



---

**FAKULTI KEJURUTERAAN  
ELEKTRONIK DAN KEJURUTERAAN  
KOMPUTER**

**BENR 2423**

**DATABASE AND CLOUD SYSTEM**

**SEMESTER 2 SESSION 2022/23**

**ASSIGNMENT REPORT**

**SCHOOL VISITOR MANAGEMENT SYSTEM**

**LECTURER:**

**DR SOO YEW GUAN**

No.	Name	Matric No.	Section
1	MUHAMMAD AMRI TAJUDIN BIN NUKMAN	B022110178	2 BENR S2
2	MOHAMAD SAFWAN BIN MD NAZAR	B022110172	2 BENR S2
3	HAZIM FAHMI BIN MOHD SOFIAN	B022110047	2 BENR S2
4	MUHD ALIFF AIZAT BIN ABIDIN	B022110076	2 BENR S2
5	KHAIRUL HASNAWI BIN ABDUL RAHIM	B022110008	2 BENR S2

# **SCHOOL VISITOR MANAGEMENT SYSTEM**

## **INTRODUCTION**

This report provides a comprehensive overview of a School Visitor Management System implemented using Express.js and MongoDB. User registration, login, visitor management, visitor information retrieval, visitor data editing, visitor deletion, and check-in/check-out recording are just a few of the capabilities available on the system. To enable safe access control based on user responsibilities, the solution includes strong authentication and authorization techniques utilizing JSON Web Tokens (JWT).

### **1.0 OBJECTIVES**

The analysis of a user and visitor management system built with the Express.js framework and MongoDB is the main goal of this report. The system has features for managing user and visitor data as well as for user registration, login, and visitor registration, check-in, and administration.

### **2.0 PROBLEM STATEMENT**

A school's manual and ineffective guest control system causes a number of problems and security issues. To simplify visitor registration, increase security, and enhance the entire visitor experience, the school administration requires a digital solution. The objective is to build a School Visitor Management System that streamlines access control inside the school's boundaries, maintains accurate record-keeping, and automates visitor management procedures.

### **3.0 APPLICATION OVERVIEW**

User login, user registration, and visitor data are managed by the Node.js application, which acts as a backend. It has a number of endpoints, including '/login', '/register', '/addvisitors', '/visitorinfo', '/editvisitor/:id', '/deletevisitor/:id', and '/checkin', to make these activities easier. The program uses JSON Web Tokens (JWT) for authentication and authorisation, enabling access to certain routes for various roles (admin, user, security).

### 3.1 USER AUTHENTICATION

User authentication is handled through the '/login route'. A JWT token is produced and sent back to the client once the given username and password are checked against the 'dbUsers' array. The token contains the user's role information and is signed with a secret key.

### 3.2 USER REGISTRATION

An admin user can register new users by using the '/register' route. It performs data validation to make sure the username is unique and confirms the role of the requesting user. When a user successfully registers, their information is added to the 'dbUsers' array and the MongoDB users collection is updated using the 'updateUsersCollection' function. The function determines whether the user is already present in the collection and updates or adds the user as necessary.

### 3.3 VISITOR MANAGEMENT

Registered users can add visitor data via the '/addvisitors' route. The visitor's information is then added to the 'dbVisitors' array once the visitor's ID has been verified for uniqueness. In order to maintain consistency between the array and the database, the 'updateVisitorsCollection' method changes the visitors collection in MongoDB. An error message is issued if another visitor with the same ID already exists.

### 3.4 DATABASE INTEGRATION

For storing user and visitor data, the application uses the MongoDB database. Using the supplied connection URI, it creates a connection to the MongoDB cluster. The 'dbUsers' and 'dbVisitors' local arrays are synchronized with the corresponding collections in MongoDB using the 'updateUsersCollection' and 'updateVisitorsCollection' methods. The routines loop over the arrays, updating or inserting data as necessary based on whether each element already exists in the collection.

### 3.5 TOKEN-BASED AUTHENTICATION AND AUTHORIZATION

The application's authentication and authorization procedures heavily rely on JWT tokens. The JWT token that is included in the request headers is verified by the 'verifyToken' middleware method. It guarantees the integrity of the token, examines its expiration date, then decodes the payload to obtain the user's personal data and role. Certain routes are available depending on the user's job, offering role-based access control (RBAC) to safeguard critical functions.

### 3.6 ADDITIONAL FUNCTIONALITY

i. /visitorinfo:

Administrator and security roles can access visitor data from the visitors collection in MongoDB by using the '/visitorinfo' route. All visitor records are returned to admin users, whereas security users can only see visitor records that are linked to their account.

ii. /editvisitor/:id:

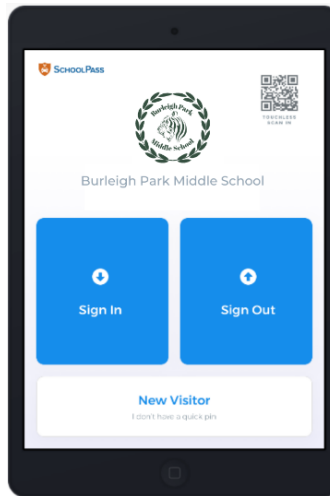
Users can change visitor information by providing the visitor's ID as a parameter on the '/editvisitor/:id' route. The route modifies the visitor's information in MongoDB's 'dbVisitors' array and visitors collection. In the event that the visitor cannot be located, the proper error message is sent back.

iii. /deletevisitor/:id:

Users can delete visitors via the '/deletevisitor/:id' route.

## 4.0 UI IMPLEMENTATION

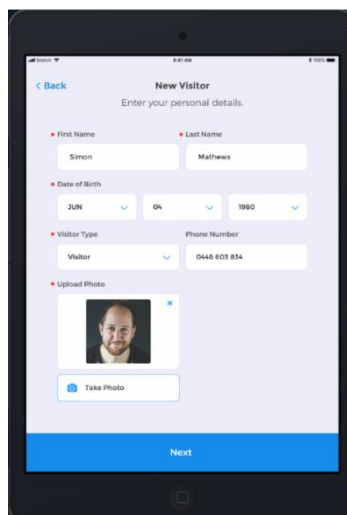
### 4.1 Login Page



*Figure 1.1 Login Page*

- i. In a school visitor system, implementing a login and authentication system based on roles guarantees that users have various kinds of access and permissions, boosting security and data privacy.
- ii. The main purpose of the school pass login page is to authenticate and validate the login information of authorised users, ensuring that only authorised users are permitted access to the system.

### 4.2 Visitor Detail Page



*Figure 1.2 Visitor Detail*

i. Basic Visitor Information:

This includes the visitor's name, contact details, organization or affiliation, and any identification information collected during the check-in process.

ii. Appointment Date Details:

The visitor detail page displays the date and time of the visitor's check-in and, if applicable, their expected check-out time. It provides a clear record of when the visitor arrived on the premises.

iii. Purpose and Destination:

The page may specify the purpose of the visit and the specific location or person the visitor is authorized to meet. This helps staff members ensure that visitors are directed to the correct areas and enhances security protocols.

iv. Additional Information:

This section allows about more detail visitor such as phone number, email, purpose, destination,

#### 4.3 Pass Identification Page



*Figure 1.3 Visitor Pass*

i. Monitoring visitor records by generating visitor passes to enhance security in school visitor management.

ii. Validation based on the visitor pass ensures authorized entry and facilitates quick identification of visitors, students, and staff.

#### 4.4 Admin Page

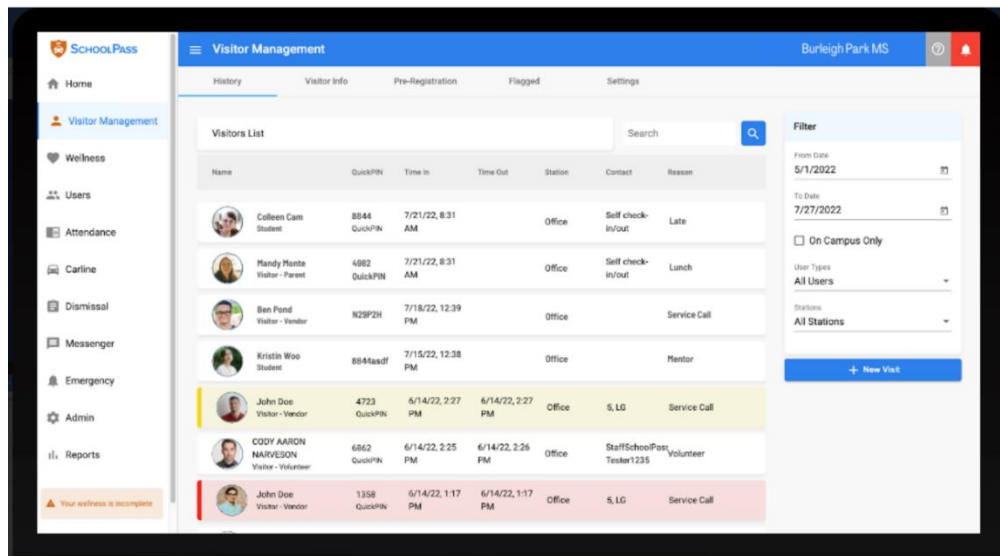


Figure 1.4 Admin view

- i. Managing user accounts and the process of registering new users in a system or application such as creating user profiles, collecting user information, verifying user credentials, and storing user data securely.
- ii. Actively tracking and gathering information about visitors who are currently visiting using data that uploaded to cloud system

