).

SCALABLE ARCHITECTURE

When it came to technological architecture, scalability was vital. It meets growth qualities in so far as future enlargement in terms of user base and functions is concerned. This was done in a procedural manner followed in React, by coding in modules, so each part could be adjusted independently of the others. In addition, this form of architecture significantly accelerates the velocity of development and can is more readily maintainable.

PERFORMANCE OPTIMIZATION

Specific software-related goals were to boost sites' performance by reducing loading times, as well as organizing data effectively. Features like lazy loading and effective calls to the API were employed. This still is optimized for request processing on the back end, which is necessary for real-time additions such as business data and verification status.

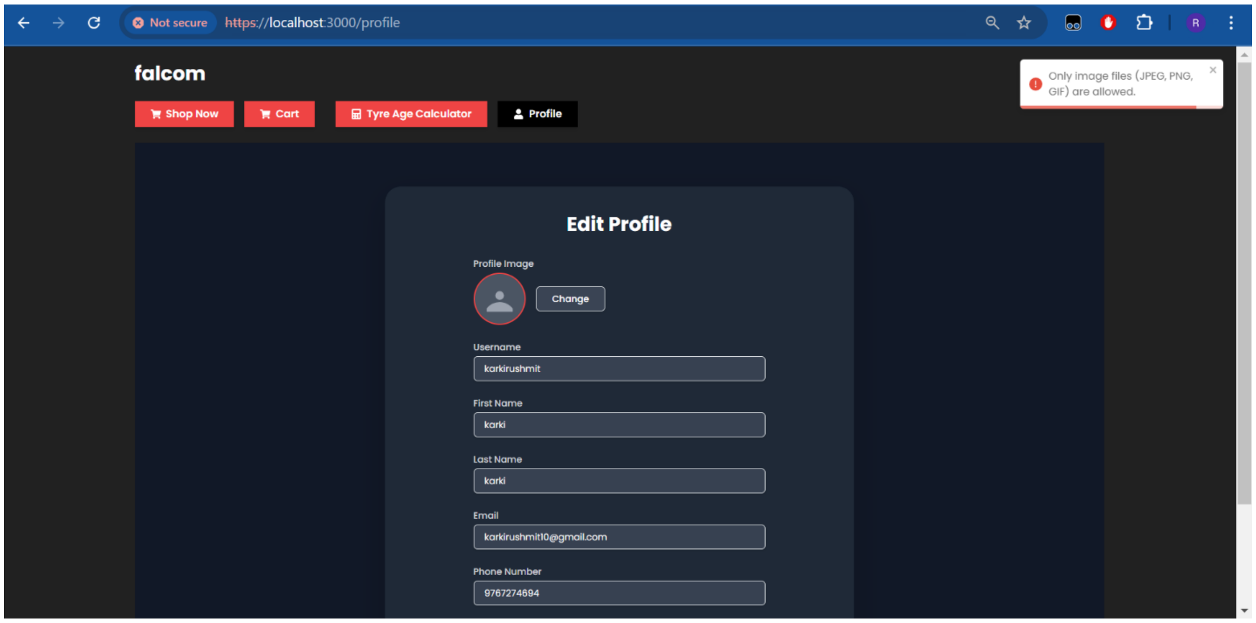
SECURITY INTEGRATION

Since its creation, there has been an emphasis on the topic of security. This means that while creating the system architectural framework, one had to add robust security technologies like HTTPS, JWT, and Bcrypt amongst others. More so, the division of the presentation layer from the business layer aids in the prevention of hazards (Moteria, 2024).

SECURITY IMPLEMENTED BY DESIGN PRINCIPLE

The security of Falcom system architecture is reinforced by adhering to several OWASP (Open Web Application Security Project) design principles, ensuring robust protection against common vulnerabilities and threats.

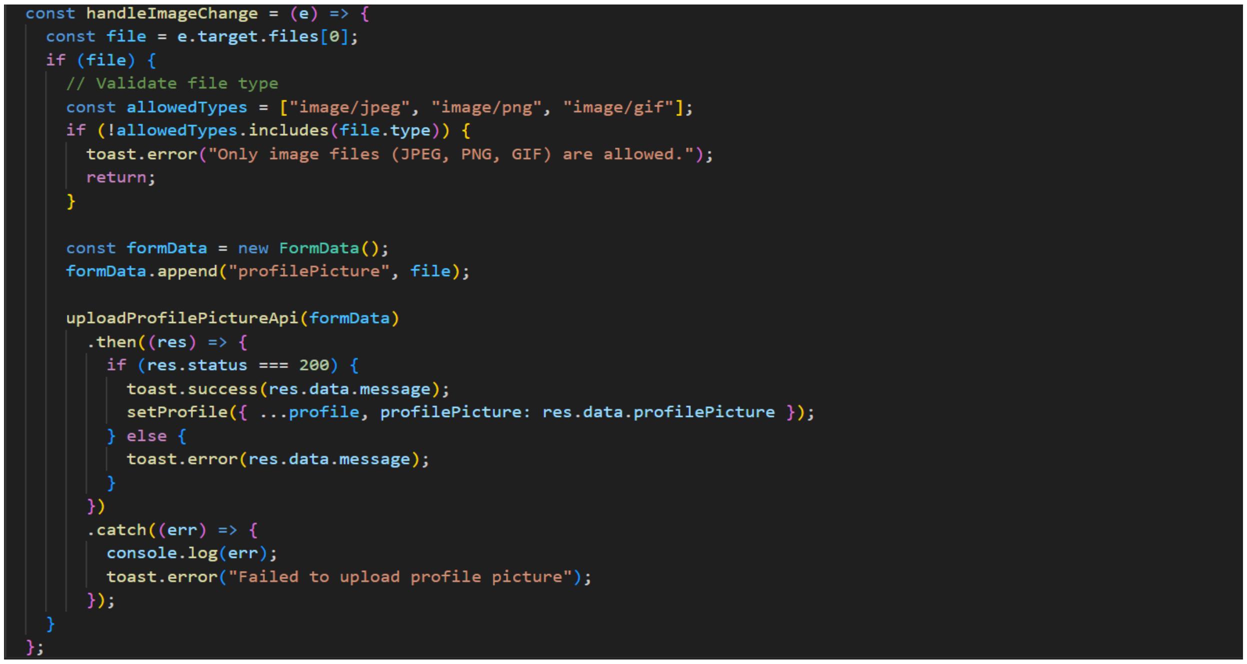
PREVENT FILE UPLOAD VULNERABILITY



Only image files (JPEG, PNG, GIF) are allowed.

Username: karkirushmit
First Name: karki
Last Name: karki
Email: karkirushmit10@gmail.com
Phone Number: 9767274694

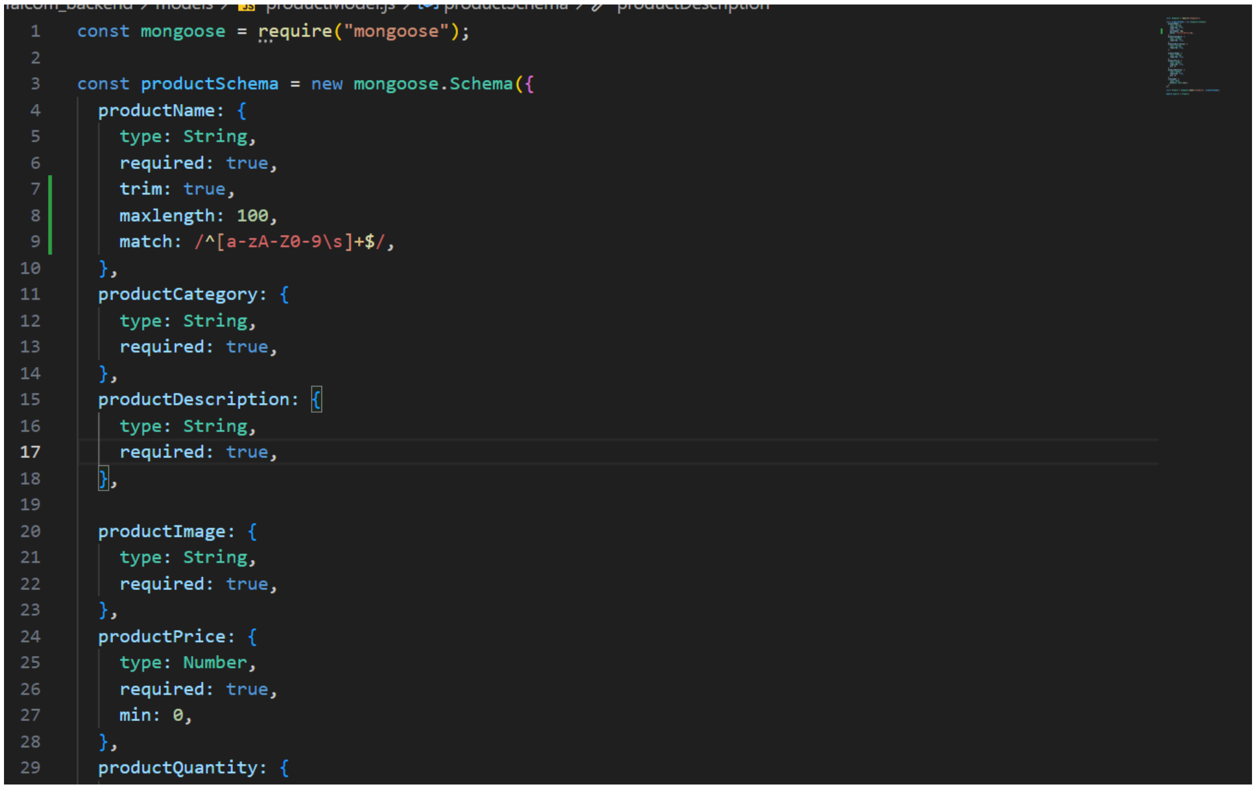
Figure 6: Prevent File Upload Vulnerability



```
const handleImageChange = (e) => {
  const file = e.target.files[0];
  if (file) {
    // Validate file type
    const allowedTypes = ["image/jpeg", "image/png", "image/gif"];
    if (!allowedTypes.includes(file.type)) {
      toast.error("Only image files (JPEG, PNG, GIF) are allowed.");
      return;
    }

    const formData = new FormData();
    formData.append("profilePicture", file);

    uploadProfilePictureApi(formData)
      .then((res) => {
        if (res.status === 200) {
          toast.success(res.data.message);
          setProfile({ ...profile, profilePicture: res.data.profilePicture });
        } else {
          toast.error(res.data.message);
        }
      })
      .catch((err) => {
        console.log(err);
        toast.error("Failed to upload profile picture");
      });
  }
};
```

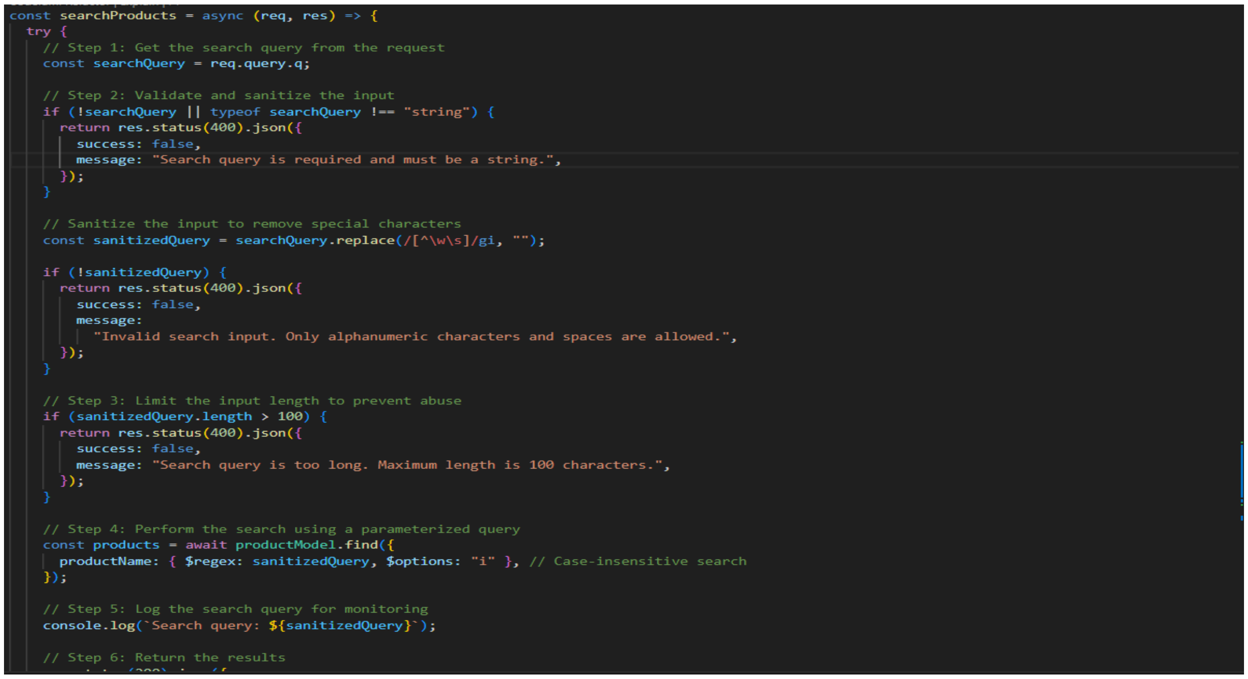
Figure 7: Implementation of File Upload

PREVENTING COMMAND INJECTION IN SEARCH BAR



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

const productSchema = new mongoose.Schema({
  productName: {
    type: String,
    required: true,
    trim: true,
    maxlength: 100,
    match: /^[a-zA-Z0-9\s]+$/,
  },
  productCategory: {
    type: String,
    required: true,
  },
  productDescription: {
    type: String,
    required: true,
  },

  productImage: {
    type: String,
    required: true,
  },
  productPrice: {
    type: Number,
    required: true,
    min: 0,
  },
  productQuantity: {
```

Figure 8: Database Validation



```
const searchProducts = async (req, res) => {
  try {
    // Step 1: Get the search query from the request
    const searchQuery = req.query.q;

    // Step 2: Validate and sanitize the input
    if (!searchQuery || typeof searchQuery !== "string") {
      return res.status(400).json({
        success: false,
        message: "Search query is required and must be a string.",
      });
    }

    // Sanitize the input to remove special characters
    const sanitizedQuery = searchQuery.replace(/[^\w\s]/gi, "");

    if (!sanitizedQuery) {
      return res.status(400).json({
        success: false,
        message:
          "Invalid search input. Only alphanumeric characters and spaces are allowed.",
      });
    }

    // Step 3: Limit the input length to prevent abuse
    if (sanitizedQuery.length > 100) {
      return res.status(400).json({
        success: false,
        message: "Search query is too long. Maximum length is 100 characters.",
      });
    }

    // Step 4: Perform the search using a parameterized query
    const products = await productModel.find({
      productName: { $regex: sanitizedQuery, $options: "i" }, // Case-insensitive search
    });

    // Step 5: Log the search query for monitoring
    console.log(`Search query: ${sanitizedQuery}`);

