User Authentication and Management Code Explanation

Overview

This code defines several functions related to user authentication and management in a web application. It uses Node.js with Express for handling HTTP requests, Mongoose for MongoDB interactions, and JSON Web Tokens (JWT) for authentication. Here's a breakdown of each function:

1. Importing Required Modules

javascript
Copy code

```
import User from "../models/user.model.js";
import Otp from "../models/otp.model.js";
import sendSMS from "../utils/sendSMS.js";
import jwt from "jsonwebtoken";
const JWT_SECRET = process.env.JWT_SECRET || "your_jwt_secret";
```

- **User and Otp**: These are Mongoose models for interacting with the user and OTP (One Time Password) collections in the MongoDB database.
- **sendSMS**: A utility function to send SMS messages, likely used to send OTPs.
- **jwt**: A library for creating and verifying JSON Web Tokens.
- **JWT_SECRET**: A secret key used to sign JWTs, either from environment variables or a default value.

2. Sending OTP (One Time Password)

javascript
Copy code

```
export const sendOTP = async (req, res) => {
    try {
        const { phoneNumber } = req.body;
        const generatedOtp = Math.floor(100000 + Math.random() *
900000).toString();
        const newOtp = new Otp({ phoneNumber, otp: generatedOtp });
```

```
        await newOtp.save();
        const smsSent = await sendSMS(phoneNumber, generatedOtp);

        if (smsSent) {
            console.log(`Sending OTP ${generatedOtp} to phone number
${phoneNumber}`);
            return res.status(200).json({ message: "OTP sent
successfully" });
        } else {
            return res.status(500).json({ message: "Unable to send
OTP" });
        }
    } catch (error) {
        console.log(error);
        return res.status(500).json({
            message: "Something went wrong in send OTP post function",
            error: error.message,
        });
    }
};
```

Explanation:

- **Purpose**: This function generates and sends an OTP to a user's phone number.
- **Flow**:
    1. The phone number is extracted from the request body.
    2. A random 6-digit OTP is generated.
    3. An OTP record is created and saved in the database.
    4. The OTP is sent to the user via SMS.
    5. If successful, a success message is returned; otherwise, an error message is sent.

3. Verifying OTP

javascript
Copy code

```
export const verifyOTP = async (req, res) => {
    try {
        const { phoneNumber, otp } = req.body;
        const otpRecord = await Otp.findOne({ phoneNumber, otp });
```

```
        if (otpRecord) {
            await Otp.deleteMany({ phoneNumber });
            const user = await User.findOne({ phoneNumber });

            if (user) {
                const token = jwt.sign({ userId: user._id },
JWT_SECRET, { expiresIn: "30d" });
                return res.status(200).json({
                    message: "OTP verified successfully",
                    userExists: true,
                    token,
                });
            } else {
                return res.status(200).json({
                    message: "OTP verified successfully",
                    userExists: false,
                });
            }
        } else {
            console.log("Invalid OTP");
            return res.status(400).json({ message: "Invalid OTP" });
        }
    } catch (error) {
        console.log("Error in verifyOTP:", error);
        return res.status(500).json({
            message: "Something went wrong in verify OTP post
function",
            error: error.message,
        });
    }
};
```

Explanation:

- **Purpose**: This function checks if the provided OTP is valid and authenticates the user.
- **Flow**:
    1. The function checks the OTP against the database record.
    2. If valid, it deletes the used OTP and checks if the user exists.

3. A JWT token is generated if the user exists and sent back to the client; if not, it responds accordingly.

4. Registering a User

javascript
Copy code

```
export const registerUser = async (req, res) => {
    try {
        const { phoneNumber, name, email, address, landmark, pincode,
city, state } = req.body;
        const existingUser = await User.findOne({ phoneNumber });

        if (existingUser) {
            return res.status(400).json({ message: "User already
exists", user: existingUser });
        }

        const newUser = new User({
            phoneNumber,
            name,
            email,
            address,
            landmark,
            pincode,
            city,
            state,
        });

        await newUser.save();
        const token = jwt.sign({ userId: newUser._id }, JWT_SECRET, {
expiresIn: "30d" });
        return res.status(200).json({
            message: "User registered successfully",
            token,
        });
    } catch (error) {
        console.log(error);
```

```
        return res.status(500).json({ message: "Internal server error"
});
    }
};
```

Explanation:

- **Purpose**: This function registers a new user in the application.
- **Flow**:
    1. User details are extracted from the request body.
    2. The function checks if the user already exists.
    3. If not, a new user record is created and saved.
    4. A JWT token is generated and returned along with a success message.