## 5.0 ERD DIAGRAM

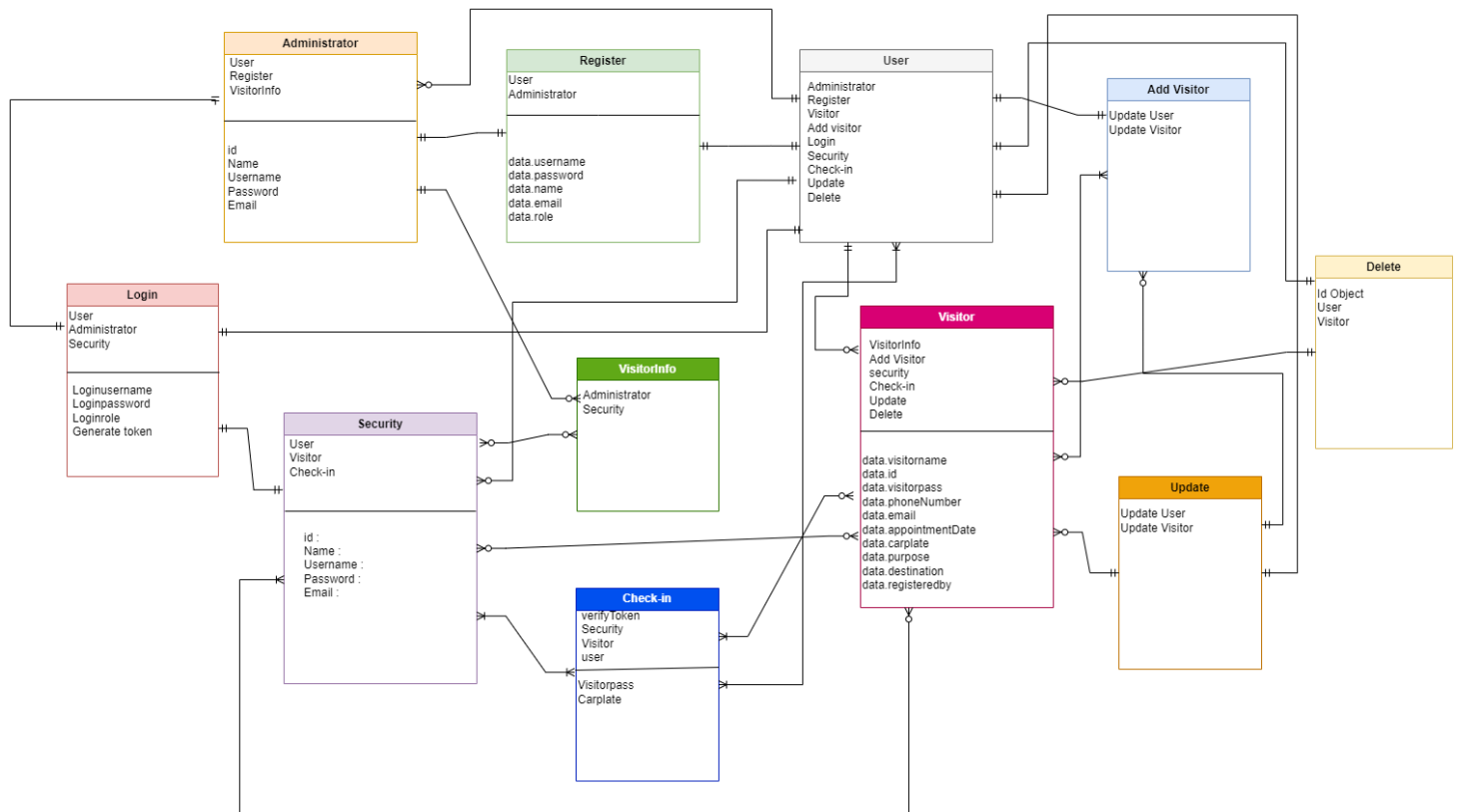


Figure 2.1 ERD Diagram for SCHOOL VISITOR MANAGEMENT SYSTEM



## 6.0 FLOWCHART OF THE SYSTEM

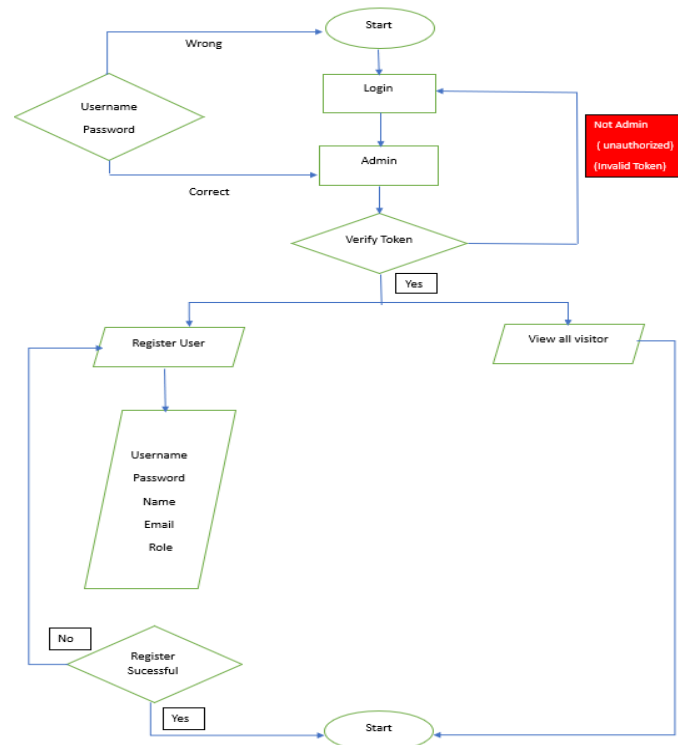


Figure 3.1 Admin Flowchart

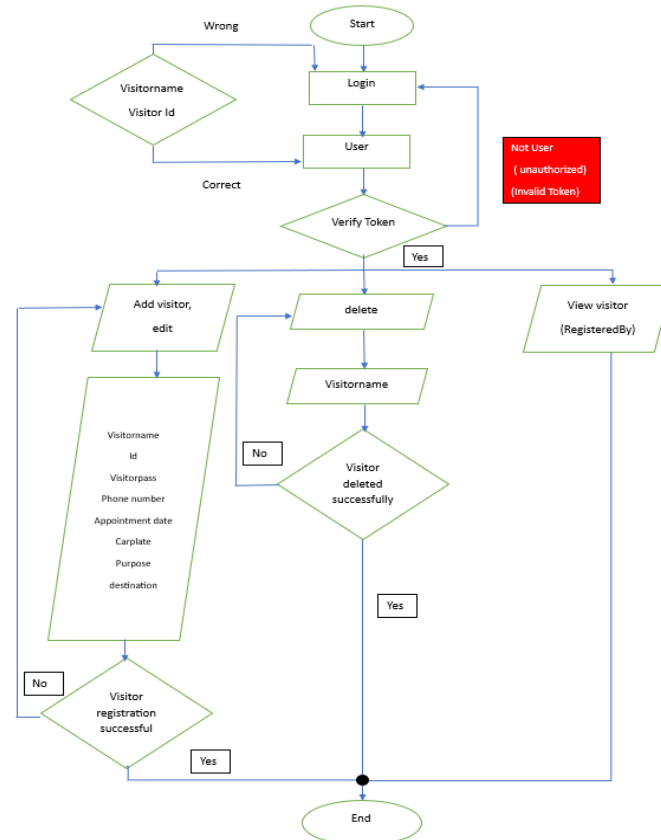
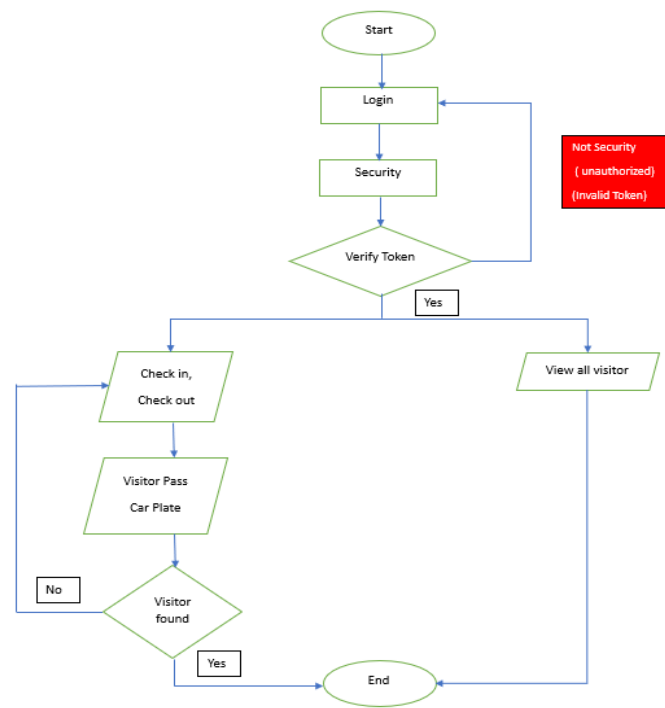