    // Step 6: Return the results
```

Figure 9: Server-Side logic in search

PRINCIPLE OF LEAST PRIVILEGE

Access constraints are tightly enforced. Users are provided only the rights necessary to perform their roles. Admins have verification and management privileges, firm owners can manage their profiles, and typical users have read-only access to business information.

SECURE DATA TRANSMISSION

HTTPS is used to encrypt data exchanged between the client and server, guarding against eavesdropping and man-in-the-middle attacks. This ensures that all data, including user credentials and company information, is securely delivered.

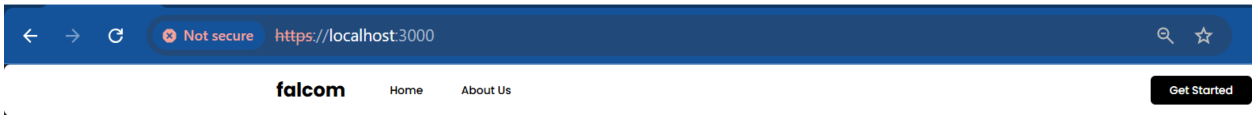
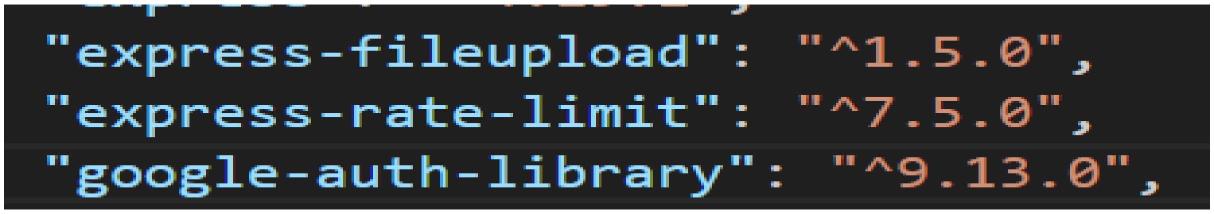


Figure 10: Https is enabled for server

RATE LIMITING AND THROTTLING

Express-Rate-Limit is implemented to restrict the number of requests a user may make during a specific timeframe. This helps reduce brute-force assaults and exploitation of the system. If the user makes the multiple requests from the same system, then there should be the capability of limiting the request to the server which eventually stops the server to have a multiple same request from the same system. For this "express rate-limit": "^7.5.0", is employed.

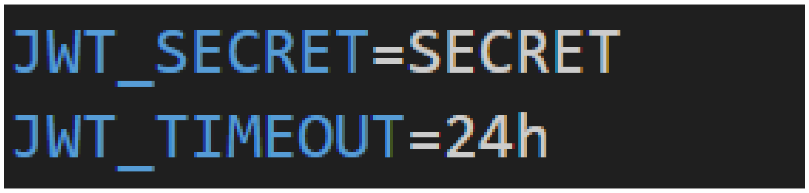


```
"express-fileupload": "^1.5.0",
"express-rate-limit": "^7.5.0",
"google-auth-library": "^9.13.0",
```

Figure 11: Rate Limit Package

AUTHENTICATION AND SESSION MANAGEMENT

JWT (JSON Web Tokens) is applied for secure authentication and session management. Tokens are securely produced, stored, and verified, guaranteeing that only authenticated users can access protected resources.

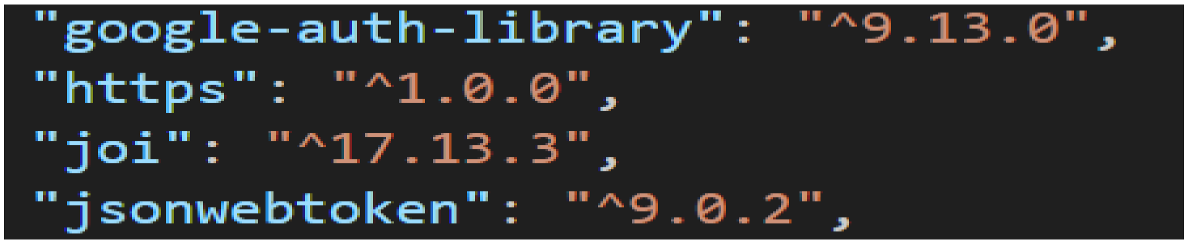


```
JWT_SECRET=SECRET
JWT_TIMEOUT=24h
```

Figure 12:Jwt Secret Key

INPUT VALIDATION

All user inputs undergo extensive validation and sanitization using tools like Express-Validator and Joi. This protects against typical attacks such as NoSQL injection, cross-site scripting (XSS), and other injection flaws. By utilizing these principles, Falcom offers a secure platform that efficiently protects user data and maintains the integrity of its operations. This focus to security by design helps in developing trust with users and provides a firm foundation for handling company information safely.



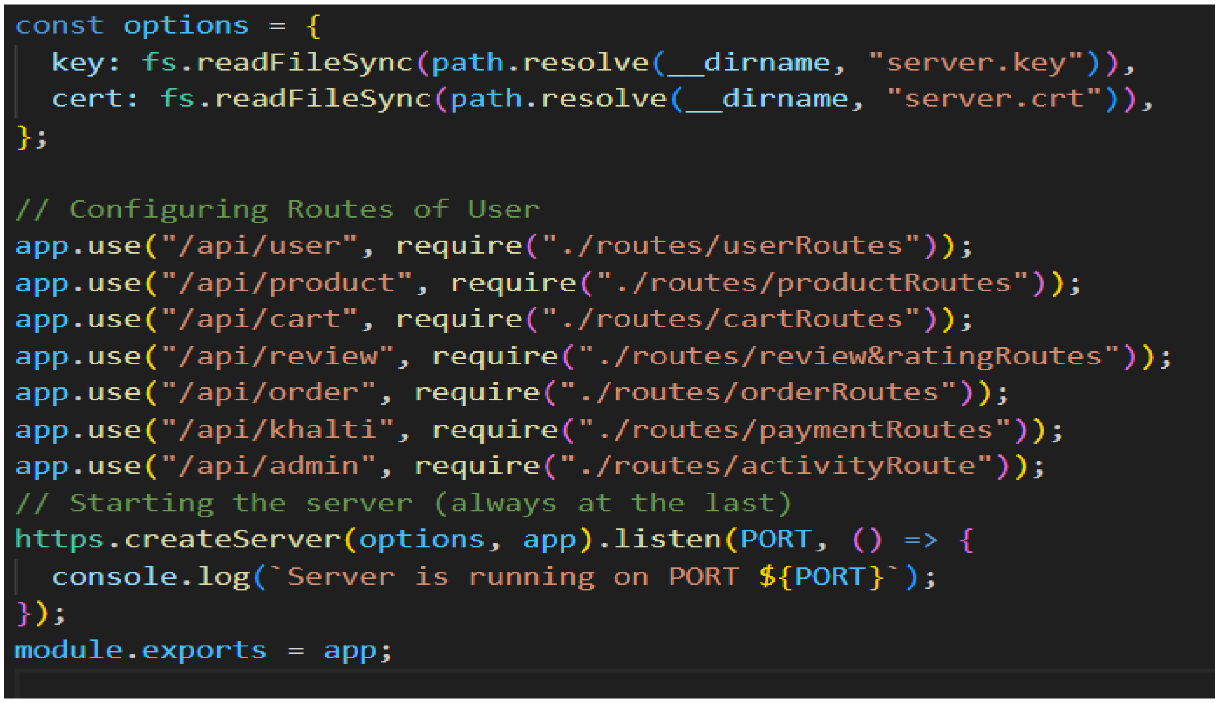
```
"google-auth-library": "^9.13.0",
"https": "^1.0.0",
"joi": "^17.13.3",
"jsonwebtoken": "^9.0.2",
```

Figure 13:Joi Package

IMPLEMENTATION OF SECURITY

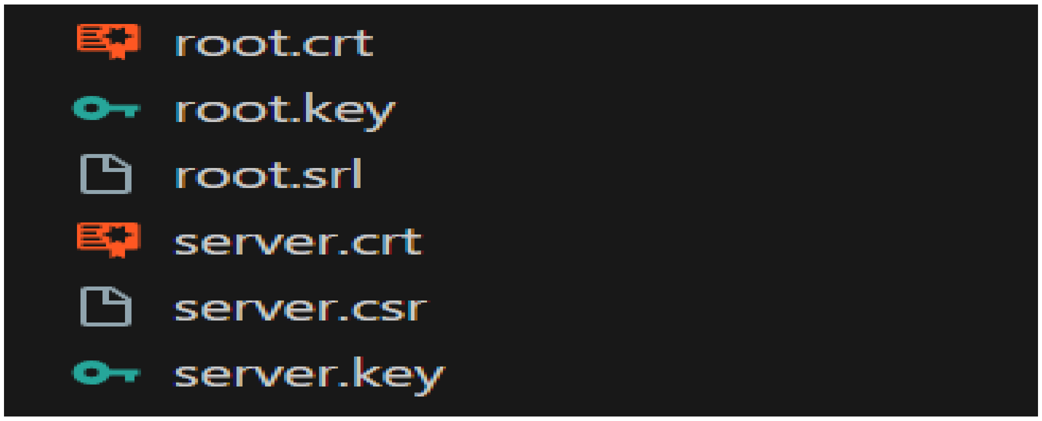
ADDED HTTPS

Transitioning from HTTP to HTTPS in the local environment was a key step, ensuring encrypted data transfer. This change enhanced the security of user transactions and data, simulating real-world secure transfer protocols and showcasing the platform's capability to handle secure transactions and protect user information *(What Is HTTPS? A Definition and How to Switch to HTTPS? | Fortinet*, n.d.).



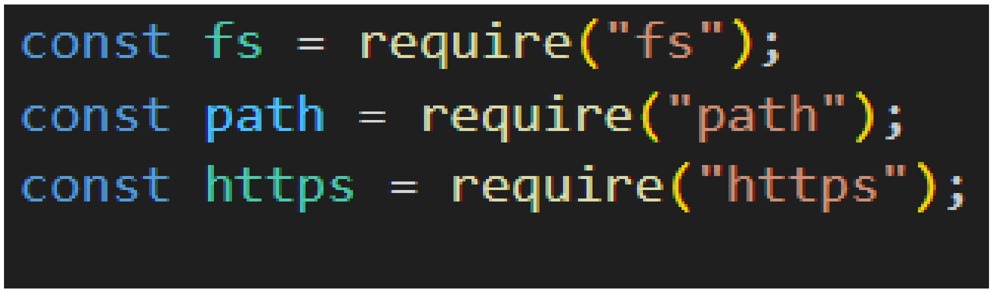
```
const options = {
  key: fs.readFileSync(path.resolve(__dirname, "server.key")),
  cert: fs.readFileSync(path.resolve(__dirname, "server.crt")),
};

// Configuring Routes of User
app.use("/api/user", require("./routes/userRoutes"));
app.use("/api/product", require("./routes/productRoutes"));
app.use("/api/cart", require("./routes/cartRoutes"));
app.use("/api/review", require("./routes/review&ratingRoutes"));
app.use("/api/order", require("./routes/orderRoutes"));
app.use("/api/khalti", require("./routes/paymentRoutes"));
app.use("/api/admin", require("./routes/activityRoute"));
// Starting the server (always at the last)
https.createServer(options, app).listen(PORT, () => {
  console.log(`Server is running on PORT ${PORT}`);
});
module.exports = app;
```

Figure 14: Server Site Https



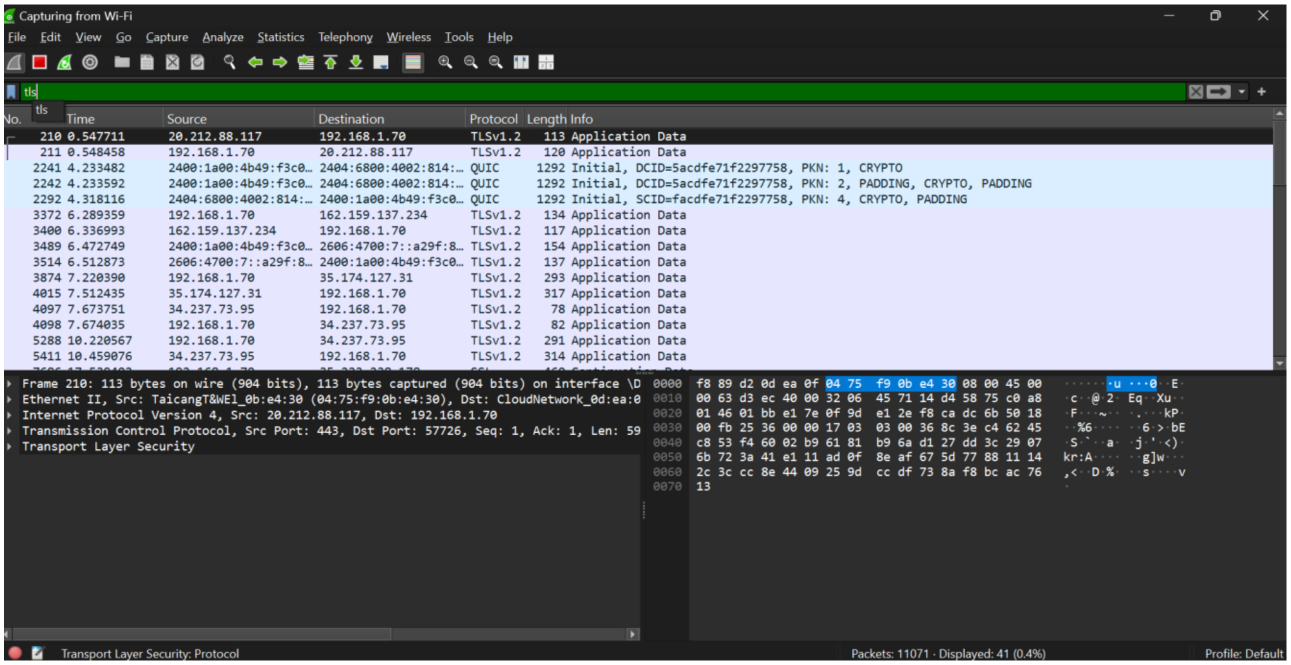
```
root.crt
root.key
root.srl
server.crt
server.csr
server.key
```

Figure 15: List of Certificate



```javascript
const fs = require("fs");
const path = require("path");
const https = require("https");
```

Figure 16: Importing package



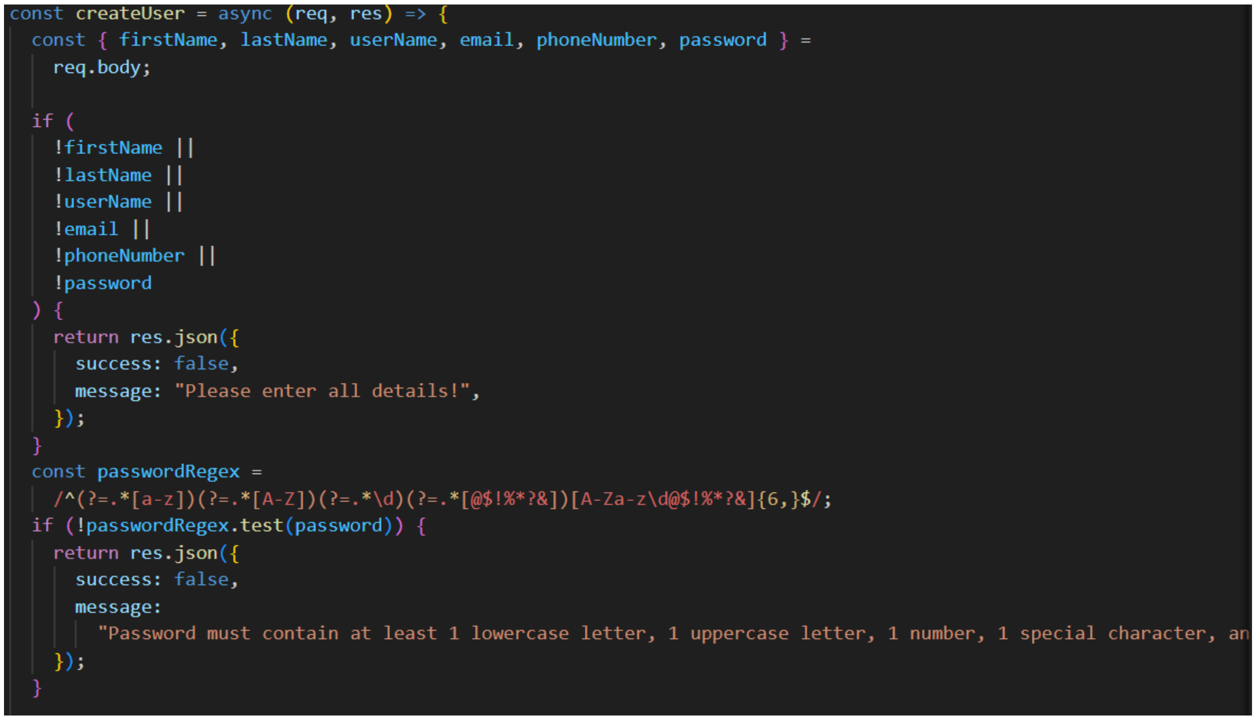
Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tls