5. Updating User Information

javascript
Copy code

```
export const updateUser = async (req, res) => {
    try {
        const token = req.headers.authorization?.split(" ")[1];
        if (!token) {
            console.log("No token in request");
            return res.status(401).json({ message: "Unauthorized-no
token in update user" });
        }

        const decoded = jwt.verify(token, JWT_SECRET);
        const userIdFromToken = decoded.userId;

        if (userIdFromToken !== req.params.userId) {
            return res.status(403).json({
                message: "Forbidden-user not authorized to update this
user",
            });
        }

        const { userId } = req.params;
        const updateData = req.body;
```

```
        let user = await User.findById(userId);

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

        Object.keys(updateData).forEach((key) => {
            if (updateData[key] !== undefined) {
                user[key] = updateData[key];
            }
        });

        user = await user.save();
        return res.status(200).json({ message: "User updated
successfully", user });
    } catch (error) {
        console.log(error);
        return res.status(500).json({ message: "Internal server
error", error: error.message });
    }
};
```

Explanation:

- **Purpose**: This function updates user information.
- **Flow**:
    1. The function extracts the token from the request headers and verifies it.
    2. It checks if the user is authorized to make the update based on the token.
    3. The user record is fetched and updated with new data.
    4. Finally, the updated user information is returned.

6. Getting User Information

javascript
Copy code

```
export const getUser = async (req, res) => {
    try {
        const token = req.headers.authorization?.split(" ")[1];
```

```
        if (!token) {
            console.log("No token in request");
            return res.status(401).json({ message: "Unauthorized-no
token in getUser" });
        }

        const decoded = jwt.verify(token, JWT_SECRET);
        const userId = decoded.userId;
        const user = await User.findById(userId);

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

        return res.status(200).json({ user });
    } catch (error) {
        console.log("Error in getUser:", error);
        return res.status(500).json({ message: "Internal server
error", error: error.message });
    }
};
```

Explanation:

- **Purpose**: This function retrieves the details of the authenticated user.
- **Flow**:
    1. The function checks for a valid token in the request headers.
    2. It decodes the token to get the user ID.
    3. The user details are fetched from the database and returned.

7. Logging Out User

javascript
Copy code

```
export const logoutUser = (req, res) => {
    res.status(200).json({ message: "Logged out successfully" });
};
```

Explanation:

- **Purpose**: This function handles user logout.
- **Flow**: It simply responds with a message indicating successful logout. (In a JWT-based system, logging out usually involves client-side token invalidation.)

8. Simulating Payment

javascript
Copy code

```
export const simulatePayment = async (req, res) => {
    const { userId } = req.body;
    try {
        const user = await
User.findById(userId).populate("cart.productId");
        if (!user) {
            return res.status(404).json({ message: "User not found"
});
        }
        const totalAmount = user.cart.reduce((total, item) => total +
item.productId.price * item.quantity, 0);

        return res.status(200).json({ message: "Payment successful",
totalAmount });
    } catch (error) {
        return res.status(500).json({ message: "Something went wrong
during payment", error: error.message });
    }
};
```

Explanation:

- **Purpose**: This function simulates a payment process.
- **Flow**:
    1. It retrieves the user's cart and calculates the total amount based on product prices.
    2. A success message with the total amount is returned.

Conclusion

This code provides a comprehensive user authentication and management system using Express, Mongoose, and JWT. Each function handles specific tasks related to user registration, login (via OTP), updating information, and retrieving user details, while ensuring proper error handling and response messages.