Figure 3.2 User Flowchart



*Figure 3.3 Security Flowchart*

## 7.0 CODE EXPLANATION

### 1. Initialize all required modules

```
const express = require('express');
const app = express();
const port = 5000;
const jwt = require('jsonwebtoken');
const { MongoClient, ServerApiVersion } = require('mongodb');
const uri = "mongodb+srv://alifjr763:QWE12345@cluster30.chmmxsq.mongodb.net/benr2423?retryWrites=true&w=majority";
const dbName = "benr2423";
const usersCollectionDB = "users";
const visitorsCollectionDB = "visitors";
const { ObjectId } = require('mongodb');
const moment = require('moment-timezone');

const client = new MongoClient(uri, {
  serverApi: {
    version: ServerApiVersion.v1,
    strict: true,
    deprecationErrors: true,
  }
});

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

Figure 4.1 Module and Client Connect

- Use Express JSON as middleware, JSON WebToken, and set the destination port.
- Connect to MongoDB Compass using username , password and cluster name.
- Database Name : BENR 2423 Collection Name: users & visitors.

### 2. Database Array

```
let dbUsers = [
  {
    username: "aliff08",
    password: "0987654321",
    name: "aliffaizat",
    email: "alifjr763@gmail.com",
    role : "user"
  },
  {
    username: "security",
    password: "123456789",
    name: "Pak Guard",
    email: "Pak guard.com",
    role : "security"
  },
  {
    username: "admin",
    password: "password",
    email : "admin@example.com",
    role: "admin"
  }
]

let dbVisitors = [
  {
    visitorname: "John Placebo",
    visitorpass: "john123",
    id: "871212053345",
    phoneNumber: "010202067543",
    email: "johnplacebo@example.com",
    appointmentDate: "2023-06-21",
    carPlate: "JL84102",
    purpose: "Majlis Convo",
    destination: "Dewan Seminar",
    registeredBy: "aliffaizat"
  },
  {
    visitorname: "Jenny Kim",
    visitorpass: "Jenny123",
    id: "090909048454",
    phoneNumber: "0987654321",
    email: "jenniebp@example.com",
    appointmentDate: "2023-06-22",
    carPlate: "XYZ2987",
    purpose: "Mesyuarat PIBG",
    destination: "Fakulti Mekanikal",
    registeredBy: "Albino Rafael"
  },
  // Add more visitors as needed
]
```

Figure 4.2 dbUser and dbVisitor

- Create two database array for user and visitors
- All the existing data for test purpose
- New user or visitor data will be added to this array before uploading to MongoDB

### 3. API POST request for login

```
app.post('/login', (req, res) => {
  let data = req.body;
  let user = login(data.username, data.password);

  if (user.role === 'admin') {
    res.send(generateToken(user, 'admin'));
  } else if (user.role === 'user') {
    res.send(generateToken(user, 'user'));
  } else if (user.role === 'security') {
    res.send(generateToken(user, 'security'));
  } else {
    res.send({ error: "User not found" });
  }
});
```

Figure 4.3 API Login

- For user to login, it must provide the data request by req.body which are username and password.
- In the login, the system will determine the user role based on the username and password
- User will be given a unique token according to their role upon successful login

### 4. API POST request for register

```
app.post('/register', verifyToken, async (req, res) => {
  if (req.user.role === 'admin') {
    let data = req.body;
    let username = data.username;
    let match = dbUsers.find(element => element.username === username);
    if (match) {
      res.send("Error! User already registered.");
    } else {
      let result = await register(
        data.username,
        data.password,
        data.name,
        data.email,
        data.role
      );
      if (result.status === 'Registration successful') {
        await updateUsersCollection(); // Update the users collection in MongoDB
      }
      res.send(result);
    }
  } else {
    res.send("Unauthorized");
  }
});
```

Figure 4.4 API Register

- Register request will be handled when user is admin role. To register a user, an authorization token needed to verify the access to do the task. If the token is false or incorrect, the response will send an unauthorized message.
- User need to provide username, password, name, email and most importantly their role.
- If the user have already registered, it will detect using elementmatch that find the same username to prevent from duplicate data.
- After admin finish to register new user, the new user data will be upload to MongoDB by using updateUsersCollection function.

## 5. API POST request for add visitor

```
app.post('/addvisitors', verifyToken, async (req, res) => {
  if (req.user.role === 'user') {
    let data = req.body;
    let id = data.id;
    let match = dbVisitors.find(element => element.idnumber === id);
    if (match) {
      res.send("Error! Visitor data already in the system.");
    } else {
      let result = await addvisitor(
        data.visitorname,
        data.id,
        data.visitorpass,
        data.phoneNumber,
        data.email,
        data.appointmentDate,
        data.carPlate,
        data.purpose,
        data.destination,
        data.registeredBy
      );
      if (result === 'Visitor registration successfull') {
        await updateVisitorsCollection(); // Update the visitors collection in MongoDB
      }
      res.send(result);
    }
  } else {
    res.send("Unauthorized");
  }
});
```

Figure 4.5 API addvisitors

- Addvisitors request will be handled by user and authorization token from user login process needed in order to access the task.
- The data required to addvisitor will be specified in req.body and a match function that filtered visitor idnumber added to prevent user from adding the existing visitor data.
- After successfull attempt, visitor data will be upload to MongoDB using updateVisitorsCollection function.

## 6. API GET request for view visitor information

```
app.get('/visitorinfo', verifyToken, async (req, res) => {
  try {
    // Connect to the MongoDB server
    await client.connect();

    if (req.user.role === 'admin' || req.user.role === 'security') {
      const visitorsCursor = client
        .db("benr2423")
        .collection("visitors")
        .find();
      const visitors = await visitorsCursor.toArray();
      res.send(visitors);
    } else if (req.user.role === 'user') {
      const visitorsCursor = client
        .db("benr2423")
        .collection("visitors")
        .find({ registeredBy: req.user.userProfile.name });
      const visitors = await visitorsCursor.toArray();
      res.send(visitors);
    } else {
      res.status(401).send('Unauthorized');
    }
  } catch (error) {
    console.error('Error retrieving visitor information:', error);
    res.status(500).send('Internal Server Error');
  } finally {
    // Close the MongoDB connection
    await client.close();
  }
});
```

Figure 4.6 API visitorinfo

- Get HTTP Method used to retrieve all visitor info from visitors collection in MongoDB.
- Admin and Security role can view all visitors info after verification token process
- If the users use their token when in user role, it can only find the visitor based on registeredBy.
- The visitor data will be displayed in Array

## 7. API PATCH request to edit visitor data

```
app.patch('/editvisitor/:id', verifyToken, async (req, res) => {
  const visitorId = req.params.id;
  const updateData = req.body;

  try {
    const visitorsCollection = client.db(dbName).collection(visitorsCollectionDB);

    if (!visitorId) {
      res.status(400).send('Invalid visitor ID');
      return;
    }

    const result = await visitorsCollection.findOneAndUpdate(
      { _id: new ObjectId(visitorId) },
      { $set: updateData },
      { returnOriginal: false }
    );

    if (!result.value) {
      res.status(404).send('Visitor not found');
    } else {
      await updateVisitorsCollection(); // Update the visitors collection in MongoDB
      res.send('Visitor info updated successfully');
    }
  } catch (error) {
    console.error('Error updating visitor info:', error);
    res.status(500).send('An error occurred while updating the visitor info');
  }
});
```

Figure 4.7 API editvisitor

- Patch HTTP method will be used for user to edit existing visitor data from MongoDB collection
- The req.params.id retrieves the visitor ID from the URL MongoDB
- The findOneAndUpdate() use find the visitor with the specified ID and update its data with the updateData object.
- The returnOriginal: false option ensures that the updated document is returned instead of the original document.

## 8. API DELETE request to delete visitor

```
app.delete('/deletevisitor/:id', verifyToken, async (req, res) => {
  if (req.user.role === 'user') {
    const visitorId = req.params.id;

    try {
      const visitorsCollection = client.db(dbName).collection(visitorsCollectionDB);
      const result = await visitorsCollection.deleteOne({ _id: new ObjectId(visitorId) });

      if (result.deletedCount === 0) {
        res.status(404).send('Visitor not found');
      } else {
        await updateVisitorsCollection(); // Update the visitors collection in MongoDB
        res.send('Visitor deleted successfully');
      }
    } catch (error) {
      console.error('Error deleting visitor:', error);
      res.status(500).send('An error occurred while deleting the visitor');
    }
  } else {
    res.status(403).send('Unauthorized');
  }
});
```

Figure 4.8 API deletevisitor

- Delete HTTP method used by user to delete visitor data by id that representing the ID of the visitor that want to be deleted.
- The method uses deleteOne() to find and delete the visitor document with the specified ID.
- If no visitor is found (result.deletedCount is 0), it sends status code with the message 'Visitor not found'.
- If the visitor is found and deleted successfully, it calls the updateVisitorsCollection() function to update the visitors collection in MongoDB.

## 9. API POST request for visitor check in

```
app.post('/checkin', verifyToken, async (req, res) => {
  if (req.user.role !== 'security') {
    return res.status(401).send('Unauthorized');
  }

  const { visitorpass, carplate } = req.body;
  const visitor = dbVisitors.find(visitor => visitor.visitorpass === visitorpass);

  if (!visitor) {
    return res.status(404).send('Visitor not found');
  }

  const gmt8Time = moment().tz('GMT+8').format('YYYY-MM-DD HH:mm:ss');
  visitor.checkinTime = gmt8Time;
  visitor.carPlate = carplate;