```
No.   Time       Source                  Destination             Protocol Length Info
  210 0.547711   20.212.88.117           192.168.1.70            TLSv1.2   113 Application Data
  211 0.548458   192.168.1.70            20.212.88.117           TLSv1.2   120 Application Data
 2241 4.233482   2400:1a00:4b49:f3c0…    2404:6800:4002:814:…    QUIC     1292 Initial, DCID=5acdfe71f2297758, PKN: 1, CRYPTO
 2242 4.233592   2400:1a00:4b49:f3c0…    2404:6800:4002:814:…    QUIC     1292 Initial, DCID=5acdfe71f2297758, PKN: 2, PADDING, CRYPTO, PADDING
 2292 4.318116   2404:6800:4002:814:…    2400:1a00:4b49:f3c0…    QUIC     1292 Initial, SCID=facdfe71f2297758, PKN: 4, CRYPTO, PADDING
 3372 6.289359   192.168.1.70            162.159.137.234         TLSv1.2   134 Application Data
 3400 6.336993   162.159.137.234         192.168.1.70            TLSv1.2   117 Application Data
 3489 6.472749   2400:1a00:4b49:f3c0…    2606:4700:7::a29f:8…    TLSv1.2   154 Application Data
 3514 6.512873   2606:4700:7::a29f:8…    2400:1a00:4b49:f3c0…    TLSv1.2   137 Application Data
 3874 7.220390   192.168.1.70            35.174.127.31           TLSv1.2   293 Application Data
 4015 7.512435   35.174.127.31           192.168.1.70            TLSv1.2   317 Application Data
 4097 7.673751   34.237.73.95            192.168.1.70            TLSv1.2    78 Application Data
 4098 7.674035   192.168.1.70            34.237.73.95            TLSv1.2    82 Application Data
 5288 10.220567  192.168.1.70            34.237.73.95            TLSv1.2   291 Application Data
 5411 10.459076  34.237.73.95            192.168.1.70            TLSv1.2   314 Application Data

> Frame 210: 113 bytes on wire (904 bits), 113 bytes captured (904 bits) on interface \D
> Ethernet II, Src: TaicangT&WEl_0b:e4:30 (04:75:f9:0b:e4:30), Dst: CloudNetwork_0d:ea:0
> Internet Protocol Version 4, Src: 20.212.88.117, Dst: 192.168.1.70
> Transmission Control Protocol, Src Port: 443, Dst Port: 57726, Seq: 1, Ack: 1, Len: 59
> Transport Layer Security

0000  f8 89 d2 0d ea 0f 04 75  f9 0b e4 30 08 00 45 00   ······u···0··E·
0010  00 63 d3 ec 40 00 32 06  45 71 14 d4 58 75 c0 a8   ·c··@·2·Eq··Xu··
0020  01 46 01 bb e1 7e 0f 9d  e1 2e f8 ca dc 6b 50 18   ·F···~·····.··kP·
0030  00 fb 25 36 00 00 17 03  03 00 36 8c 3e c4 62 45   ··%6······6·>·bE
0040  c8 53 f4 60 02 b9 61 81  b9 6a d1 27 dd 3c 29 07   ·S·`··a··j·'·<)·
0050  6b 72 3a 41 e1 11 ad 0f  8e af 67 5d 77 88 11 14   kr:A······g]w···
0060  2c 3c cc 8e 44 09 25 9d  cc df 73 8a f8 bc ac 76   ,<··D·%···s····v
0070  13                                                 ·

Transport Layer Security: Protocol    Packets: 11071 · Displayed: 41 (0.4%)    Profile: Default
```

Figure 17: Verifying the data transfer with Wireshark

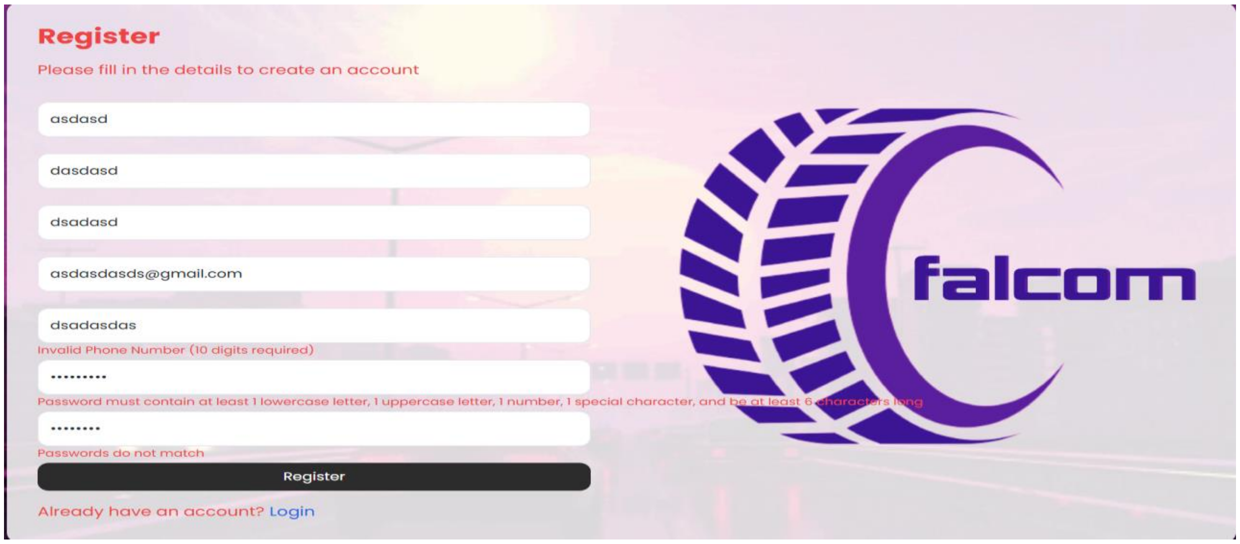
FORM VALIDATION

Every input undergoes validation to minimize human input errors. This guarantees that all data submitted by users is correct and consistent, lowering the likelihood of errors and boosting the overall reliability and security of the platform. To further strengthen security, data sanitization is conducted on all inputs to avoid harmful data and ensure clean, safe data handling.



```
const createUser = async (req, res) => {
  const { firstName, lastName, userName, email, phoneNumber, password } =
    req.body;

  if (
    !firstName ||
    !lastName ||
    !userName ||
    !email ||
    !phoneNumber ||
    !password
  ) {
    return res.json({
      success: false,
      message: "Please enter all details!",
    });
  }
  const passwordRegex =
    /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{6,}$/;
  if (!passwordRegex.test(password)) {
    return res.json({
      success: false,
      message:
        "Password must contain at least 1 lowercase letter, 1 uppercase letter, 1 number, 1 special character, an
    });
  }
```

Figure 18: Server-Side input validation



Register

Please fill in the details to create an account

asdasd

dasdasd

dsadasd

asdasdasds@gmail.com

dsadasdas

Invalid Phone Number (10 digits required)

•••••••••

Password must contain at least 1 lowercase letter, 1 uppercase letter, 1 number, 1 special character, and be at least 6 characters long

••••••••

Passwords do not match

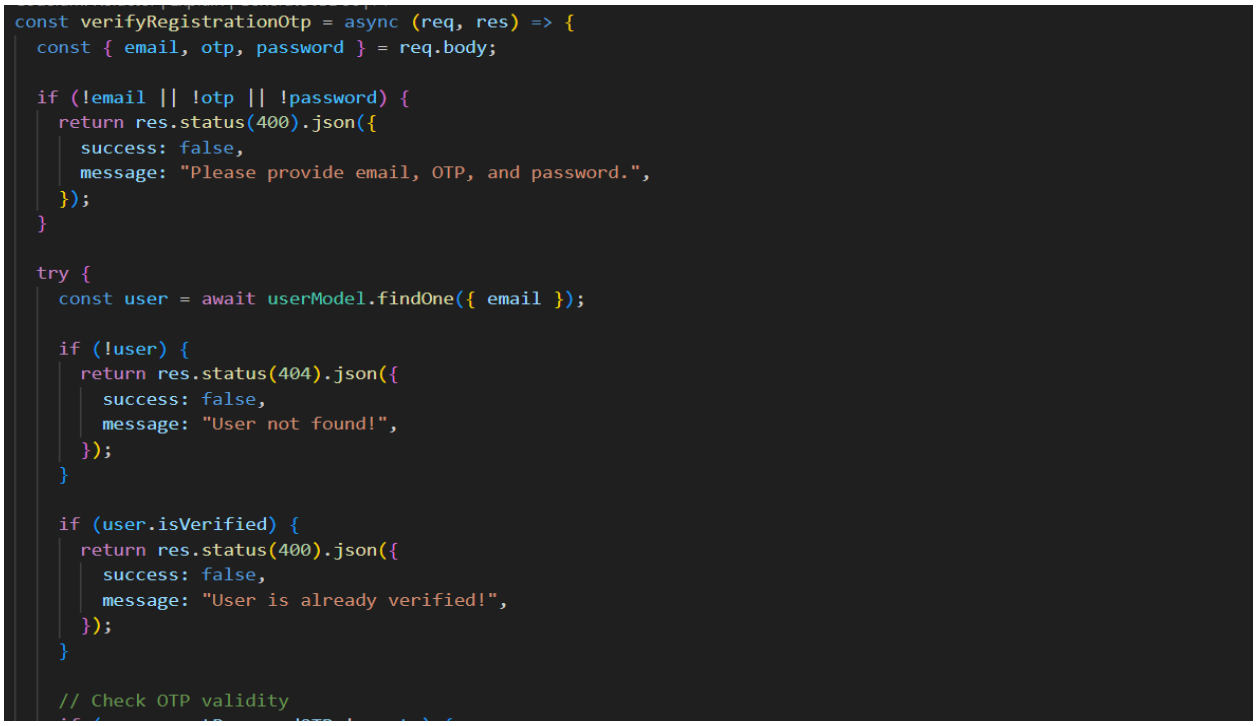
Register

Already have an account? Login

Figure 19: Client-Side validation

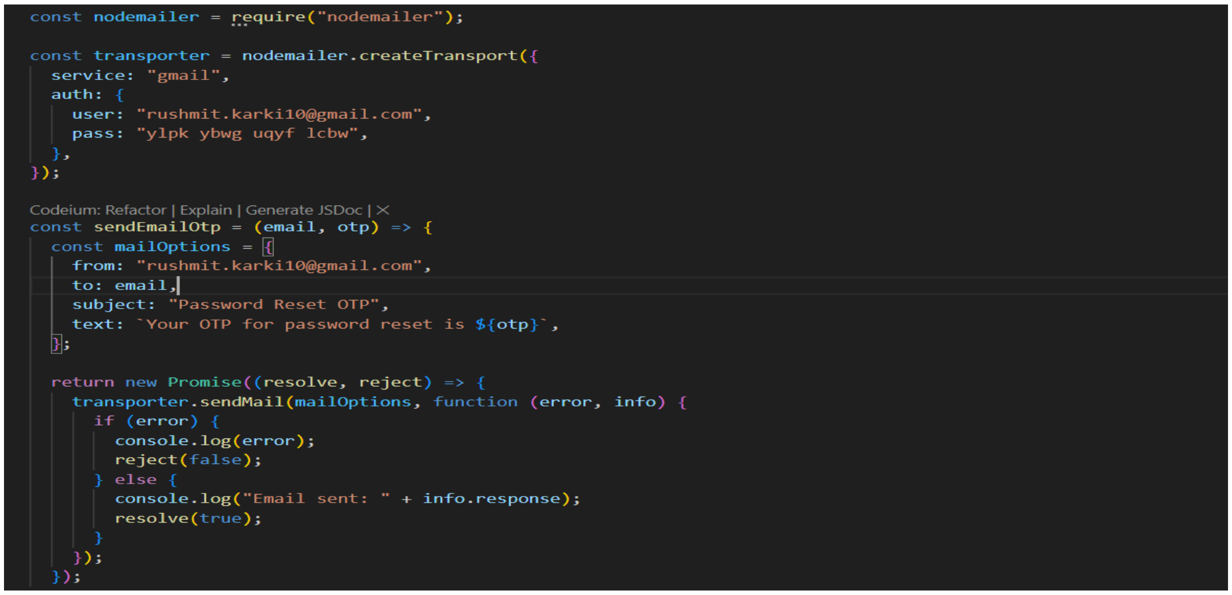
OTP VERIFICATION TO CREATE USER ACCOUNT

Additionally, user accounts are generated only after verification. The verification procedure occurs immediately after the user data is saved in an unverified state, ensuring that only verified users may use the site. After saving the user info the OTP is sent to the email address and if the OTP is verified then the user is verified and can only login to the system.

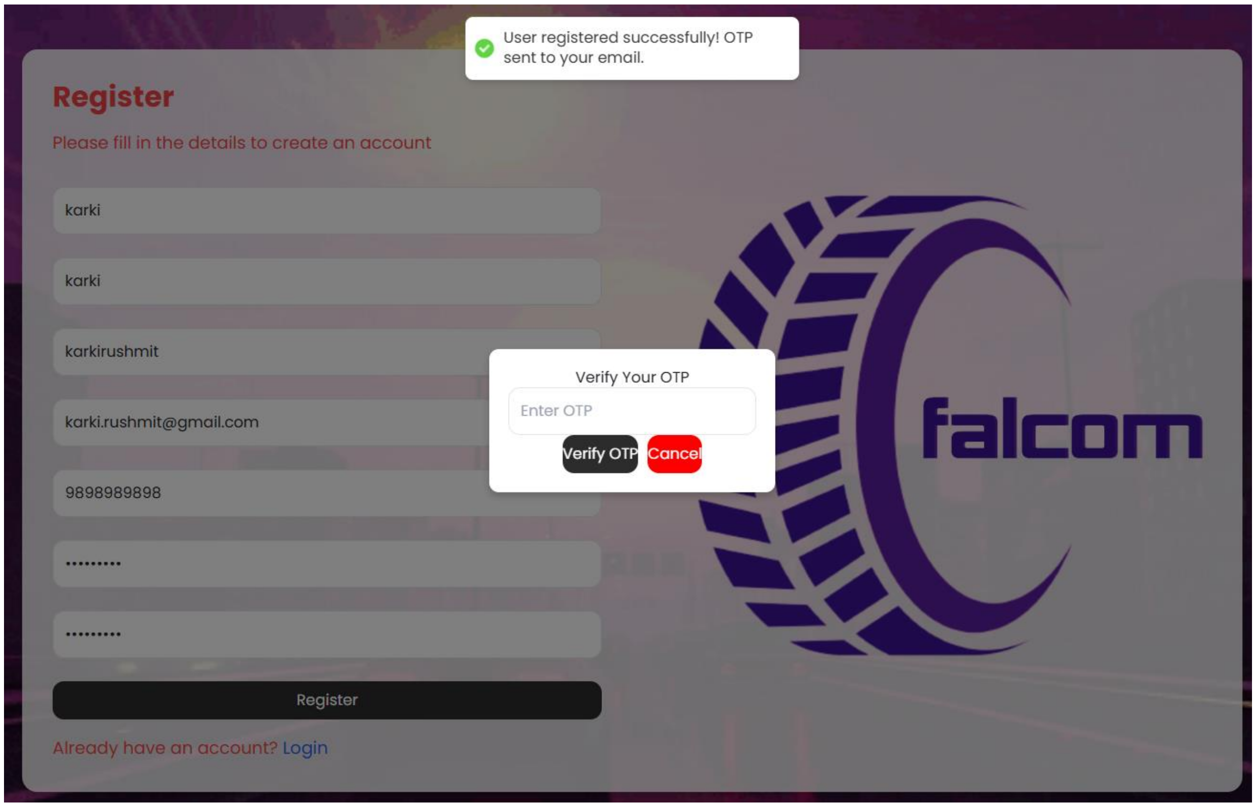


```
const verifyRegistrationOtp = async (req, res) => {
  const { email, otp, password } = req.body;

  if (!email || !otp || !password) {
    return res.status(400).json({
      success: false,
      message: "Please provide email, OTP, and password.",
    });
  }

  try {
    const user = await userModel.findOne({ email });

    if (!user) {
      return res.status(404).json({
        success: false,
        message: "User not found!",
      });
    }

    if (user.isVerified) {
      return res.status(400).json({
        success: false,
        message: "User is already verified!",
      });
    }

    // Check OTP validity
```

Figure 20: Server-Side OTP Validation



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

const transporter = nodemailer.createTransport({
  service: "gmail",
  auth: {
    user: "rushmit.karki10@gmail.com",
    pass: "ylpk ybwg uqyf lcbw",
  },
});

Codeium: Refactor | Explain | Generate JSDoc | ×
const sendEmailOtp = (email, otp) => {
  const mailOptions = {
    from: "rushmit.karki10@gmail.com",
    to: email,
    subject: "Password Reset OTP",
    text: `Your OTP for password reset is ${otp}`,
  };

  return new Promise((resolve, reject) => {
    transporter.sendMail(mailOptions, function (error, info) {
      if (error) {
        console.log(error);
        reject(false);
      } else {
        console.log("Email sent: " + info.response);
        resolve(true);
      }
    });
  });
```



User registered successfully! OTP sent to your email.

