Full Stack Development with MERN

API Development and Integration Report

Date	8 July 2024
Team ID	SWTID1720170691
Project Name	Flight Booking APP
Maximum Marks	10

Project Title: Flight Booking APP
Date: 09-07-2024
Team: S Koushik, Surya Teja A, Chandrakanth J, Anurag Reddy A

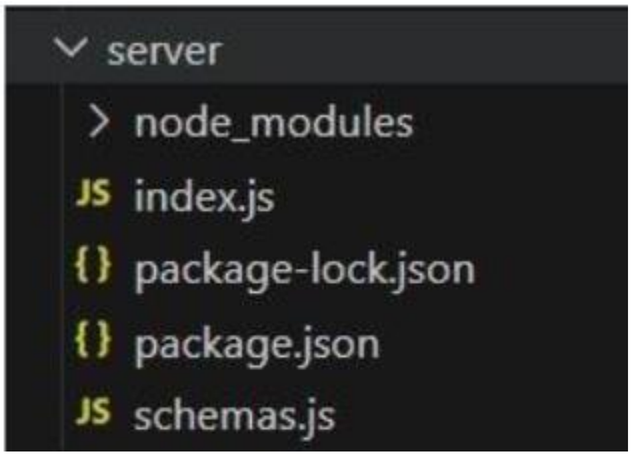
Objective

The objective of this report is to document the API development progress and key aspects of the backend services implementation for the Flight Booking project.

Technologies Used

- **Backend Framework:** Node.js with Express.js
- **Database:** MongoDB
- **Authentication:** JWT

Project Structure



```
server
  node_modules
  index.js
  package-lock.json
  package.json
  schemas.js
```

project-root/

|

├── node_modules/ **Directory containing all the installed npm packages.**

├── index.js **Entry point for the Node.js application where the server is initialized**.

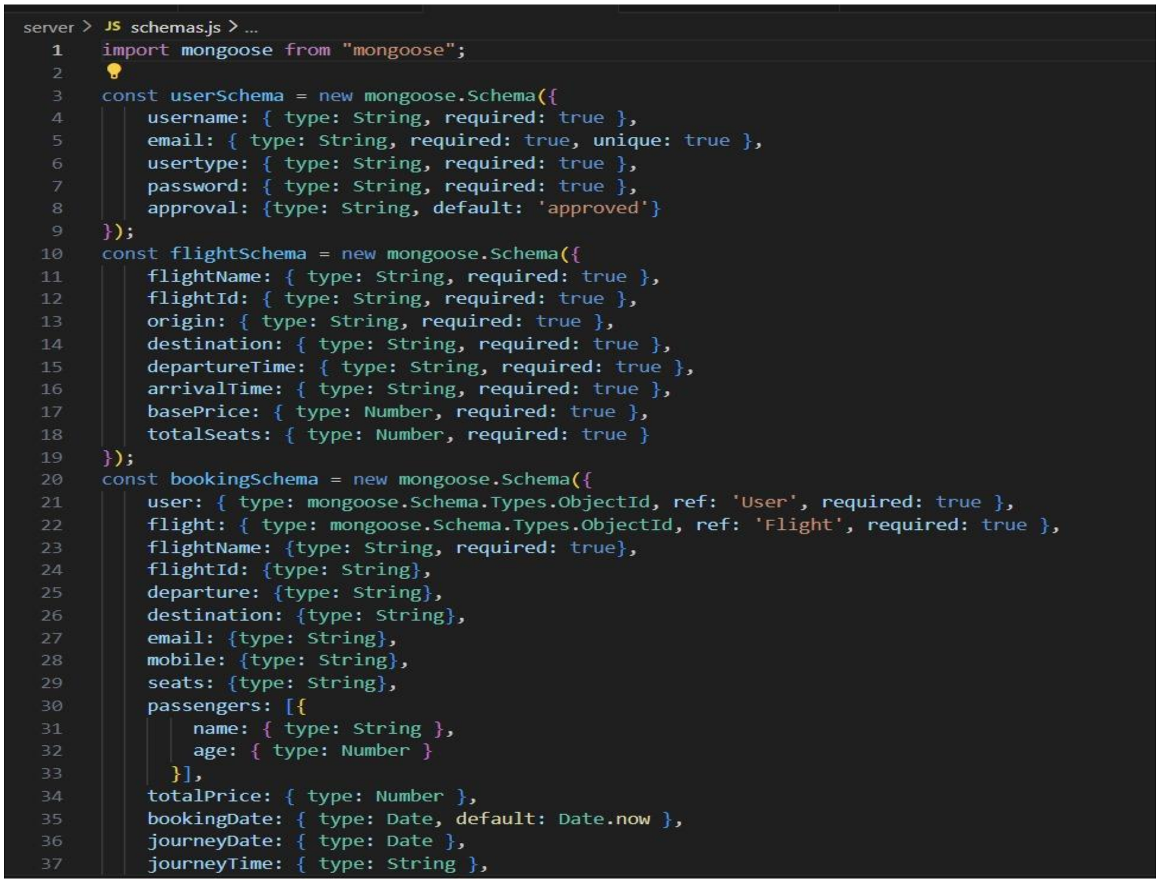
├── package-lock.json **Automatically generated file that records the exact versions of installed npm dependencies.**