  // Insert or update the check-in data in the RecordTime collection
  try {
    await visitingtime(visitorpass, visitor.visitorname, visitor.checkinTime);
    res.send(`Check-in recorded for visitor: ${visitor.visitorname}
      Check-in time: ${visitor.checkinTime}
      Car plate number: ${carplate}`);
  } catch (error) {
    console.error('Error inserting/updating RecordTime:', error);
    res.status(500).send('Internal Server Error');
  }
});
```

Figure 4.9 API checkin

- Security can check in the visitor by retrieves the visitorpass and carplate from the existing visitor data.
- It searches for a visitor in the dbVisitors array where the visitorpass matches..
- The code retrieves the current time in GMT+8 timezone (Kuala Lumpur) and assigns it to gmt8Time in desired format.
- It updates the checkinTime and carPlate properties of the visitor and then update the check-in data in the RecordTime collection using the visitingtime() function.

## 10. API POST request for visitor checkout

```
app.post('/checkout', verifyToken, async (req, res) => {
  if (req.user.role !== 'security') {
    return res.status(401).send('Unauthorized');
  }

  const { visitorpass } = req.body;
  const visitor = dbVisitors.find(visitor => visitor.visitorpass === visitorpass);

  if (!visitor) {
    return res.status(404).send('Visitor not found');
  }

  if (!visitor.checkinTime) {
    return res.send('Visitor has not checked in');
  }

  const gmt8Time = moment().tz('GMT+8').format('YYYY-MM-DD HH:mm:ss');
  const checkinTime = moment(visitor.checkinTime, 'YYYY-MM-DD HH:mm:ss');
  const checkoutTime = moment(gmt8Time, 'YYYY-MM-DD HH:mm:ss');
  visitor.checkoutTime = gmt8Time;

  // Update the check-out time in the RecordTime collection
  try {
    await visitingtime(visitorpass, visitor.visitorname, visitor.checkinTime, visitor.checkoutTime);
    res.send('Checkout recorded for visitor: ${visitor.visitorname}');
  } catch (error) {
    console.error('Error inserting/updating RecordTime:', error);
    res.status(500).send('Internal Server Error');
  }
});
```

Figure 4.10 API checkout

- After visitor finish visiting , Security will update check out time by using the same visitor pass and carPlate.
- It checks if the visitor has a check-in time recorded and returns a error response if the visitor has not checked in.
- It updates the visitor's checkout time with the GMT+8 timezone in desired format
- The 'visitingtime' function will be call to update the checkout time in the RecordTime collection.

## 11. Function login

```
function login(loginuser, loginpassword) {
  console.log("Someone is logging in!", loginuser, loginpassword); // Display message
  const user = dbUsers.find(user => user.username === loginuser && user.password === loginpassword);
  if (user) {
    return user;
  } else {
    return { error: "User not found" };
  }
}
```

Figure 4.11 Function Login

- Login function require user to insert username and password . After user login, it will generate a token based on user role
- Find operator searches for a user in the dbUsers array where the username and password match the provided request data.
- If a matching user is found, it returns the user login detail. Otherwise, it returns an message indicating that the user was not found



## 12. Function register

```
function register(newusername, newpassword, newname, newemail, newrole) {
  let match = dbUsers.find(element => element.username === newusername);
  if (match) {
    return "Error! Username is already taken.";
  } else {
    const newUser = {
      username: newusername,
      password: newpassword,
      name: newname,
      email: newemail,
      role: newrole
    };
    dbUsers.push(newUser);
    return {
      status: "Registration successful!",
      user: newUser
    };
  }
}
```

Figure 4.12 Function Register

- The register function takes newusername, newpassword, newname, newemail, and newrole as parameters request to register a new user.
- It checks using match query if there is already a user in the dbUsers array with the same newusername. If a match is found, it returns an error message indicating that the username is already taken.
- If no match is found, it creates a new user object with the provided information and adds it to the dbUsers array.

## 13. Function addvisitors

```
function addvisitor(name, id, visitorpass, phoneNumber, email, appointmentDate, carPlate, purpose, destination,
  , registeredBy) {
  // Check if the visitor with the same ID already exists
  let match = dbVisitors.find(element => element.idnumber === id);
  if (match) {
    return "Error! Visitor data already in the system.";
  } else {
    // Check if the visitorpass meets the required format
    const passRegex = new RegExp("^(name)\\d(4)$");
    if (!passRegex.test(visitorpass)) {
      return "Error! Invalid visitorpass. It should be a combination of 'visitorname' and 4 numbers.";
    }
    dbVisitors.push({
      visitorname: name,
      visitorpass: visitorpass,
      idnumber: id,
      phoneNumber: phoneNumber,
      email: email,
      date: appointmentDate,
      carPlate: carPlate,
      purpose: purpose,
      destination: destination,
      registerBy: registeredBy
    });
    return "Visitor registration successful!";
  }
}
```

Figure 4.13 Function Add visitor

- Using the addvisitor function, you can enter a new visitor's information into the system, such as visitor name, ID, visitor pass, contact information and appointment date.
- It initially determines whether the dbVisitors array already has a visitor with the same ID. If a match is made, an error message stating that the visitor information is already in the system is returned.
- The visitor pass is then verified to make sure it adheres to the proper format using RegExp that request a visitor pass with combination of visitor name and 4 digit number.
- The visitor's information is then added to the dbVisitors array and a success message is returned if all the field are correct.

## 14. Async function update user to MongoDB

```
async function updateUsersCollection() {
  try {
    const usersCollection = client.db(dbName).collection(usersCollectionDB);

    for (const user of dbUsers) {
      const existingUser = await usersCollection.findOne({ _id: user._id });

      if (existingUser) {
        await usersCollection.updateOne(
          { _id: user._id },
          { $set: user },
          { upsert: true }
        );
      } else {
        user._id = new ObjectId(); // Generate a new ObjectId
        await usersCollection.insertOne(user);
      }
    }

    console.log('Users collection updated successfully');
  } catch (error) {
    console.error('Error updating users collection:', error);
  }
}
```

Figure 4.14 Function Update User

- Asynchronous updateUsersCollection function use the user data in the dbUsers array to update the users collection in MongoDB.
- Every user in the dbUsers array is iterated over, and a check is made to see if the user is already in the MongoDB users collection.
- In that case, it adds the user's information to the existing user document. A new document is inserted for the user if no existing user data found.

## 15. Async function update visitors to MongoDB

```
async function updateVisitorsCollection() {
  try {
    const visitorsCollection = client.db(dbName).collection(visitorsCollectionDB);

    for (const visitor of dbVisitors) {
      const existingVisitor = await visitorsCollection.findOne({ _id: visitor._id });

      if (existingVisitor) {
        await visitorsCollection.updateOne(
          { _id: visitor._id },
          { $set: visitor },
          { upsert: true }
        );
      } else {
        visitor._id = new ObjectId(); // Generate a new ObjectId
        await visitorsCollection.insertOne(visitor);
      }
    }

    console.log('Visitors collection updated successfully');
  } catch (error) {
    console.error('Error updating visitors collection:', error);
  }
}
```

Figure 4.15 Function Update visitors

- The updateVisitorsCollection function is an asynchronous function for updating the visitors collection in MongoDB based on the data in the dbVisitors array.
- It find through each visitor by id in the dbVisitors array and checks if the visitor already exists in the MongoDB visitors collection.
- If the visitor exists, it will use updateOne query to update the existing visitor document with the visitor's data. Otherwise, it inserts a new document for the visitor.

## 16. Async function update visitor checkin/out to MongoDB

```
async function visitingTime(visitorPass, visitorName, checkinTime, checkoutTime) {
  try {
    await client.connect();
    const db = client.db(dbName);
    const RecordCollectionDB = db.collection('RecordTime');
    // Check if the visitor record already exists
    const existingRecord = await RecordCollectionDB.findOne({ visitorpass: visitorPass });

    if (existingRecord) {
      // Update the existing record with the visitor name and checkout time
      await RecordCollectionDB.updateOne(
        { visitorpass: visitorPass },
        { $set: { visitorName: visitorName, checkoutTime: checkoutTime } }
      );
      console.log('RecordTime updated successfully');
    } else {
      // Create a new document for the visitor
      const document = {
        visitorpass: visitorPass,
        visitorName: visitorName,
        checkinTime: checkinTime,
        checkoutTime: checkoutTime
      };
      // Insert the document
      await RecordCollectionDB.insertOne(document);
      console.log('RecordTime inserted successfully');
    }
    // Close the connection
    client.close();
  } catch (error) {
    console.error('Error inserting/updating RecordTime:', error);
  }
}
```

Figure 4.16 Function Update User

- The visitingtime function is an asynchronous function responsible for inserting or updating visitor records in the RecordTime collection in MongoDB.
- Firstly, it checks if the visitor record already exists based on the visitor pass then updates the existing record with the visitor's name and checkout time. Otherwise, it creates a new document for the visitor with check-in and check-out details.
- The function logs a success message if the record is inserted or updated successfully or outputs an error message if an error occurs.

## 17. Function generate and verify token