Register

Please fill in the details to create an account

karki

karki

karkirushmit

karki.rushmit@gmail.com

9898989898

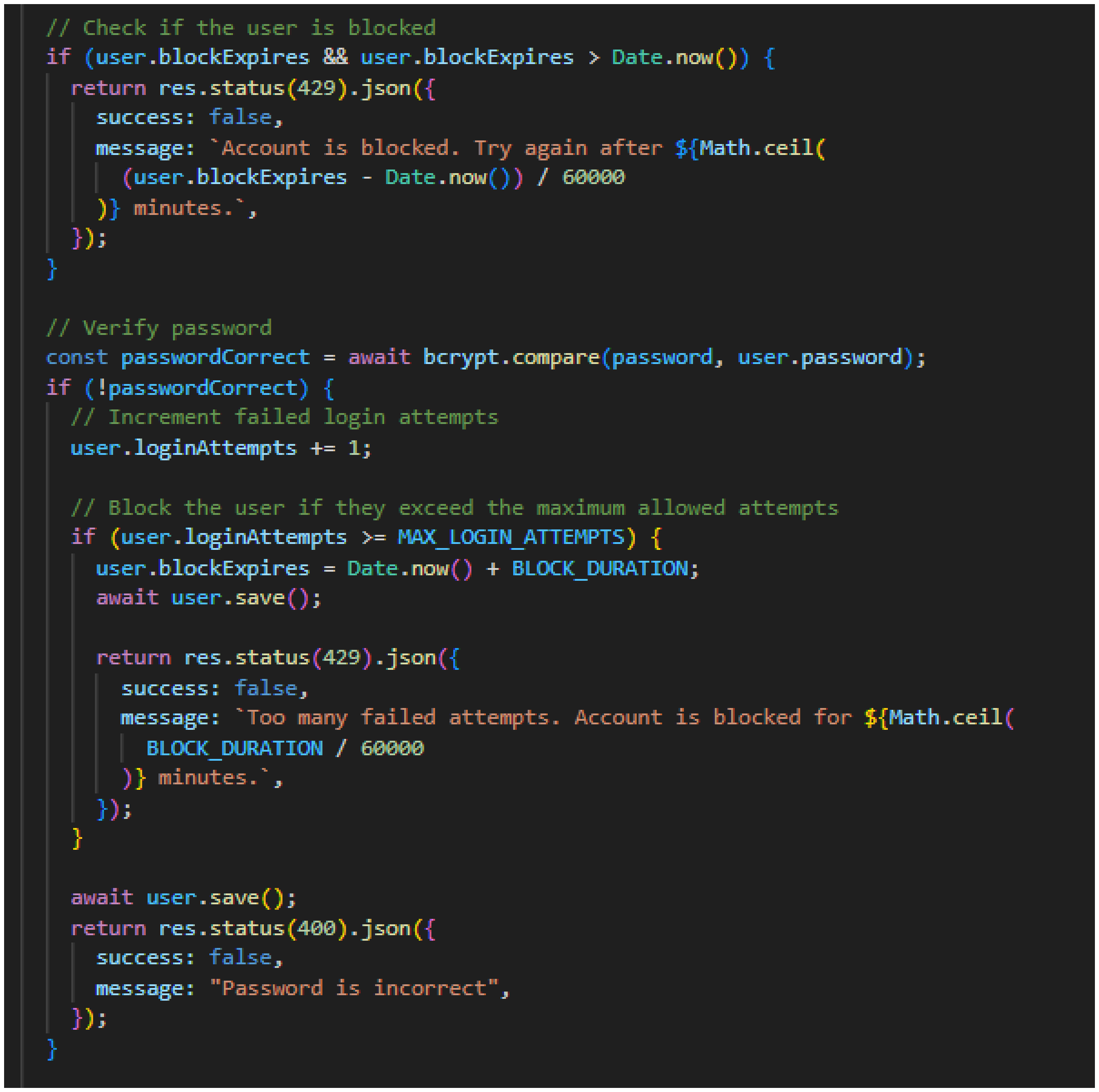
•••••••••

•••••••••

Figure 21: Client-Side OTP in Register

LOCK ACCOUNT AFTER MULTIPLE FAILED LOGINS

To increase security and coding safeguards against bad givers or guessing, it is recommended to freeze one's account if repeated failed attempts are made. This strategy entails determination of a lockout policy that is a measure of the number of tries a person can make before their account is locked in a particular amount of time. The standard would be to set the authorized number of attempts to be five and the time lock would be set to half an hour. To this aim, further fields are created in the user model to keep track of the failed login attempts and the lockout situation. In the event of the login process, if the number of failures that arise from improper entries is beyond the authorized limit, the account is closed for the given time. This is done by comparing and changing the login credentials and lockout property in the user database. Through this, the system disallows any unauthorized attempts to log in through different erroneous user ID and password by locking the account to protect the user accounts and the overall application.



```
// Check if the user is blocked
if (user.blockExpires && user.blockExpires > Date.now()) {
  return res.status(429).json({
    success: false,
    message: `Account is blocked. Try again after ${Math.ceil(
      (user.blockExpires - Date.now()) / 60000
    )} minutes.`,
  });
}

// Verify password
const passwordCorrect = await bcrypt.compare(password, user.password);
if (!passwordCorrect) {
  // Increment failed login attempts
  user.loginAttempts += 1;

  // Block the user if they exceed the maximum allowed attempts
  if (user.loginAttempts >= MAX_LOGIN_ATTEMPTS) {
    user.blockExpires = Date.now() + BLOCK_DURATION;
    await user.save();

    return res.status(429).json({
      success: false,
      message: `Too many failed attempts. Account is blocked for ${Math.ceil(
        BLOCK_DURATION / 60000
      )} minutes.`,
    });
  }

  await user.save();
  return res.status(400).json({
    success: false,
    message: "Password is incorrect",
  });
}
```

Figure 22: Server-Side Lock user for multiple attempts

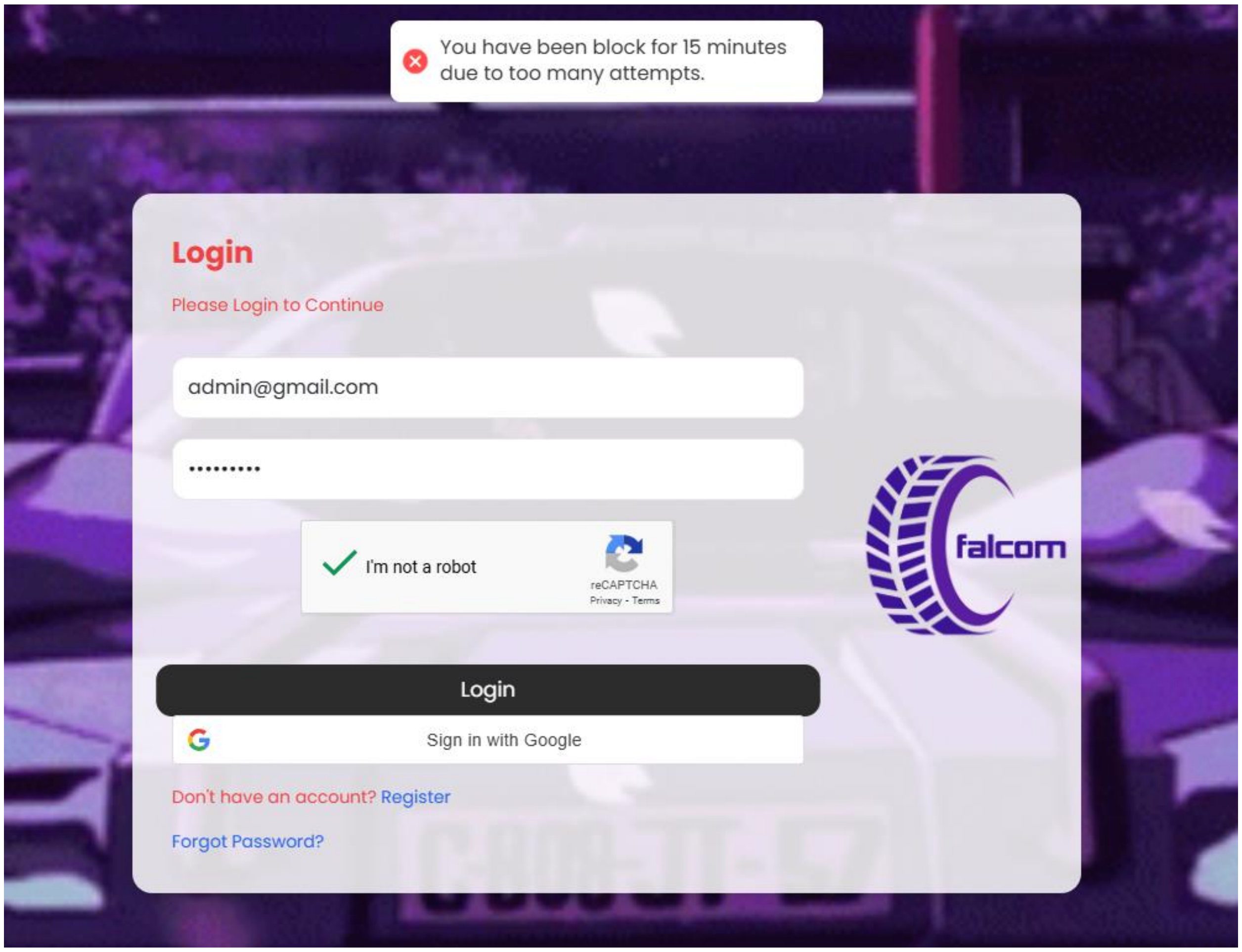
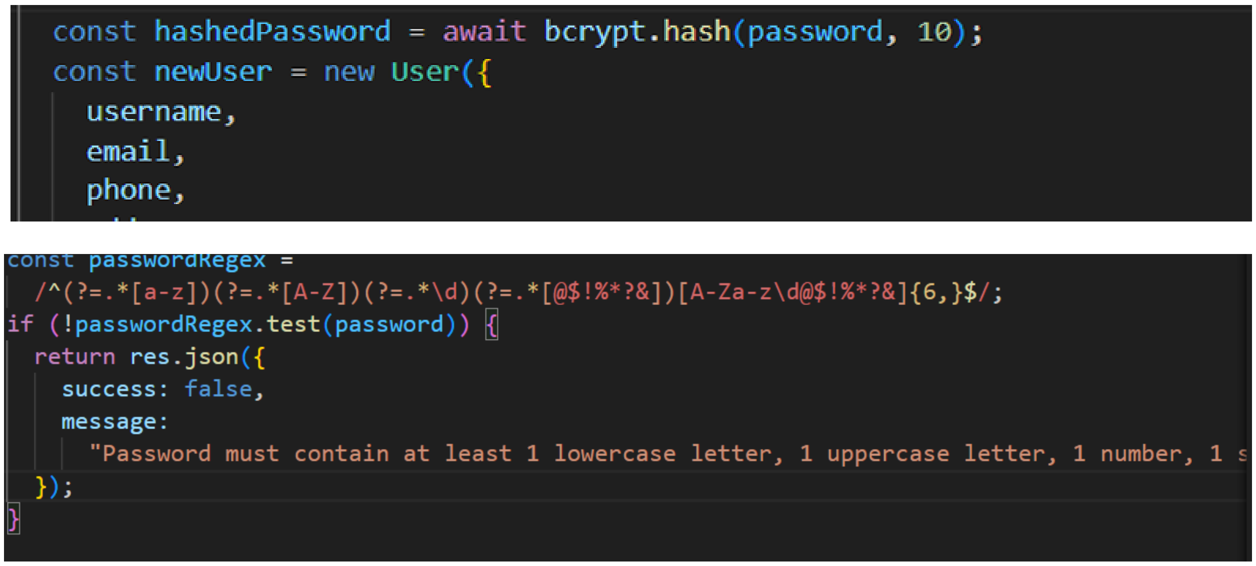

You have been block for 15 minutes due to too many attempts.

Figure 23: Client-Side Implementation

HASHING PASSWORD

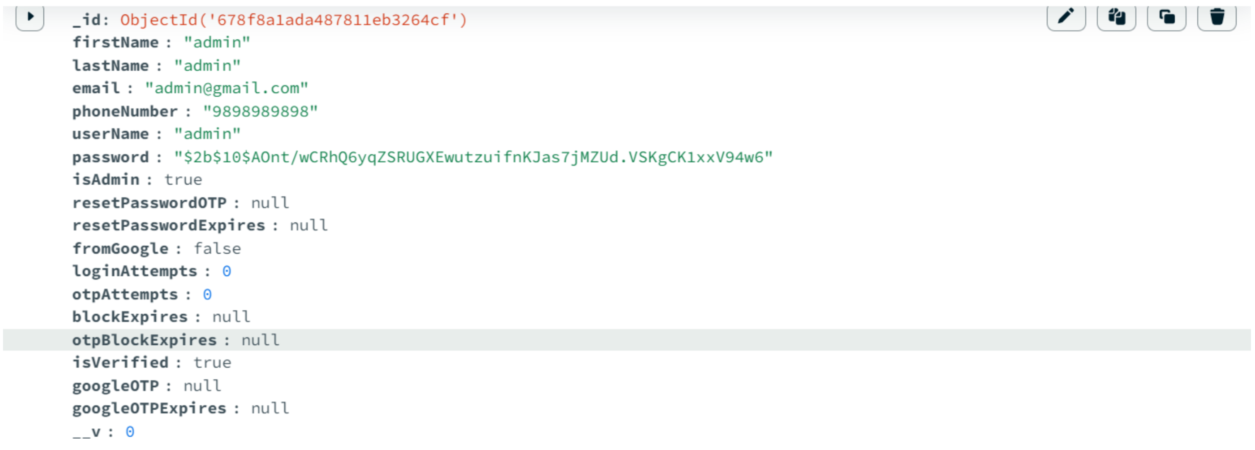
Password hashing is something that lays in the basics of security for keeping the users' login details in a database. Instead of maintaining the passwords as plain-text it is encrypted using a cryptographic hash function so in case the database is penetrated the passwords cannot be simply assumed. They produce a string of characters of a specified size that seems random and does not contain any identifiable information with the original password. Also, another salt of say random string is generally applied to the passwords to increase security since, for the same value of password, a different value of hash string is produced. This keeps the hackers from using tables they had pre-computed known as rainbow tables and reverse computer for passwords. By maintaining only password hashes as well as the salt added to them, the database is made more secure as even if the hashed data is somehow exposed the actual passwords are a lot tougher for the attackers to access. This strategy considerably increases the general security of the application and keeps the users' data from being hacked (*How to Hash Passwords: One-Way Road to Enhanced Security*, n.d.).



```
const hashedPassword = await bcrypt.hash(password, 10);
const newUser = new User({
  username,
  email,
  phone,
```

```
const passwordRegex =
  /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{6,}$/;
if (!passwordRegex.test(password)) {
  return res.json({
    success: false,
    message:
      "Password must contain at least 1 lowercase letter, 1 uppercase letter, 1 number, 1 s
  });
}
```

Figure 24:Server-Side Hashed Password



```
_id: ObjectId('678f8a1ada487811eb3264cf')
firstName : "admin"
lastName : "admin"
email : "admin@gmail.com"
phoneNumber : "9898989898"
userName : "admin"
password : "$2b$10$AOnt/wCRhQ6yqZSRUGXEwutzuifnKJas7jMZUd.VSKgCK1xxV94w6"
isAdmin : true
resetPasswordOTP : null
resetPasswordExpires : null
fromGoogle : false
loginAttempts : 0
otpAttempts : 0
blockExpires : null
otpBlockExpires : null
isVerified : true
googleOTP : null
googleOTPExpires : null
__v : 0
```

Figure 25:Hased Password save in database

IMPLEMENTATION OF CAPTCHA

The integration of CAPTCHA before login in the **Falcom App** enhances the security of the platform, ensuring a safe and reliable user experience. This feature protects against malicious activities, such as brute-force attacks, credential stuffing, and automated bots, by requiring users to complete a CAPTCHA challenge before submitting their login credentials. CAPTCHA stands for Completely Automated Public Turing test to tell Computers and Humans Apart (Balankdharan, 2023).

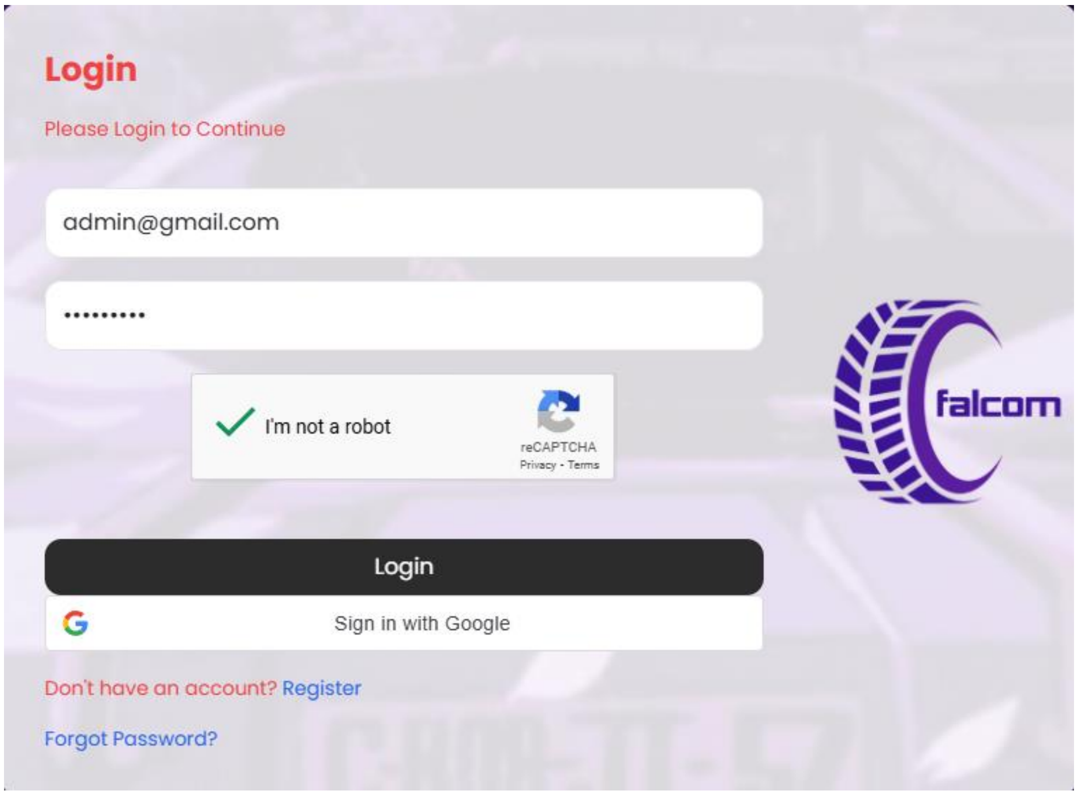
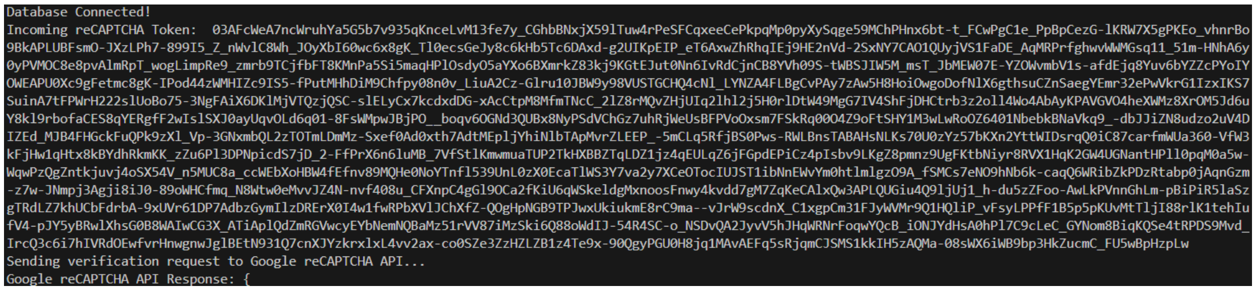


Figure 26: Implement Captcha in Client-Side

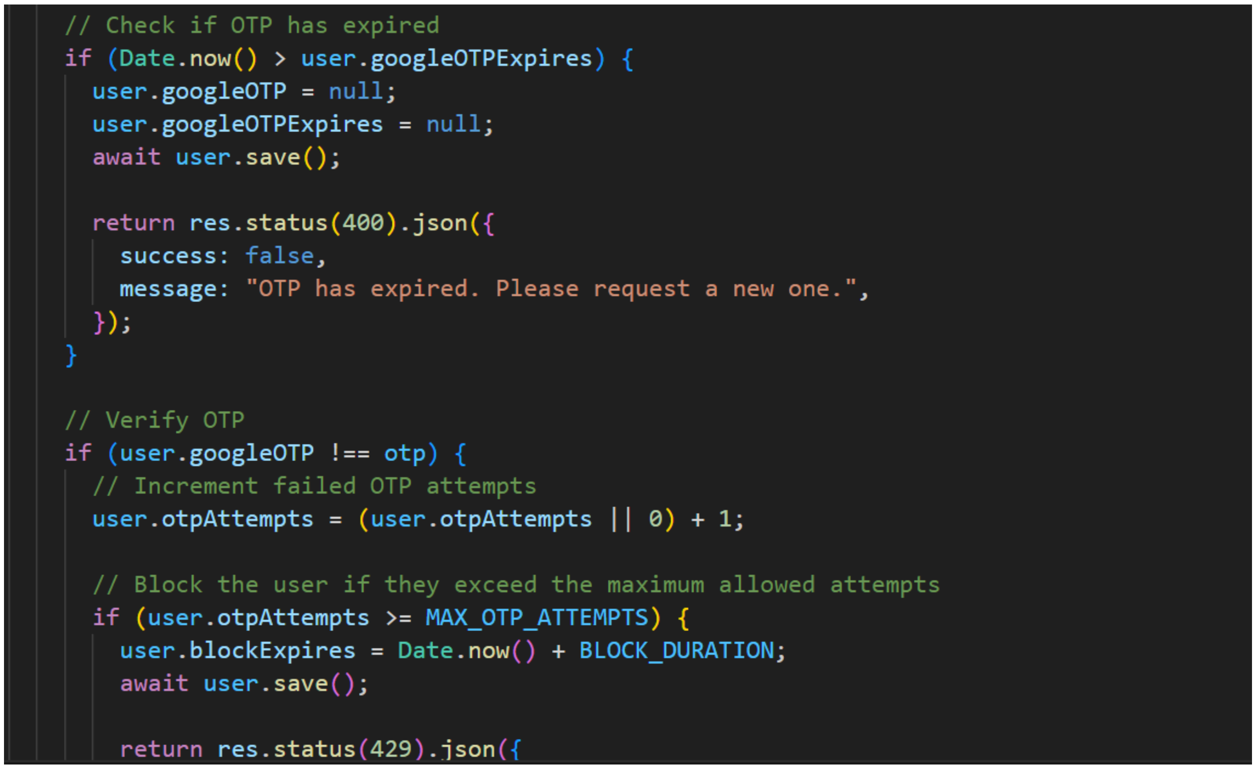


```
Database Connected!
Incoming reCAPTCHA Token:  03AFcWeA7ncWruhYa5G5b7v935qKnceLvM13fe7y_CGhbBNxjX59lTuw4rPeSFCqxeeCePkpqMp0pyXySqge59MChPHnx6bt-t_FCwPgC1e_PpBpCezG-lKRW7X5gPKEo_vhnrBo
9BkAPLUBFsmO-JXzLPh7-899I5_Z_nWvlC8Wh_JOyXbI60wc6x8gK_Tl0ecsGeJy8c6kHb5Tc6DAxd-g2UIKpEIP_eT6AxwZhRhqIEj9HE2nVd-2SxNY7CAO1QUyjVS1FaDE_AqMRPrfghwvWWMGsq11_51m-HNhA6y
0yPVMOC8e8pvAlmRpT_wogLimpRe9_zmrb9TCjfbFT8KMnPa5Si5maqHPlOsdyO5aYXo6BXmrkZ83kj9KGtEJut0Nn6IvRdCjnCB8YVh09S-tWBSJIW5M_msT_JbMEW07E-YZOWvmbV1s-afdEjq8Yuv6bYZZcPYoIY
OWEAPU0Xc9gFetmc8gK-IPod44zWMHIZc9IS5-fPutMHhDiM9Chfpy08n0v_LiuA2Cz-Glru10JBW9y98VUSTGCHQ4cNl_LYNZA4FLBgCvPAy7zAw5H8HoiOwgoDofNlX6gthsuCZnSaegYEmr32ePwVkrG1IzxIKS7
SuinA7tFPWrH222slUoBo75-3NgFAiX6DKlMjVTQzjQSC-slELyCx7kcdxdDG-xAcCtpM8MfmTNcC_2lZ8rMQvZHjUIq2lhl2j5H0rlDtW49MgG7IV4ShFjDHCtrb3z2oll4Wo4AbAyKPAVGVO4heXWMz8XrOM5Jd6u
Y8kl9rbofaCES8qYERgfF2wIslSXJ0ayUqvOLd6q01-8FsWMpwJBjPO__boqv6OGNd3QUBx8NyPSdVChGz7uhRjWeUsBFPVoOxsm7FSkRq00O4Z9oFtSHY1M3wLwRoOZ6401NbebkBNaVkq9_-dbJJiZN8udzo2uV4D
IZEd_MJB4FHGckFuQPk9zXl_Vp-3GNxmbQL2zTOTmLDmMz-Sxef0Ad0xth7AdtMEpljYhiNlbTApMvrZLEEP_-5mCLq5RfjBS0Pws-RWLBnsTABAHsNLKs70U0zYz57bKXn2YttWIDsrqQ0iC87carfmWUa360-VfW3
kFjHw1qHtx8kBYdhRkmKK_zZu6Pl3DPNpicdS7jD_2-FfPrX6n6luMB_7VfStlKmwmuaTUP2TkHXBBZTqLDZ1jz4qEULqZ6jFGpdEPiCz4pIsbv9LKgZ8pmnz9UgFKtbNiyr8RVX1HqK2GW4UGNantHPll0pqM0a5w-
WqwPzQgZntkjuvj4oSX54V_n5MUC8a_ccWEbXoHBW4fEfnv89MQHe0NoYTnfl539UnL0zX0EcaTlWS3Y7va2y7XCeOTocIUJST1ibNnEWvYm0htlmlgzO9A_fSMCs7eNO9hNb6k-caqQ6WRibZkPDzRtabp0jAqnGzm
-z7w-JNmpj3Agji8iJ0-89oWHCfmq_N8Wtw0eMvvJZ4N-nvf408u_CFXnpC4gGl9OCa2fKiU6qWSkeldgMxnoosFnwy4kvdd7gM7ZqKeCAlxQw3APLQUGiu4Q9ljUj1_h-du5zZFoo-AwLkPVnnGhLm-pBiPiR5laSz
gTRdLZ7khUCbFdrbA-9xUVr61DP7AdbzGymIlzDRErX0I4w1fwRPbXVlJChXfZ-QOgHpNGB9TPJwxUkiukmE8rC9ma--vJrW9scdnX_C1xgpCm31FJyWVMr9Q1HQliP_vFsyLPPfF1B5p5pKUvMtTljI88rlK1tehIu
fV4-pJY5yBRwlXhsG0B8WAIwCG3X_ATiAplQdZmRGVwcyEYbNemNQBaMz51rVV87iMzSki6Q88oWdIJ-54R4SC-o_NSDvQA2JyvV5hJHqWRNrFoqwYQcB_iONJYdHsA0hPl7C9cLeC_GYNom8BiqKQSe4tRPDS9Mvd_
IrcQ3c6i7hIVRdOEwfvrHnwgnwJglBEtN931Q7cnXJYzkrxlxL4vv2ax-co0SZe3ZzHZLZB1z4Te9x-90QgyPGU0H8jq1MAvAEFq5sRjqmCJSMS1kkIH5zAQMa-08sWX6iWB9bp3HkZucmC_FU5wBpHzpLw
Sending verification request to Google reCAPTCHA API...
Google reCAPTCHA API Response: {
```

Figure 27: Captcha Server-side response

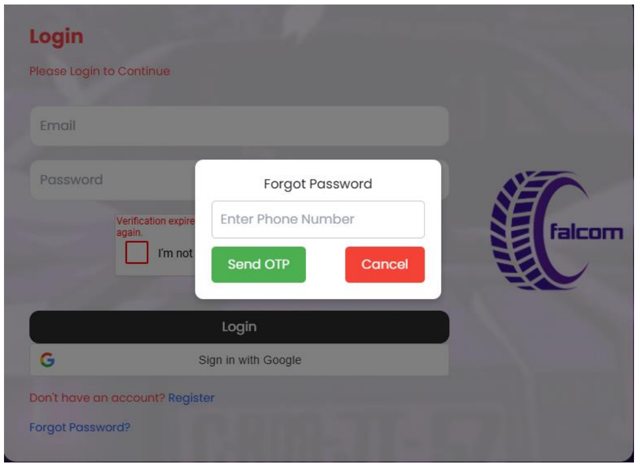
RATE LIMITING

The common security feature known as rate limiting prevents a user from making too many requests within a specific timeframe to a web application. This is good as a measure against abusive access and DoS attacks where the attacker churns through requests much faster than the server can handle, and the result is a slowing down of the server or complete unavailability to regular users Because of this, they also limit the rate of requests so that it also reduces brute force attacks; a method in which attackers use several combinations of username and password to access a system a program. This precaution prevents a given user from utilizing most of the application's resources, thus preventing inefficiency and making the system more stable. Also, rate limiting aids in moderating traffic flow on the server; if traffic is allowed to rush in within a short time, this puts a lot of pressure on the server, and therefore affects everyone that must use the server side.



```
// Check if OTP has expired
if (Date.now() > user.googleOTPExpires) {
  user.googleOTP = null;
  user.googleOTPExpires = null;
  await user.save();

  return res.status(400).json({
    success: false,
    message: "OTP has expired. Please request a new one.",
  });
}

// Verify OTP
if (user.googleOTP !== otp) {
  // Increment failed OTP attempts
  user.otpAttempts = (user.otpAttempts || 0) + 1;

  // Block the user if they exceed the maximum allowed attempts
  if (user.otpAttempts >= MAX_OTP_ATTEMPTS) {
    user.blockExpires = Date.now() + BLOCK_DURATION;
    await user.save();

    return res.status(429).json({
```

Figure 28: OTP limitation Server-Side



Login

Please Login to Continue

Email

Password

Verification expire again.

I'm not

Forgot Password

Enter Phone Number

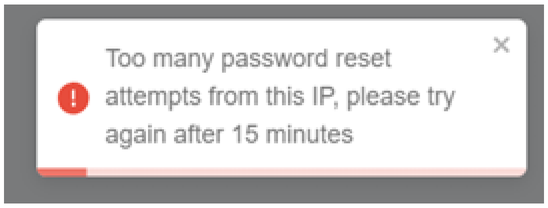
Send OTP Cancel

Login

Sign in with Google

Don't have an account? Register

Forgot Password?