```
function generateToken(userProfile, role) {
  const payload = {
    userProfile,
    role
  };
  return jwt.sign(payload, 'access_token', { expiresIn: 30 * 60 });
}

function verifyToken(req, res, next) {
  let header = req.headers.authorization;
  let token = header.split(' ')[1];

  jwt.verify(token, 'access_token', function (err, decoded) {
    if (err) {
      res.send("Invalid Token");
    } else {
      req.user = decoded;
      next();
    }
  });
}
```

Figure 4.17 Function Generate and Verify Token

- This function is to generate a JSON Web Token (JWT) for authentication and authorization purposes.
- It creates a payload object containing the userProfile and role.
- The jwt.sign function is called with the payload, a secret key ('access\_token'), and an expiration time of 30 minutes to generate the token.
- A JWT sent in the request header can be authenticated and validated using the middleware function verifyToken.
- The 'Bearer' prefix is removed before splitting the token once it is extracted from the authorization header.
- Finally, the jwt.verify method will validate and decode the token then proceed to add the decoded token to the req.user property.

## 18. Start the server

```
app.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
});
```

*Figure 4.18 Starting the server*

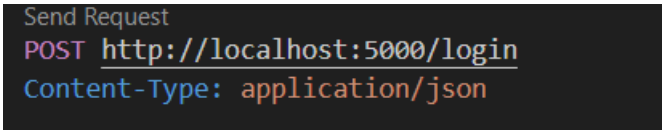
- The app.listen function is called to start the application and listen on a specified port which is set to 5000
- It takes a callback function that will be executed once the server is running to allow the application to start listening for incoming requests on the specified port

## 8.0 API SEQUENCE DIAGRAM

An API sequence diagram represents the flow of communication and the order of actions taken by various entities. The sequence diagram in this instance begins with the User submitting an API request via the Client Application. Next, an HTTP request is made to the API by the client application. To check the user's credentials, the API converses with an authentication service. The authentication outcome is returned to the API by the authentication service. The required data is then retrieved from a database through the API. The Database replies by returning to the API the user information that was requested. The API then returns the data to the client application through an HTTP response, and the client application shows the API response to the user.

- i. For login user:

POST <http://localhost:5000/login>  
CLIENT SERVER

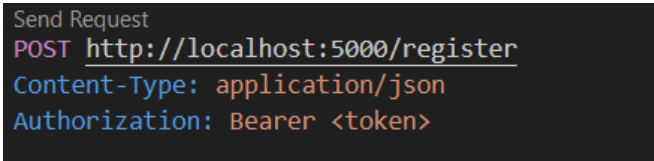


```
Send Request
POST http://localhost:5000/login
Content-Type: application/json
```

*Figure 5.1 Login request*

- ii. For register a user

POST <http://localhost:5000/register>  
CLIENT SERVER

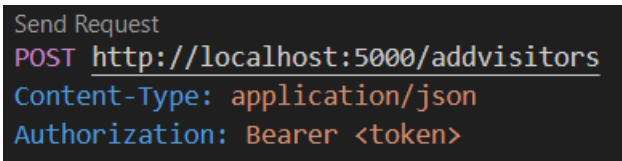


```
Send Request
POST http://localhost:5000/register
Content-Type: application/json
Authorization: Bearer <token>
```

*Figure 5.2 Register request*

- iii. For add visitor by user:

POST <http://localhost:5000/addvisitors>  
CLIENT SERVER

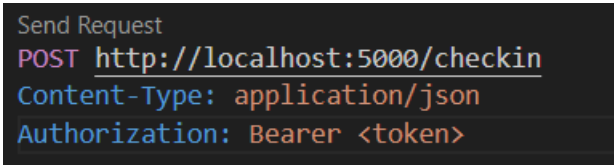


```
Send Request
POST http://localhost:5000/addvisitors
Content-Type: application/json
Authorization: Bearer <token>
```

*Figure 5.3 Addvisitors request*

- iv. For visitor checkin by security:

POST <http://localhost:5000/checkin>  
CLIENT SERVER



```
Send Request
POST http://localhost:5000/checkin
Content-Type: application/json
Authorization: Bearer <token>
```

*Figure 5.4 Checkin request*

- v. For visitor checkout by security:

POST <http://localhost:5000/checkout>  
CLIENT SERVER  
Send Request  
POST <http://localhost:5000/checkout>  
Content-Type: application/json  
Authorization: Bearer <token>

*Figure 5.5 Checkout request*

- vi. For view all visitor by admin or security

GET <http://localhost:5000/visitorinfo>  
CLIENT SERVER  
Send Request  
GET <http://localhost:5000/visitorinfo>  
Authorization: Bearer <token>

*Figure 5.6 Visitorinfo request*

- vii. For edit data by user:

PATCH <http://localhost:5000/editvisitor/visitorObjectId>  
CLIENT SERVER  
Send Request  
PATCH <http://localhost:5000/editvisitor/visitorObjectId>  
Content-Type: application/json  
Authorization: Bearer <token>

*Figure 5.7 Editvisitor request*

- viii. For delete data by user:

DELETE <http://localhost:5000/deletevisitor/VisitorObjectId>  
CLIENT SERVER  
Send Request  
DELETE <http://localhost:5000/deletevisitor/visitorObjectId>  
Content-Type: application/json  
Authorization: Bearer <token>

*Figure 5.8 Deletevisitor request*

## 9.0 RESULT

Client HTTP	Response
<p>1. Login user</p> <pre>Send Request POST http://localhost:5000/login Content-Type: application/json  {     "username": "aliffø8",     "password": "0987654321", }</pre>	<p>Successful login</p> <pre>HTTP/1.1 200 OK X-Powered-By: Express Content-Type: text/html; charset=utf-8 Content-Length: 313 ETag: W/"139-tl4lhptKZfOOLmzDxRDgcZZmgUCB" Date: Thu, 29 Jun 2023 12:23:45 GMT Connection: close  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpvcCJ9.eyJ1c2VybmFtZWVudHlwZXNlcjEwYmlkLWVmbS1jaGVzaWkiOnRhdGF1eXBvbnQzMTEyOTg0MTAweTJleHAoImZldGEpdnpyMjYwLnUpfEno_ciolIUkwY7QSMpie3baqapoxjjzgihv7dls</pre> <p>Wrong username or password</p> <pre>HTTP/1.1 200 OK X-Powered-By: Express Content-Type: application/json; charset=utf-8 Content-Length: 26 ETag: W/"1a-hq/hTØORGtkTFyrPVCZ/JB/r8EG" Date: Thu, 29 Jun 2023 12:26:33 GMT Connection: close  ✓{     "error": "User not found" }</pre>
<p>2. Login Admin</p> <pre>Send Request POST http://localhost:5000/login Content-Type: application/json  {     "username": "admin",     "password": "password", }</pre>	<p>Successful login</p> <pre>HTTP/1.1 200 OK X-Powered-By: Express Content-Type: text/html; charset=utf-8 Content-Length: 281 ETag: W/"119-lkBxEIucw0KHnz7ubr@jkHFUGLw" Date: Thu, 29 Jun 2023 12:24:33 GMT Connection: close  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpvcCJ9.eyJ1c2VybmFtZWVudHlwZXNlcjEwYmlkLWVmbS1jaGVzaWkiOnRhdGF1eXBvbnQzMTEyOTg0MTAweTJleHAoImZldGEpdnpyMjYwLnUpfEno_ciolIUkwY7QSMpie3baqapoxjjzgihv7dls</pre> <p>Wrong username or password</p> <pre>HTTP/1.1 200 OK X-Powered-By: Express Content-Type: application/json; charset=utf-8 Content-Length: 26 ETag: W/"1a-hq/hTØORGtkTFyrPVCZ/JB/r8EG" Date: Thu, 29 Jun 2023 12:26:33 GMT Connection: close  ✓{     "error": "User not found" }</pre>

### 3. Login Security

```
Send Request
POST http://localhost:5000/login
Content-Type: application/json
```

```
{
  "username": "security",
  "password": "123456789",
}
```

## Sucessful Login

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 315
ETag: W/"13b-ZzJZCJAKUgp1kZe9UwPOZN+CI"
Date: Thu, 29 Jun 2023 12:25:24 GMT
Connection: close
```

**Wrong username or password**

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 26
ETag: W/"1a-hq/hT0ORGKtFyrPVcz/JB/r8Eg"
Date: Thu, 29 Jun 2023 12:26:33 GMT
Connection: close
```

```
✓ {
  "error": "User not found"
}
```

#### 4. Register User By Admin

```
Send Request
POST http://localhost:5000/register
Content-Type: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiI
```

```
{
  "username": "Azzz",
  "password": "nzzzz",
  "name": "Albino Rafael",
  "email": "newuser@example.com",
  "role": "user"
}
```

## Invalid token

```
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: text/html; charset=utf-8
4 Content-Length: 13
5 ETag: W/"d-GLqvnw/10/MwANK/40hPBcb7NQs"
6 Date: Thu, 29 Jun 2023 12:34:41 GMT
7 Connection: close
8
9 Invalid Token
```

### Unauthorized ( Not admin token)

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 12
ETag: W/"c-dAuDFQrdjS3hezqxDTNgW7A0lyk"
Date: Thu, 29 Jun 2023 12:35:24 GMT
Connection: close
```

Unauthorized



### Username already exist in database

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 31
ETag: W/"1f-1iGZ96zYdMRuqWSDi5ovDLf15vE"
Date: Thu, 29 Jun 2023 12:46:30 GMT
Connection: close

Error! User already registered.
```

### Successful register

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 183
ETag: W/"b7-SHREJagm5FRgm2mTN2xqdrwKesc"
Date: Thu, 29 Jun 2023 12:36:27 GMT
Connection: close

{
  "status": "Registration successful!",
  "user": {
    "username": "Azzz",
    "password": "nzzzz",
    "name": "Albino Rafael",
    "email": "newuser@example.com",
    "role": "user",
    "_id": "649d7acbbfe7196cddf24c05"
  }
}
```

### Data uploaded to MongoDB

```
▶ _id: ObjectId('649d7acbbfe7196cddf24c05')
  username: "Azzz"
  password: "nzzzz"
  name: "Albino Rafael"
  email: "newuser@example.com"
  role: "user"
```

## 5. Add visitor by users

```
Send Request
POST http://localhost:5000/addvisitors
Content-Type: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cGU6IjY4bGZ9

{
  "visitorname": "ayam",
  "id": "740612020453",
  "visitorpass": "ayam1234",
  "phoneNumber": "0112456356",
  "email": "Ruben@gmail.com",
  "appointmentDate": "2023-06-23",
  "carPlate": "KER8195",
  "purpose": "Football Training",
  "destination": "Tasik UTeM",
  "registeredBy": "aliffaizat"
}
```

### Invalid token

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 13
ETag: W/"d-GLqvNW/10/MWaNK/40hPBcb7NQs"
Date: Thu, 29 Jun 2023 12:40:11 GMT
Connection: close

Invalid Token
```

```
{
  "visitorname": "ayam",
  "id": "740612020453",
  "visitorpass": "ay1234",
  "phoneNumber": "0112456356",
  "email": "Ruben@gmail.com",
  "appointmentDate": "2023-06-23",
  "carPlate": "KER8195",
  "purpose": "Football Training",
  "destination": "Tasik UTeM",
  "registeredBy": "aliffaizat"
}
```

## Unauthorized ( Not User role)

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 12
ETag: W/"c-dAuDFQrdjs3hezqxDTNgw7A0lyk"
Date: Thu, 29 Jun 2023 12:42:09 GMT
Connection: close

Unauthorized
```

## Invalid visitorpass

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 86
ETag: W/"56-A0VaAPnVQ+oh6XbeKY9rGchK3lc"
Date: Thu, 29 Jun 2023 12:43:30 GMT
Connection: close

Error! Invalid visitorpass. It should be a combination of 'visitorname'
and 4 numbers.
```

## Sucessful add visitor

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 32
ETag: W/"20-WbQEZPowpVid5zDsrShA7Ia80Jw"
Date: Thu, 29 Jun 2023 12:44:21 GMT
Connection: close

Visitor registration successful!
```

## Uploaded to MongoDB

```
▶ {
  _id: ObjectId('649d7ca5bfe7196cddf24c08'),
  visitorname: "ayam",
  visitorpass: "ayam1234",
  idnumber: "740612020453",
  phoneNumber: "0112456356",
  email: "Ruben@gmail.com",
  date: "2023-06-23",
  carPlate: "KER8195",
  purpose: "Football Training",
  destination: "Tasik UTeM",
  registerBy: "aliffaizat"
}
```

## 6. Check In visitor by Security

Send Request

```
POST http://localhost:5000/checkin
Content-Type: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR
```

```
{
  "visitorpass": "ayam1234",
  "carplate": "KER8195"
}
```

```
{
  "visitorpass": "ay1234",
  "carplate": "KER8195"
}
```

### Invalid token

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 13
ETag: W/"d-GLqvnw/10/MWaNK/40hPBcb7NQs"
Date: Thu, 29 Jun 2023 12:57:44 GMT
Connection: close

Invalid Token
```

### Unauthorized ( Not Security role )

```
HTTP/1.1 401 Unauthorized
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 12
ETag: W/"c-dAuDFQrdjS3hezqxDTNgW7A0LYk"
Date: Thu, 29 Jun 2023 12:59:01 GMT
Connection: close

Unauthorized
```

### Wrong visitorpass / carplate

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 17
ETag: W/"11-4rldQyN4Ax1LSr2smcDeIrbIUoU"
Date: Thu, 29 Jun 2023 13:00:13 GMT
Connection: close

Visitor not found
```

### Sucessful visitor check in

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 108
ETag: W/"6c-lN48NIh50Up3eit9mg6lxtvWpT0"
Date: Thu, 29 Jun 2023 13:03:26 GMT
Connection: close

Check-in recorded for visitor: ayam
Check-in time: 2023-06-29 21:03:26
Car plate number: KER8195
```

	<p><b>Uploaded to MongoDB</b></p> <pre> _id: ObjectId('649d811ebfe7196cddf24c09') visitorpass: "ayam1234" visitorName: "ayam" checkinTime: "2023-06-29 21:03:26" checkoutTime: null </pre>
<p><b>7. Checkout visitor by Security</b></p> <div data-bbox="73 813 681 1158"> <p>Send Request</p> <p><b>POST</b> <u>http://localhost:5000/checkout</u></p> <p><b>Content-Type:</b> application/json</p> <p><b>Authorization:</b> Bearer eyJhbGciOiJIUzI1</p> <pre> {   "visitorpass": "ayam1234",   "carplate": "KER8195" } </pre> </div> <div data-bbox="73 1659 568 1825"> <pre> {   "visitorpass": "ay1234",   "carplate": "KER8195" } </pre> </div>	<p><b>Invalid token</b></p> <div data-bbox="911 813 1458 1149"> <pre> HTTP/1.1 401 Unauthorized X-Powered-By: Express Content-Type: text/html; charset=utf-8 Content-Length: 12 ETag: W/"c-dAuDFQrdjS3hezqxDTNgW7A0lYk" Date: Thu, 29 Jun 2023 12:59:01 GMT Connection: close  Unauthorized </pre> </div> <p><b>Unauthorized ( Not Security role )</b></p> <div data-bbox="911 1234 1458 1568"> <pre> HTTP/1.1 401 Unauthorized X-Powered-By: Express Content-Type: text/html; charset=utf-8 Content-Length: 12 ETag: W/"c-dAuDFQrdjS3hezqxDTNgW7A0lYk" Date: Thu, 29 Jun 2023 12:59:01 GMT Connection: close  Unauthorized </pre> </div> <p><b>Wrong visitorpass / carplate</b></p> <div data-bbox="879 1653 1490 2007"> <pre> HTTP/1.1 404 Not Found X-Powered-By: Express Content-Type: text/html; charset=utf-8 Content-Length: 17 ETag: W/"11-4r1DqyN4Ax1LSr2smcDeIrbIUoU" Date: Thu, 29 Jun 2023 13:00:13 GMT Connection: close  Visitor not found </pre> </div>



```
_id: ObjectId('64994ee23c9d03924720c246')  
visitorname: "John Placebo"  
visitorpass: "john123"  
id: "871212053345"  
phoneNumber: "010202067543"  
email: "johnplacebo@example.com"  
appointmentDate: "2023-06-21"  
carPlate: "JLB4102"  
purpose: "Majlis Convo"  
destination: "Dewan Seminar"  
registeredBy: "aliffaizat"
```

## Wrong visitor Object ID

```
HTTP/1.1 500 Internal Server Error
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 49
ETag: W/"31-KY6G5uw2U32b1cNsueVI4mecuqw"
Date: Thu, 29 Jun 2023 13:18:05 GMT
Connection: close

An error occurred while updating the visitor info
```

## Sucessful edit visitor data

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 33
ETag: W/"21-pLcVTWn+34MxcmVIrod1/xRFEMo"
Date: Thu, 29 Jun 2023 13:21:07 GMT
Connection: close

Visitor info updated successfully
```

## Updated to MongoDB

```
_id: ObjectId('64994ee23c9d03924720c246')
visitorname: "Ahmad Ahmad"
visitorpass: "john123"
id: "871212053345"
phoneNumber: "85555555555555555555"
email: "johnplacebo@example.com"
appointmentDate: "2025-10-22"
carPlate: "JLB4102"
purpose: "Majlis Convo"
destination: "Dewan Seminar"
registeredBy: "aliffaizat"
```

## 9. Delete visitor by users

```
Send Request  
DELETE http://localhost:5000/deletevisitor/64994ee23c9d03924720c246  
Content-Type: application/json  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm
```

## Invalid token

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 13
ETag: W/"d-GLQvnw/10/MWaNK/40hPBcb7NQs"
Date: Thu, 29 Jun 2023 13:16:02 GMT
Connection: close