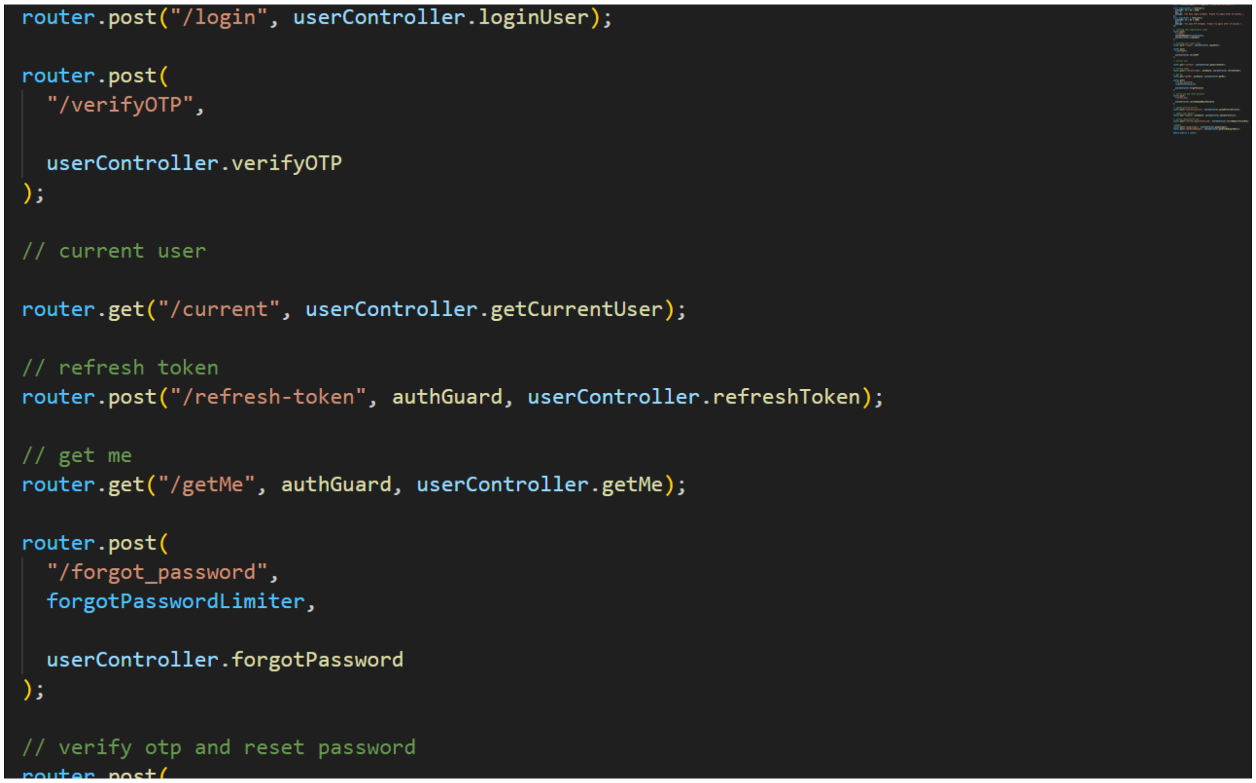
Too many password reset attempts from this IP, please try again after 15 minutes

Figure 29:Limitation in Frontend

ROLE BASED ACCESS CONTROL

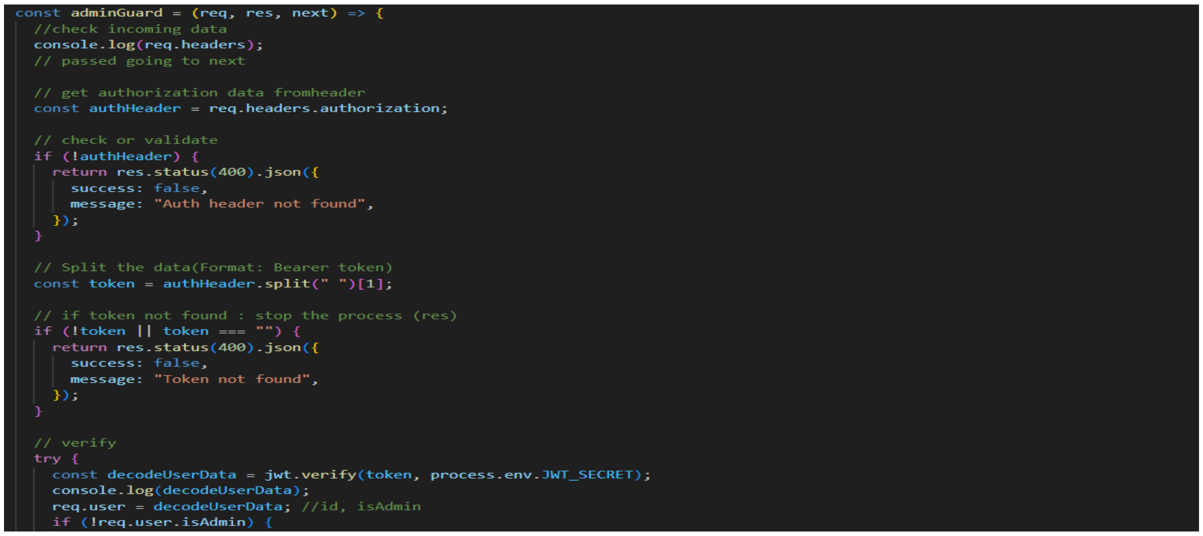
RBAC can be defined as a security model that regulates system's access to users with roles in the organization. Every role relates to privileges, which determines which actions the end user authorized to certain role can make (*What Is Role-Based Access Control (RBAC)? Examples, Benefits, and More*, n.d.). The approach guarantees that user has only the specific access level required to perform his job profile and thus discourages any unnamed activity (Team, n.d.).

Server-Side



```
router.post("/login", userController.loginUser);

router.post(
  "/verifyOTP",

  userController.verifyOTP
);

// current user

router.get("/current", userController.getCurrentUser);

// refresh token
router.post("/refresh-token", authGuard, userController.refreshToken);

// get me
router.get("/getMe", authGuard, userController.getMe);

router.post(
  "/forgot_password",
  forgotPasswordLimiter,

  userController.forgotPassword
);

// verify otp and reset password
```

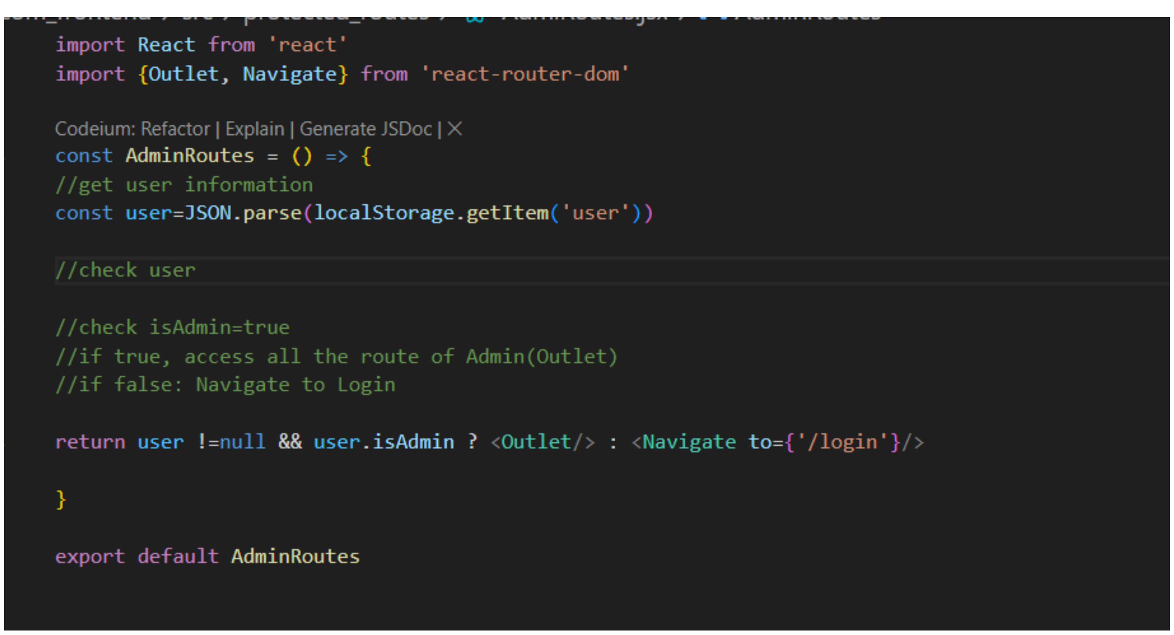
Figure 30: Guard in route



```
const adminGuard = (req, res, next) => {
  //check incoming data
  console.log(req.headers);
  // passed going to next

  // get authorization data fromheader
  const authHeader = req.headers.authorization;

  // check or validate
  if (!authHeader) {
    return res.status(400).json({
      success: false,
      message: "Auth header not found",
    });
  }

  // Split the data(Format: Bearer token)
  const token = authHeader.split(" ")[1];

  // if token not found : stop the process (res)
  if (!token || token === "") {
    return res.status(400).json({
      success: false,
      message: "Token not found",
    });
  }

  // verify
  try {
    const decodeUserData = jwt.verify(token, process.env.JWT_SECRET);
    console.log(decodeUserData);
    req.user = decodeUserData; //id, isAdmin
    if (!req.user.isAdmin) {
```

Figure 31:Middleware of guard

Client-Side

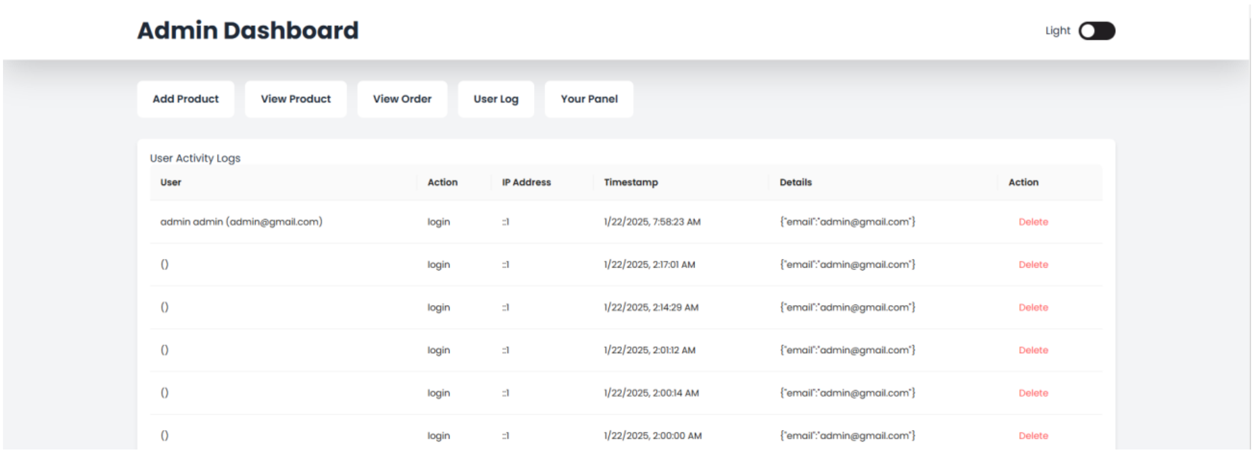


```
import React from 'react'
import {Outlet, Navigate} from 'react-router-dom'

Codeium: Refactor | Explain | Generate JSDoc | X
const AdminRoutes = () => {
//get user information
const user=JSON.parse(localStorage.getItem('user'))

//check user

//check isAdmin=true
//if true, access all the route of Admin(Outlet)
//if false: Navigate to Login

return user !=null && user.isAdmin ? <Outlet/> : <Navigate to={'/login'}/>

}

export default AdminRoutes
```

Figure 32:Admin Navigation

USERLOG

Admin can access all the user log from admin dashboard such as user ip, username, user email and even the time when user login or user failed to login and if the user has attempts to login for many times, then user will be blocked by admin.



User Activity Logs

User	Action	IP Address	Timestamp	Details	Action
admin admin (admin@gmail.com)	login	::1	1/22/2025, 7:58:23 AM	{"email":"admin@gmail.com"}	Delete
()	login	::1	1/22/2025, 2:17:01 AM	{"email":"admin@gmail.com"}	Delete
()	login	::1	1/22/2025, 2:14:29 AM	{"email":"admin@gmail.com"}	Delete
()	login	::1	1/22/2025, 2:01:12 AM	{"email":"admin@gmail.com"}	Delete
()	login	::1	1/22/2025, 2:00:14 AM	{"email":"admin@gmail.com"}	Delete
()	login	::1	1/22/2025, 2:00:00 AM	{"email":"admin@gmail.com"}	Delete

Figure 33:User-Log Dashboard

CONCLUSION

In conclusion, Falcom is constructed with a comprehensive security framework that complies with industry standards and best practices, guaranteeing substantial protection against potential threats and weaknesses. The system integrates many security protocols, including the prevention of unauthorized multiple login attempts, user activity tracking, and the implementation of XSS mitigation, alongside other sophisticated techniques. These safeguards together protect the site, its users, and their information. Moreover, Falcom's security framework adheres to secure design principles, encompassing least privilege access, encrypted data transport, and robust authentication procedures. Consistent input validation, rate limitation, and periodic security evaluations enhance the platform, guaranteeing a secure and dependable environment for users.

By emphasizing security, Falcom ensures a reliable and uninterrupted experience for customers in search of premium bike, car, and truck tires, tubes, tire fluid, and additional products. Falcom's commitment to security and reliability positions it as a dependable platform that addresses the varied requirements of its users, while instilling confidence in its sustainable and customer-centric solutions.