Invalid Token
```

<pre>_id: ObjectId('64994ee23c9d03924720c246') visitorname: "Ahmad Ahmad" visitorpass: "john123" id: "871212053345" phoneNumber: "85555555555555555555" email: "johnplacebo@example.com" appointmentDate: "2025-10-22" carPlate: "JLB4102" purpose: "Majlis Convo" destination: "Dewan Seminar" registeredBy: "aliffaizat"</pre>	<div><h3>Unauthorized ( Not User role )</h3><pre>HTTP/1.1 401 Unauthorized X-Powered-By: Express Content-Type: text/html; charset=utf-8 Content-Length: 12 ETag: W/"c-dAuDFQrdjS3hezqxDTNgW7AOLYk" Date: Thu, 29 Jun 2023 12:59:01 GMT Connection: close  Unauthorized</pre></div> <div><h3>Wrong visitor Object ID</h3><pre>HTTP/1.1 500 Internal Server Error X-Powered-By: Express Content-Type: text/html; charset=utf-8 Content-Length: 49 ETag: W/"31-KY6G5uw2U32b1cNSueVI4mecuqw" Date: Thu, 29 Jun 2023 13:18:05 GMT Connection: close  An error occurred while updating the visitor info</pre></div> <div><h3>Sucessful delete visitor data</h3><pre>HTTP/1.1 200 OK X-Powered-By: Express Content-Type: text/html; charset=utf-8 Content-Length: 28 ETag: W/"1c-zvys/NvIKnJVQ7n2XTpKK532pCA" Date: Thu, 29 Jun 2023 13:29:26 GMT Connection: close  Visitor deleted successfully</pre></div> <div><h3>Updated to MongoDB</h3><pre>_id: ObjectId('64995073470ee3687b5f5bf1') visitorname: "John Placebo" visitorpass: "john123" id: "871212053345" phoneNumber: "010202067543" email: "johnplacebo@example.com" appointmentDate: "2023-06-21" carPlate: "JLB4102" purpose: "Majlis Convo" destination: "Dewan Seminar" registeredBy: "aliffaizat"</pre></div>
--	--

## 10. Retrieved visitor data by admin,security or users

[illegible]

## Invalid token

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 13
ETag: W/"d-GLqvNW/10/MWaNK/40hPBcb7NQs"
Date: Thu, 29 Jun 2023 13:16:02 GMT
Connection: close

Invalid Token
```

### Wrong inserting registeredBy (user)

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 17
ETag: W/"11-4r1DqyN4AxlLSr2smcDeIrbIUoU"
Date: Thu, 29 Jun 2023 13:00:13 GMT
Connection: close

Visitor not found
```

**Sucessful retrieve all visitor data (admin/security)**

```
{
  "_id": "649d7ca5bfe7196cddf24c08",
  "visitorname": "ayam",
  "visitorpass": "ayam1234",
  "idnumber": "740612020453",
  "phoneNumber": "01124563656",
  "email": "Ruben@gmail.com",
  "date": "2023-06-23",
  "carPlate": "KER8195",
  "purpose": "Football Training",
  "destination": "Tasik UTeM",
  "registerBy": "aliffaizat"
},
{
  "_id": "649d854377fe4a6797bf8160",
  "visitorname": "John Placebo",
  "visitorpass": "john123",
  "id": "871212053345",
  "phoneNumber": "010202067543",
  "email": "johnplacebo@example.com",
  "appointmentDate": "2023-06-21",
  "carPlate": "JLB4102",
  "purpose": "Majlis Convo",
  "destination": "Dewan Seminar",
  "registerdBy": "aliffaizat"
},
{
```



	<p><b>Successful retrieve user visitor data</b></p> <pre>{   "_id": "64995073470ee3687b5f5bf1",   "visitorname": "John Placebo",   "visitorpass": "john123",   "id": "871212053345",   "phoneNumber": "010202067543",   "email": "johnplacebo@example.com",   "appointmentDate": "2023-06-21",   "carPlate": "JLB4102",   "purpose": "Majlis Convo",   "destination": "Dewan Seminar",   "registeredBy": "aliffaizat" },</pre> <p><b>All the visitor data in MongoDB</b></p> <pre>_id: ObjectId('649d7ca5bfe7196cddf24c07') visitorname: "Jenny Kim" visitorpass: "Jenny123" id: "890909048454" phoneNumber: "0987654321" email: "jenniep@example.com" appointmentDate: "2023-06-22" carPlate: "XYZ2987" purpose: "Mesyuarat PIBG" destination: "Fakulti Mekanikal" registeredBy: "Albino Rafael"</pre> <hr/> <pre>_id: ObjectId('649d7ca5bfe7196cddf24c08') visitorname: "ayam" visitorpass: "ayam1234" idnumber: "740612820453" phoneNumber: "0112456356" email: "Ruben@gmail.com" date: "2023-06-23" carPlate: "KER8195" purpose: "Football Training" destination: "Tasik UTeM" registerBy: "aliffaizat"</pre>
--	---

*Table 1 REST Client and Response*

## 10. FUTURE IMPROVEMENT

Looking towards room of improvements, there are several key areas in the School Visitor Management System that can be further enhanced:

- ❖ **Enhanced Visitor Experience**

Develop a user-friendly interfaces such as mobile application with enhanced visitor experience by incorporating elements like pre-registration, appointment scheduling, efficient check-in/out procedure and personalised visitor passes.

- ❖ **Advanced Security Measures**

To ensure accurate visitor identification, include reliable identity verification technology, such as biometric scanners or ID card readers. For more proactive identification, use real-time visitor monitoring technology like facial recognition and GPS tracking, and sync with a cloud database to detect potential threats to enhance overall security.

- ❖ **Integration with other School Systems**

Improving efficiency by synchronizing all data, such as updating visitor information and event details on a regular basis to ensure that the current information is correct in addition to reducing workload through integration with all school systems such as students and staff directories and Event Management System.

## 11. CONCLUSION

In conclusion, it can be stated that the school visit system offers a solution to improve the visitor management process of educational institutions. With multiple security features and measures, it enhances the overall visitor experience while keeping the school community safe. The system provides many benefits, including shorter wait times and self-service options that are accessible to all types of users. The system is open to upgradability with integration of advanced technological methods, such as facial identity recognition systems and synchronization with other school systems, to help automate the process, reduce administrative workload, and ensure accurate and up-to-date information exchange. A reliable, user-friendly and efficient solution for managing school visitors can be developed by continuously applying technological advances and using feedback from users of the school visitor system.

## APPENDIX

### NODE.JS Code

```
const express = require('express');
const app = express();
const port = 5000;
const jwt = require('jsonwebtoken');
const { MongoClient, ServerApiVersion } = require('mongodb');
const uri =
  "mongodb+srv://alifjr763:QWE12345@cluster30.chmmxsq.mongodb.net/be
  nr2423?retryWrites=true&w=majority";
const dbName = "benr2423";
const usersCollectionDB = "users";
const visitorsCollectionDB = "visitors";
const { ObjectId } = require('mongodb');
const moment = require('moment-timezone');

const client = new MongoClient(uri, {
  serverApi: {
    version: ServerApiVersion.v1,
    strict: true,
    deprecationErrors: true,
  }
});

app.use(express.json());
let dbUsers = [
  {
    username: "aliff08",
    password: "0987654321",
    name: "aliffaizat",
    email: "alifjr763@gmail.com",
    role : "user"
  },
  {
    username: "security",
    password: "123456789",
    name: "Pak Guard",
    email: "Pak guard.com",
    role : "security"
  },
  {
    username: "admin",
    password: "password",
    email : "admin@example.com",
```

```

    role: "admin"
  }
]
let dbVisitors = [
  {
    visitorname: "John Placebo",
    visitorpass: "john123",
    id: "871212053345",
    phoneNumber: "010202067543",
    email: "johnplacebo@example.com",
    appointmentDate: "2023-06-21",
    carPlate: "JLB4102",
    purpose: "Majlis Convo",
    destination:"Dewan Seminar",
    registeredBy: "aliffaizat"
  },
  {
    visitorname: "Jenny Kim",
    visitorpass: "Jenny123",
    id: "090909048454",
    phoneNumber: "0987654321",
    email: "jenniebp@example.com",
    appointmentDate: "2023-06-22",
    carPlate: "XYZ2987",
    purpose: "Mesyuarat PIBG",
    destination:"Fakulti Mekanikal",
    registeredBy: "Albino Rafael"
  },
  // Add more visitors as needed
];

client.connect().then(() => {
  console.log('Connected to MongoDB');

app.post('/login', (req, res) => {
  let data = req.body;
  let user = login(data.username, data.password);

  if (user.role === 'admin') {
    res.send(generateToken(user, 'admin'));
  } else if (user.role === 'user') {
    res.send(generateToken(user, 'user'));
  } else if (user.role === 'security') {
    res.send(generateToken(user, 'security'));
  } else {
    res.send({ error: "User not found" });
  }
});
});

```

```

app.post('/register', verifyToken, async (req, res) => {
  if (req.user.role === 'admin') {
    let data = req.body;
    let username = data.username;
    let match = dbUsers.find(element => element.username ===
username);
    if (match) {
      res.send("Error! User already registered.");
    } else {
      let result = await register(
        data.username,
        data.password,
        data.name,
        data.email,
        data.role
      );
      if (result.status === 'Registration successful!') {
        await updateUsersCollection(); // Update the users
collection in MongoDB
      }
      res.send(result);
    }
  } else {
    res.send("Unauthorized");
  }
});

app.post('/addvisitors', verifyToken, async (req, res) => {
  if (req.user.role === 'user') {
    let data = req.body;
    let id = data.id;
    let match = dbVisitors.find(element => element.idnumber ===
id);
    if (match) {
      res.send("Error! Visitor data already in the system.");
    } else
    {
      let result = await addvisitor(
        data.visitorname,
        data.id,
        data.visitorpass,
        data.phoneNumber,
        data.email,
        data.appointmentDate,
        data.carPlate,
        data.purpose,
        data.destination,

```

```

        data.registeredBy
    );
    if (result === 'Visitor registration successful!') {
        await updateVisitorsCollection(); // Update the visitors
collection in MongoDB
    }
    res.send(result);
}
} else {
    res.send("Unauthorized");
}
});

app.get('/visitorinfo', verifyToken, async (req, res) => {
    try {
        // Connect to the MongoDB server
        await client.connect();

        if (req.user.role === 'admin' || req.user.role === 'security')
        {
            const visitorsCursor = client
                .db("benr2423")
                .collection("visitors")
                .find();
            const visitors = await visitorsCursor.toArray();
            res.send(visitors);
        } else if (req.user.role === 'user') {
            const visitorsCursor = client
                .db("benr2423")
                .collection("visitors")
                .find({ registeredBy: req.user.userProfile.name });
            const visitors = await visitorsCursor.toArray();
            res.send(visitors);
        } else {
            res.status(401).send('Unauthorized');
        }
    } catch (error) {
        console.error('Error retrieving visitor information:', error);
        res.status(500).send('Internal Server Error');
    } finally {
        // Close the MongoDB connection
        await client.close();
    }
});

app.patch('/editvisitor/:id', verifyToken, async (req, res) => {
    const visitorId = req.params.id;
    const updateData = req.body;

```

```

    try {
      const visitorsCollection =
client.db(dbName).collection(visitorsCollectionDB);

      if (!visitorId) {
        res.status(400).send('Invalid visitor ID');
        return;
      }

      const result = await visitorsCollection.findOneAndUpdate(
        { _id: new ObjectId(visitorId) },
        { $set: updateData },
        { returnOriginal: false }
      );

      if (!result.value) {
        res.status(404).send('Visitor not found');
      } else {
        await updateVisitorsCollection(); // Update the visitors
collection in MongoDB
        res.send('Visitor info updated successfully');
      }
    } catch (error) {
      console.error('Error updating visitor info:', error);
      res.status(500).send('An error occurred while updating the
visitor info');
    }
  });

app.delete('/deletevisitor/:id', verifyToken, async (req, res) =>
{
  if (req.user.role === 'user') {
    const visitorId = req.params.id;

    try {
      const visitorsCollection =
client.db(dbName).collection(visitorsCollectionDB);
      const result = await visitorsCollection.deleteOne({ _id: new
ObjectId(visitorId) });

      if (result.deletedCount === 0) {
        res.status(404).send('Visitor not found');
      } else {
        await updateVisitorsCollection(); // Update the visitors
collection in MongoDB
        res.send('Visitor deleted successfully');
      }
    }
  }
}

```

```

    } catch (error) {
      console.error('Error deleting visitor:', error);
      res.status(500).send('An error occurred while deleting the
visitor');
    }
  } else {
    res.status(403).send('Unauthorized');
  }
});

app.post('/checkin', verifyToken, async (req, res) => {
  if (req.user.role !== 'security') {
    return res.status(401).send('Unauthorized');
  }

  const { visitorpass, carplate } = req.body;
  const visitor = dbVisitors.find(visitor => visitor.visitorpass
=== visitorpass);

  if (!visitor) {
    return res.status(404).send('Visitor not found');
  }

  const gmt8Time = moment().tz('GMT+8').format('YYYY-MM-DD
HH:mm:ss');
  visitor.checkinTime = gmt8Time;
  visitor.carPlate = carplate;

  // Insert or update the check-in data in the RecordTime
collection
  try {
    await visitingtime(visitorpass, visitor.visitorname,
visitor.checkinTime);
    res.send(`Check-in recorded for visitor:
${visitor.visitorname}
    Check-in time: ${visitor.checkinTime}
    Car plate number: ${carplate}`);
  } catch (error) {
    console.error('Error inserting/updating RecordTime:', error);
    res.status(500).send('Internal Server Error');
  }
});

app.post('/checkout', verifyToken, async (req, res) => {
  if (req.user.role !== 'security') {
    return res.status(401).send('Unauthorized');
  }
}

```



```

const { visitorpass } = req.body;
const visitor = dbVisitors.find(visitor => visitor.visitorpass
=== visitorpass);

if (!visitor) {
  return res.status(404).send('Visitor not found');
}

if (!visitor.checkinTime) {
  return res.send('Visitor has not checked in');
}

const gmt8Time = moment().tz('GMT+8').format('YYYY-MM-DD
HH:mm:ss');
const checkinTime = moment(visitor.checkinTime, 'YYYY-MM-DD
HH:mm:ss');
const checkoutTime = moment(gmt8Time, 'YYYY-MM-DD HH:mm:ss');
visitor.checkoutTime = gmt8Time;

// Update the check-out time in the RecordTime collection
try {
  await visitingtime(visitorpass, visitor.visitorname,
visitor.checkinTime, visitor.checkoutTime);
  res.send(`Checkout recorded for visitor:
${visitor.visitorname}
Checkout time: ${visitor.checkoutTime}`);
} catch (error) {
  console.error('Error inserting/updating RecordTime:', error);
  res.status(500).send('Internal Server Error');
}
});

function login(loginuser, loginpassword) {
  console.log("Someone is logging in!", loginuser, loginpassword);
// Display message
  const user = dbUsers.find(user => user.username === loginuser &&
user.password === loginpassword);
  if (user) {
    return user;
  } else {
    return { error: "User not found" };
  }
}

function register(newusername, newpassword, newname,
newemail,newrole) {
  let match = dbUsers.find(element => element.username ===
newusername);

```

```

    if (match) {
        return "Error! Username is already taken.";
    } else {
        const newUser = {
            username: newusername,
            password: newpassword,
            name: newname,
            email: newemail,
            role: newrole
        };
        dbUsers.push(newUser);
        return {
            status: "Registration successful!",
            user: newUser
        };
    }
}

function addvisitor(name, id, visitorpass, phoneNumber, email,
appointmentDate, carPlate, purpose, destination
, registeredBy) {
    // Check if the visitor with the same ID already exists
    let match = dbVisitors.find(element => element.idnumber === id);
    if (match) {
        return "Error! Visitor data already in the system.";
    } else {
        // Check if the visitorpass meets the required format
        const passRegex = new RegExp(`^${name}\\d{4}$`);
        if (!passRegex.test(visitorpass)) {
            return "Error! Invalid visitorpass. It should be a
combination of 'visitorname' and 4 numbers.";
        }

        dbVisitors.push({
            visitorname: name,
            visitorpass: visitorpass,
            idnumber: id,
            phoneNumber: phoneNumber,
            email: email,
            date: appointmentDate,
            carPlate: carPlate,
            purpose: purpose,
            destination: destination,
            registerBy: registeredBy
        });
        return "Visitor registration successful!";
    }
}

```

```

async function updateUserCollection() {
  try {
    const usersCollection =
client.db(dbName).collection(usersCollectionDB);

    for (const user of dbUsers) {
      const existingUser = await usersCollection.findOne({ _id:
user._id });

      if (existingUser) {
        await usersCollection.updateOne(
          { _id: user._id },
          { $set: user },
          { upsert: true }
        );
      } else {
        user._id = new ObjectId(); // Generate a new ObjectId
        await usersCollection.insertOne(user);
      }
    }

    console.log('Users collection updated successfully');
  } catch (error) {
    console.error('Error updating users collection:', error);
  }
}

async function updateVisitorsCollection() {
  try {
    const visitorsCollection =
client.db(dbName).collection(visitorsCollectionDB);

    for (const visitor of dbVisitors) {
      const existingVisitor = await visitorsCollection.findOne({
_id: visitor._id });

      if (existingVisitor) {
        await visitorsCollection.updateOne(
          { _id: visitor._id },
          { $set: visitor },
          { upsert: true }
        );
      } else {
        visitor._id = new ObjectId(); // Generate a new ObjectId
        await visitorsCollection.insertOne(visitor);
      }
    }
  }
}

```

```

        console.log('Visitors collection updated successfully');
    } catch (error) {
        console.error('Error updating visitors collection:', error);
    }
}

async function visitingtime(visitorPass, visitorName, checkinTime,
checkoutTime) {
    try {
        await client.connect();
        const db = client.db(dbName);
        const RecordCollectionDB = db.collection('RecordTime');
        // Check if the visitor record already exists
        const existingRecord = await RecordCollectionDB.findOne({
visitorpass: visitorPass });

        if (existingRecord) {
            // Update the existing record with the visitor name and
checkout time
            await RecordCollectionDB.updateOne(
                { visitorpass: visitorPass },
                { $set: { visitorName: visitorName, checkoutTime:
checkoutTime } }
            );
            console.log('RecordTime updated successfully');
        } else {
            // Create a new document for the visitor
            const document = {
                visitorpass: visitorPass,
                visitorName: visitorName,
                checkinTime: checkinTime,
                checkoutTime: checkoutTime
            };
            // Insert the document
            await RecordCollectionDB.insertOne(document);
            console.log('RecordTime inserted successfully');
        }
        // Close the connection
        client.close();
    } catch (error) {
        console.error('Error inserting/updating RecordTime:', error);
    }
}

function generateToken(userProfile, role) {
    const payload = {
        userProfile,
        role
    };

```

```
};  
return jwt.sign(payload, 'access_token',  
  { expiresIn: 30 * 60 });  
}  
  
function verifyToken(req, res, next) {  
  let header = req.headers.authorization;  
  let token = header.split(' ')[1];  
  
  jwt.verify(token, 'access_token', function (err, decoded) {  
    if (err) {  
      res.send("Invalid Token");  
    } else {  
      req.user = decoded;  
      next();  
    }  
  });  
}  
  
app.listen(port, () => {  
  console.log(`Example app listening on port ${port}`);  
});  
  
}).catch((error) => {  
  console.error('Error connecting to MongoDB:', error);  
});
```