VERSION CONTROL

GitHub Link: https://github.com/Rushmitkarki/220046_security_cw2_final

YouTube Link:

REFERENCES

MERN Stack explained. (n.d.). MongoDB. https://www.mongodb.com/resources/languages/mern-stack

Stevens, E. (2025, January 8). What is user experience (UX) design? Everything you need to know. *CareerFoundry*. https://careerfoundry.com/en/blog/ux-design/what-is-user-experience-ux-design-everything-you-need-to-know-to-get-started/

Moteria, D. (2024, September 24). *MERN Stack Security Best Practices: How to keep your Applications safe*. Elluminati Inc. https://www.elluminatiinc.com/mern-stack-security/

What is HTTPS? A Definition and How to Switch to HTTPS? | Fortinet. (n.d.). Fortinet. https://www.fortinet.com/resources/cyberglossary/what-is-https

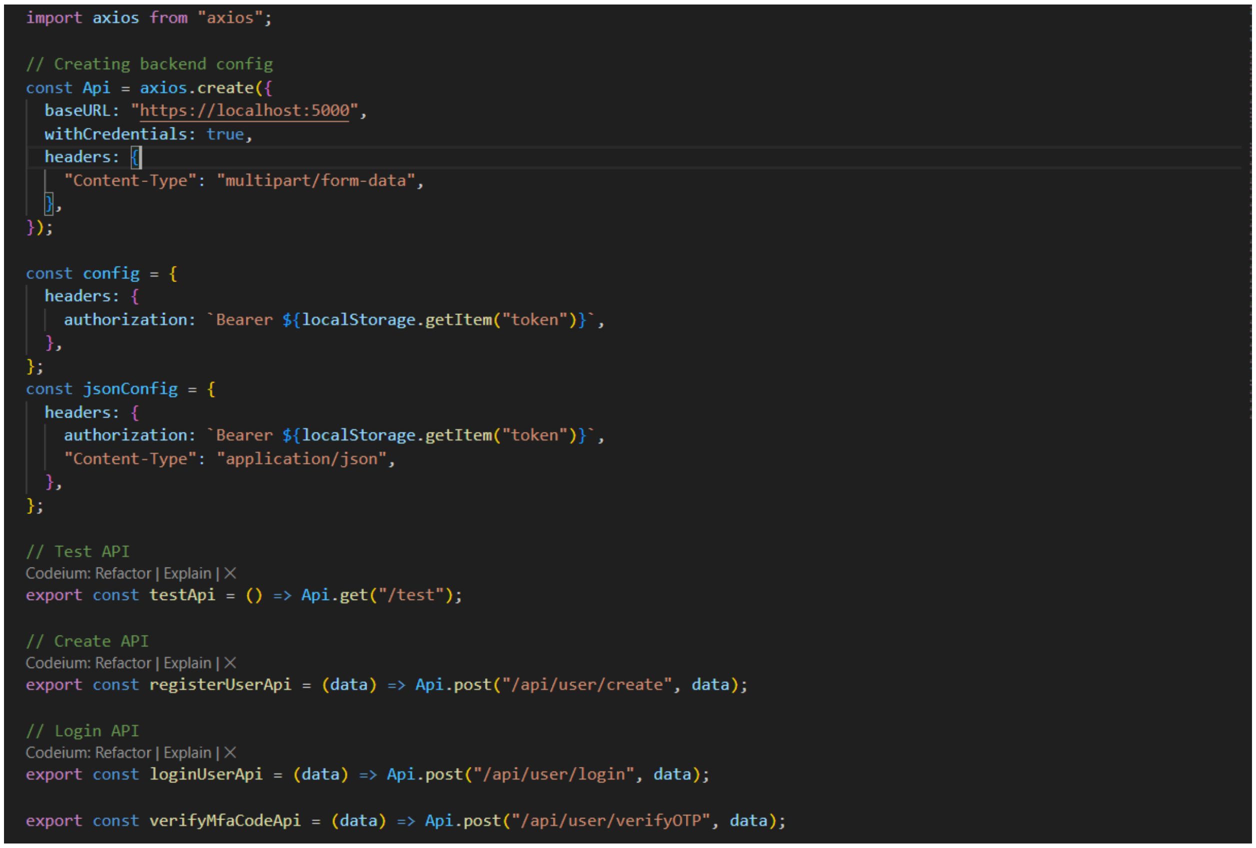
How to Hash Passwords: One-Way Road to Enhanced Security. (n.d.). Auth0 - Blog. https://auth0.com/blog/hashing-passwords-one-way-road-to-security/

Balankdharan. (2023, October 29). Adding ReCAPTCHA in MERN application - Balankdharan - Medium. *Medium*. https://medium.com/@balankdharan/adding-recaptcha-in-mern-application-10391bf10f6d

What is Role-Based Access Control (RBAC)? Examples, Benefits, and More. (n.d.). https://www.digitalguardian.com/blog/what-role-based-access-control-rbac-examples-benefits-and-more

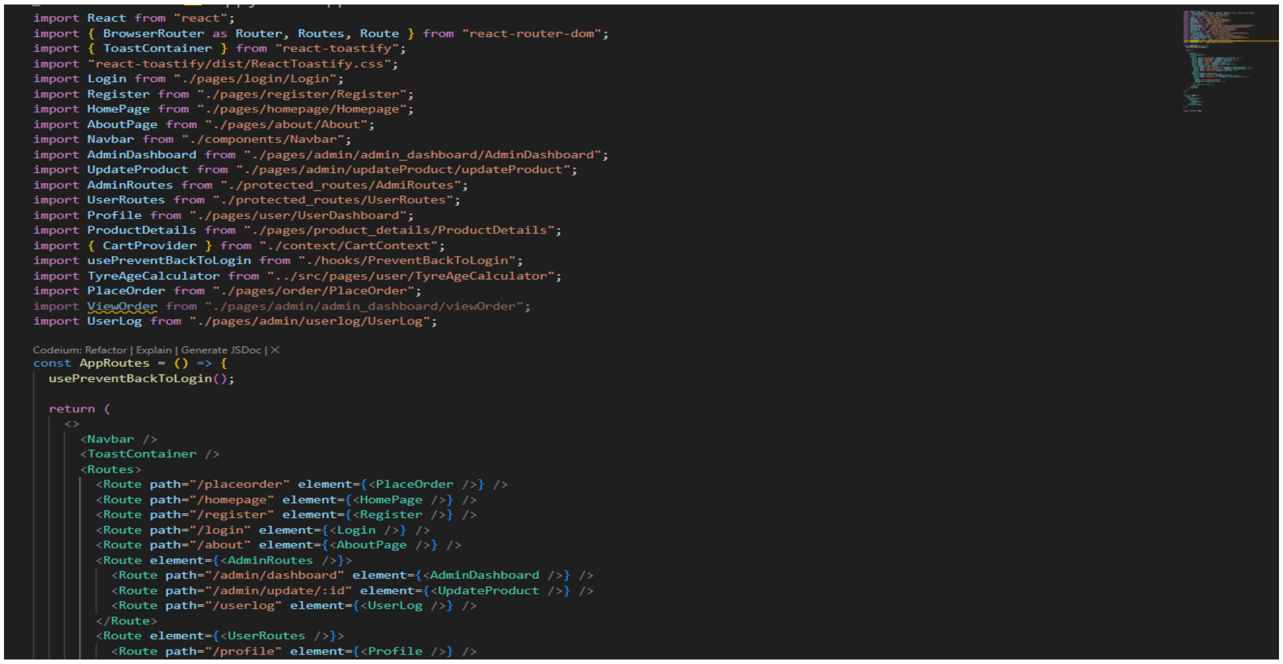
Team, M. D. (n.d.). *Role-Based access control in Self-Managed deployments*. MongoDB Manual v8.0. https://www.mongodb.com/docs/manual/core/authorization/

APPENDIX



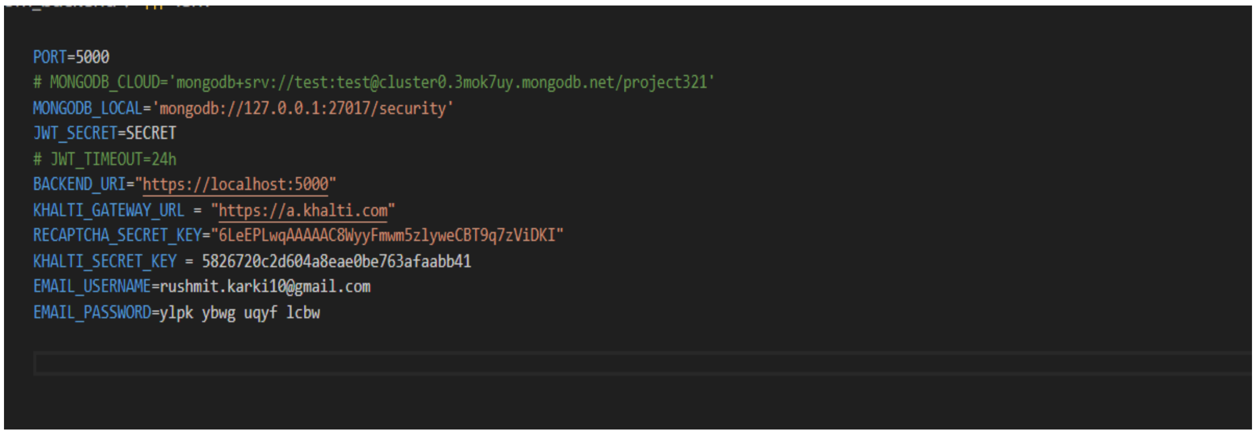
```
import axios from "axios";

// Creating backend config
const Api = axios.create({
  baseURL: "https://localhost:5000",
  withCredentials: true,
  headers: {
    "Content-Type": "multipart/form-data",
  },
});

const config = {
  headers: {
    authorization: `Bearer ${localStorage.getItem("token")}`,
  },
};
const jsonConfig = {
  headers: {
    authorization: `Bearer ${localStorage.getItem("token")}`,
    "Content-Type": "application/json",
  },
};

// Test API
Codeium: Refactor | Explain | X
export const testApi = () => Api.get("/test");

// Create API
Codeium: Refactor | Explain | X
export const registerUserApi = (data) => Api.post("/api/user/create", data);

// Login API
Codeium: Refactor | Explain | X
export const loginUserApi = (data) => Api.post("/api/user/login", data);

export const verifyMfaCodeApi = (data) => Api.post("/api/user/verifyOTP", data);
```

Figure 34:Client-Side API Endpoint



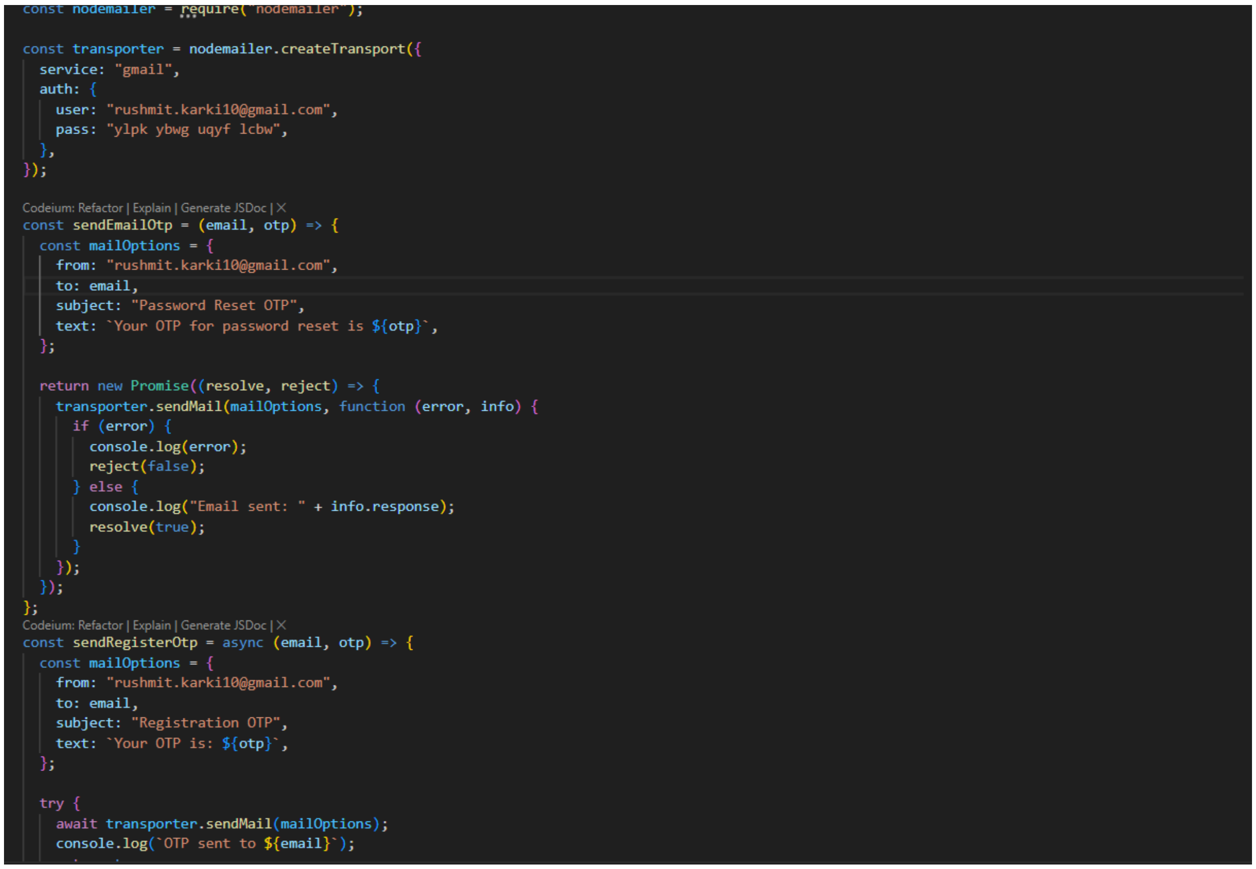
```
import React from "react";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import { ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import Login from "./pages/login/Login";
import Register from "./pages/register/Register";
import HomePage from "./pages/homepage/Homepage";
import AboutPage from "./pages/about/About";
import Navbar from "./components/Navbar";
import AdminDashboard from "./pages/admin/admin_dashboard/AdminDashboard";
import UpdateProduct from "./pages/admin/updateProduct/updateProduct";
import AdminRoutes from "./protected_routes/AdmiRoutes";
import UserRoutes from "./protected_routes/UserRoutes";
import Profile from "./pages/user/UserDashboard";
import ProductDetails from "./pages/product_details/ProductDetails";
import { CartProvider } from "./context/CartContext";
import usePreventBackToLogin from "./hooks/PreventBackToLogin";
import TyreAgeCalculator from "../src/pages/user/TyreAgeCalculator";
import PlaceOrder from "./pages/order/PlaceOrder";
import ViewOrder from "./pages/admin/admin_dashboard/viewOrder";
import UserLog from "./pages/admin/userlog/UserLog";

Codeium: Refactor | Explain | Generate JSDoc | X
const AppRoutes = () => {
  usePreventBackToLogin();

  return (
    <>
      <Navbar />
      <ToastContainer />
      <Routes>
        <Route path="/placeorder" element={<PlaceOrder />} />
        <Route path="/homepage" element={<HomePage />} />
        <Route path="/register" element={<Register />} />
        <Route path="/login" element={<Login />} />
        <Route path="/about" element={<AboutPage />} />
        <Route element={<AdminRoutes />}>
          <Route path="/admin/dashboard" element={<AdminDashboard />} />
          <Route path="/admin/update/:id" element={<UpdateProduct />} />
          <Route path="/userlog" element={<UserLog />} />
        </Route>
        <Route element={<UserRoutes />}>
          <Route path="/profile" element={<Profile />} />
```

Figure 35: Routes of Client-Side



```
PORT=5000
# MONGODB_CLOUD='mongodb+srv://test:test@cluster0.3mok7uy.mongodb.net/project321'
MONGODB_LOCAL='mongodb://127.0.0.1:27017/security'
JWT_SECRET=SECRET
# JWT_TIMEOUT=24h
BACKEND_URI="https://localhost:5000"
KHALTI_GATEWAY_URL = "https://a.khalti.com"
RECAPTCHA_SECRET_KEY="6LeEPLwqAAAAAC8WyyFmwm5zlyweCBT9q7zViDKI"
KHALTI_SECRET_KEY = 5826720c2d604a8eae0be763afaabb41
EMAIL_USERNAME=rushmit.karki10@gmail.com
EMAIL_PASSWORD=ylpk ybwg uqyf lcbw
```

Figure 36: .Env file



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

const transporter = nodemailer.createTransport({
  service: "gmail",
  auth: {
    user: "rushmit.karki10@gmail.com",
    pass: "ylpk ybwg uqyf lcbw",
  },
});

Codeium: Refactor | Explain | Generate JSDoc | X
const sendEmailOtp = (email, otp) => {
  const mailOptions = {
    from: "rushmit.karki10@gmail.com",
    to: email,
    subject: "Password Reset OTP",
    text: `Your OTP for password reset is ${otp}`,
  };

  return new Promise((resolve, reject) => {
    transporter.sendMail(mailOptions, function (error, info) {
      if (error) {
        console.log(error);
        reject(false);
      } else {
        console.log("Email sent: " + info.response);
        resolve(true);
      }
    });
  });
};
Codeium: Refactor | Explain | Generate JSDoc | X
const sendRegisterOtp = async (email, otp) => {
  const mailOptions = {
    from: "rushmit.karki10@gmail.com",
    to: email,
    subject: "Registration OTP",
    text: `Your OTP is: ${otp}`,
  };

  try {
    await transporter.sendMail(mailOptions);
    console.log(`OTP sent to ${email}`);
```

Figure 37:Nodemailer uses

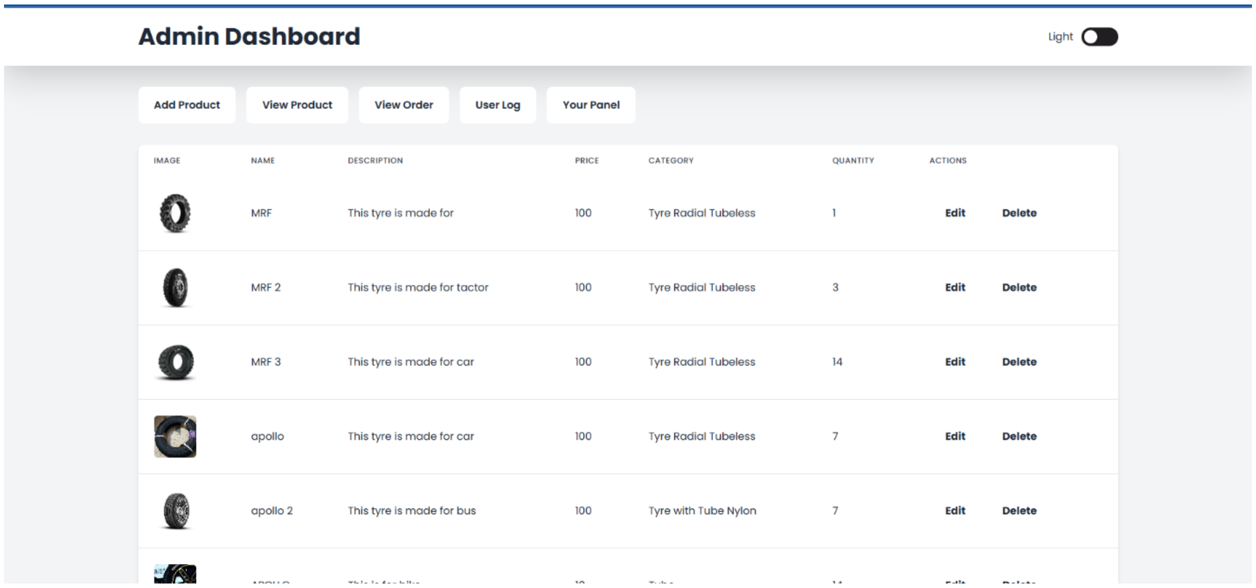
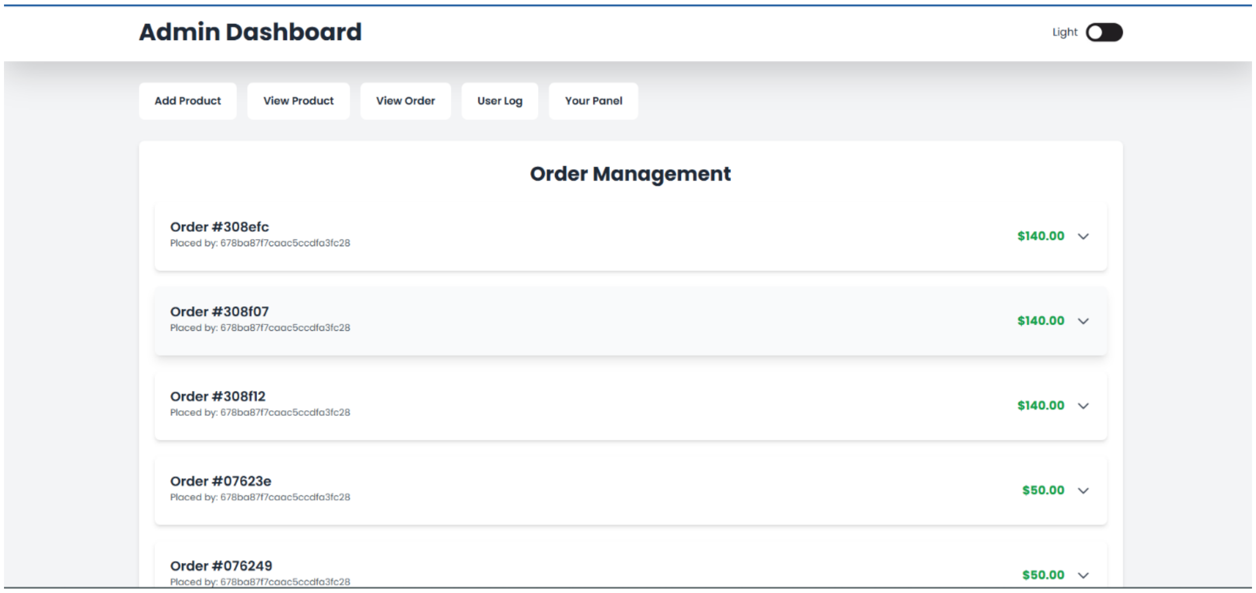


IMAGE	NAME	DESCRIPTION	PRICE	CATEGORY	QUANTITY	ACTIONS	
	MRF	This tyre is made for	100	Tyre Radial Tubeless	1	Edit	Delete
	MRF 2	This tyre is made for tactor	100	Tyre Radial Tubeless	3	Edit	Delete
	MRF 3	This tyre is made for car	100	Tyre Radial Tubeless	14	Edit	Delete
	apollo	This tyre is made for car	100	Tyre Radial Tubeless	7	Edit	Delete
	apollo 2	This tyre is made for bus	100	Tyre with Tube Nylon	7	Edit	Delete

Figure 38: UI of View Product by admin



Order Management

Order #308efc
Placed by: 678ba87f7caac5ccdfa3fc28
$140.00

Order #308f07
Placed by: 678ba87f7caac5ccdfa3fc28
$140.00

Order #308f12
Placed by: 678ba87f7caac5ccdfa3fc28
$140.00

Order #07623e
Placed by: 678ba87f7caac5ccdfa3fc28
$50.00

Order #076249
Placed by: 678ba87f7caac5ccdfa3fc28
$50.00

Figure 39:View Order by admin

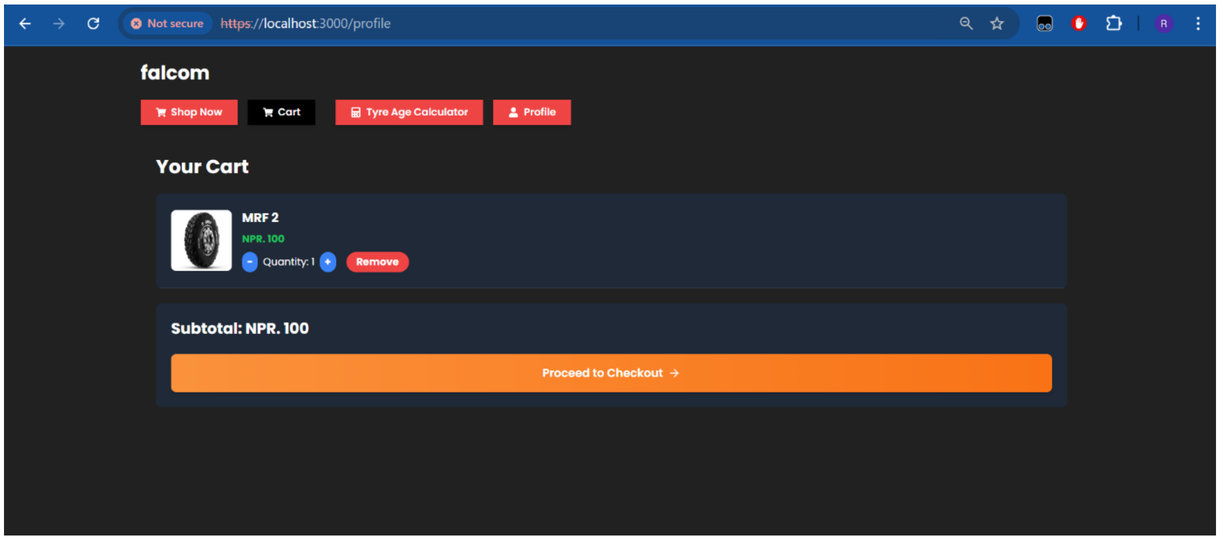
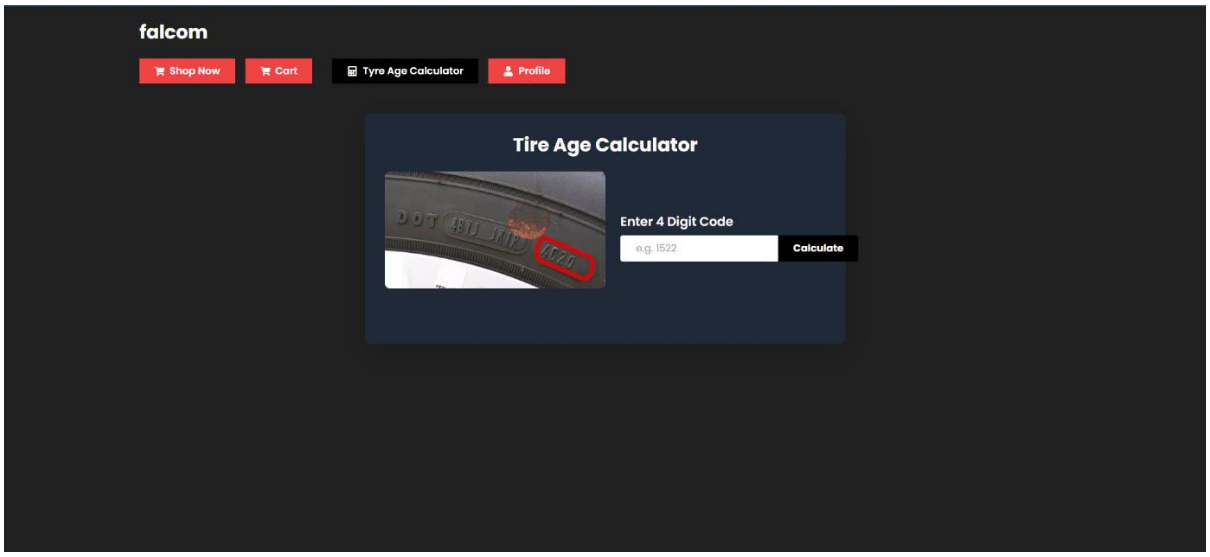
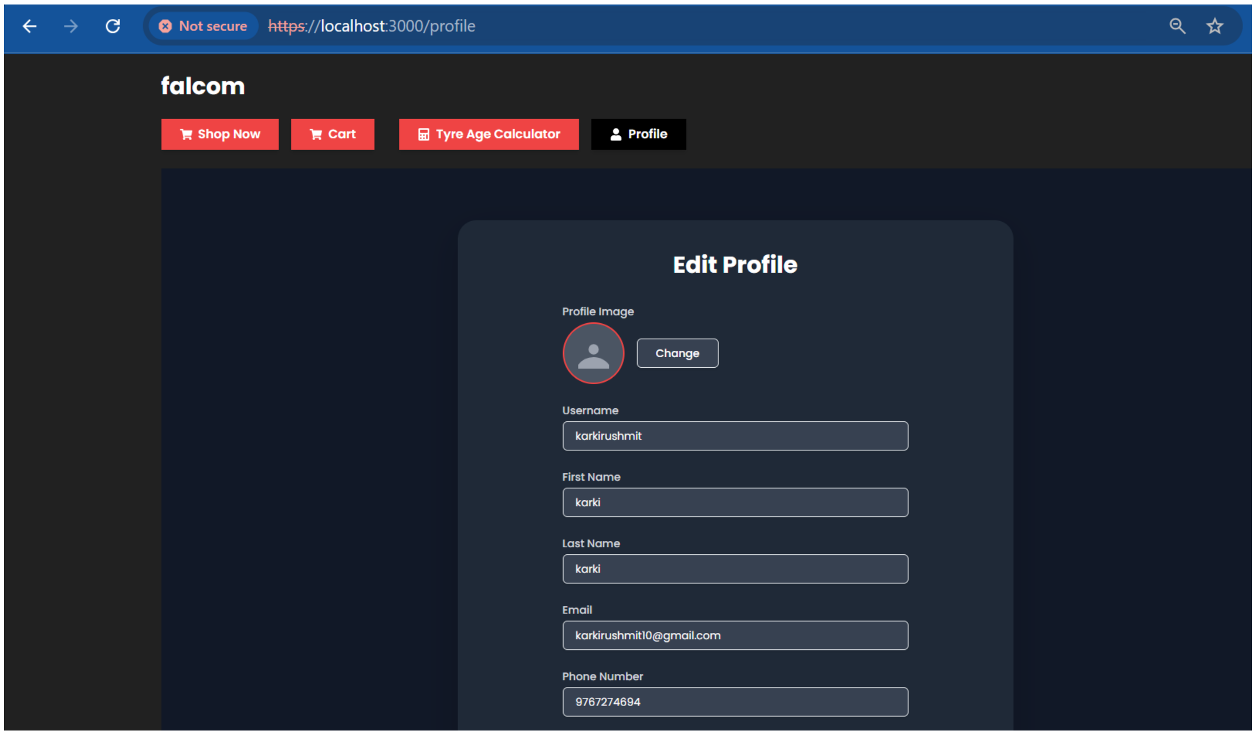


Figure 40:Add to Cart user side





Edit Profile

Profile Image

Username: karkirushmit

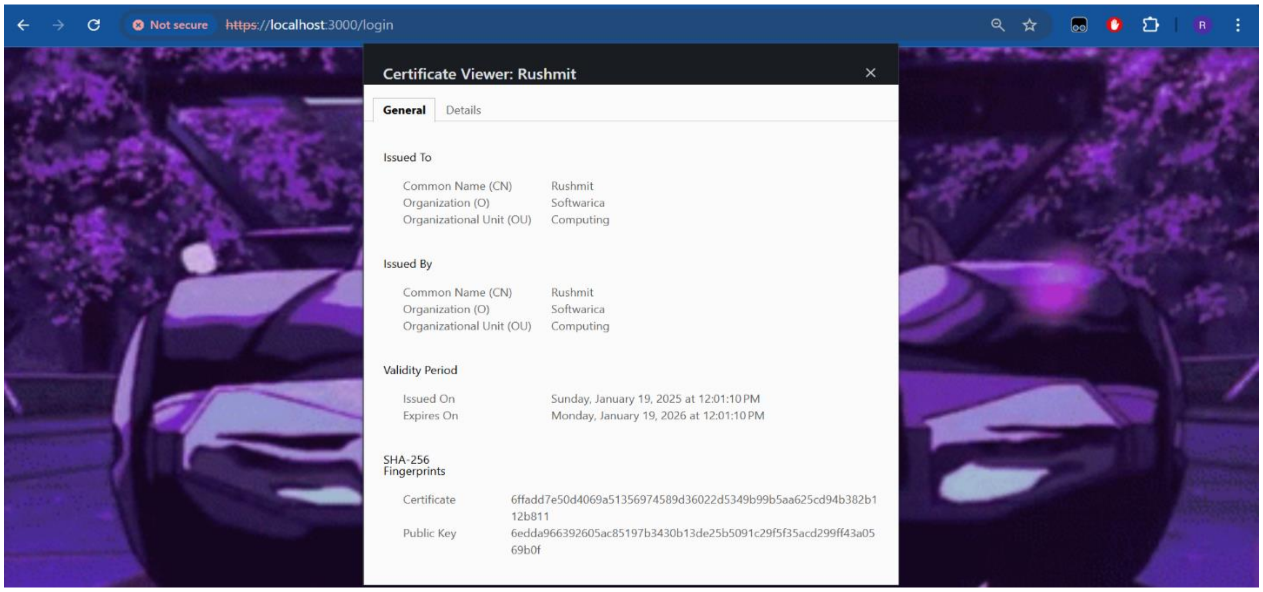
First Name: karki

Last Name: karki

Email: karkirushmit10@gmail.com

Phone Number: 9767274694

Figure 41:Other UI



Certificate Viewer: Rushmit

Issued To

Common Name (CN)	Rushmit
Organization (O)	Softwarica
Organizational Unit (OU)	Computing

Issued By

Common Name (CN)	Rushmit
Organization (O)	Softwarica
Organizational Unit (OU)	Computing

Validity Period

Issued On	Sunday, January 19, 2025 at 12:01:10 PM
Expires On	Monday, January 19, 2026 at 12:01:10 PM

SHA-256 Fingerprints

Certificate	6ffadd7e50d4069a51356974589d36022d5349b99b5aa625cd94b382b112b811
Public Key	6edda966392605ac85197b3430b13de25b5091c29f5f35acd299ff43a0569b0f

Figure 42:SSL Certificate