├── package.json **Configuration file that lists project details and dependencies.**

└── schemas.js **File containing Mongoose schema definitions for the database models.**

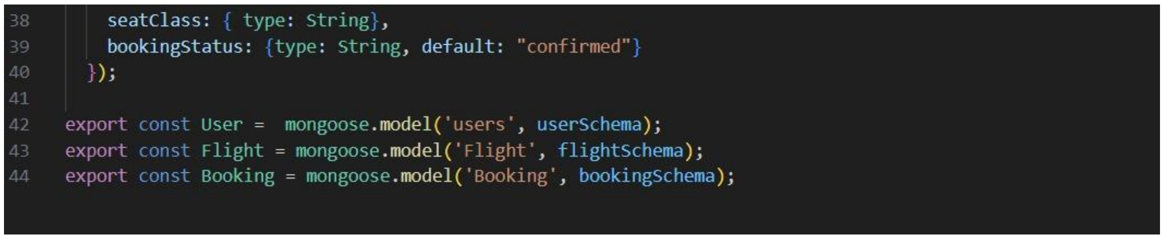
Key Directories and Files

1. **/schema.js**



server > JS schemas.js > ...

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
    username: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    usertype: { type: String, required: true },
    password: { type: String, required: true },
    approval: {type: String, default: 'approved'}
});
const flightSchema = new mongoose.Schema({
    flightName: { type: String, required: true },
    flightId: { type: String, required: true },
    origin: { type: String, required: true },
    destination: { type: String, required: true },
    departureTime: { type: String, required: true },
    arrivalTime: { type: String, required: true },
    basePrice: { type: Number, required: true },
    totalSeats: { type: Number, required: true }
});
const bookingSchema = new mongoose.Schema({
    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
    flight: { type: mongoose.Schema.Types.ObjectId, ref: 'Flight', required: true },
    flightName: {type: String, required: true},
    flightId: {type: String},
    departure: {type: String},
    destination: {type: String},
    email: {type: String},
    mobile: {type: String},
    seats: {type: String},
    passengers: [{
        name: { type: String },
        age: { type: Number }
    }],
    totalPrice: { type: Number },
    bookingDate: { type: Date, default: Date.now },
    journeyDate: { type: Date },
    journeyTime: { type: String },
```

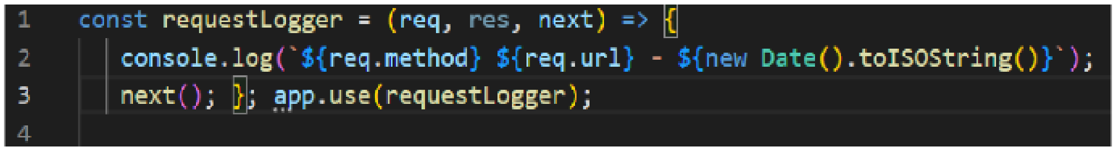


```
    seatClass: { type: String},
    bookingStatus: {type: String, default: "confirmed"}
  });

export const User =  mongoose.model('users', userSchema);
export const Flight = mongoose.model('Flight', flightSchema);
export const Booking = mongoose.model('Booking', bookingSchema);
```

2. /index.js

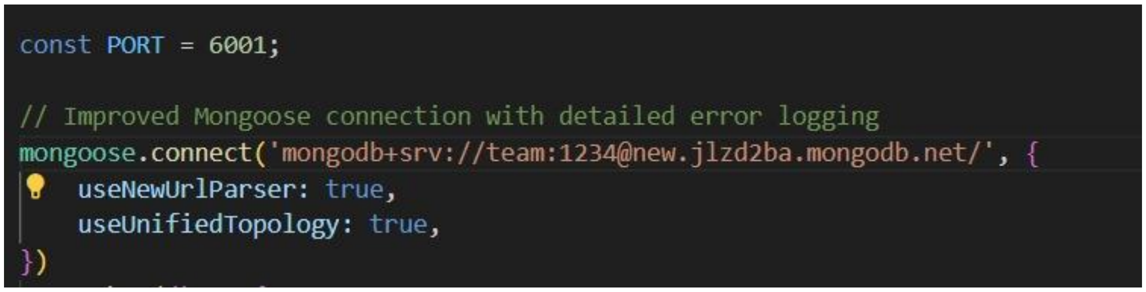
1. **/controllers**
   - User Controller
   - Flights Controller
   - Booking Controller

2. **/Middlewares**
   - Custom middleware functions for request processing.



```
const requestLogger = (req, res, next) => {
    console.log(`${req.method} ${req.url} - ${new Date().toISOString()}`);
    next(); }; app.use(requestLogger);
```

3. **/config**



```
const PORT = 6001;

// Improved Mongoose connection with detailed error logging
mongoose.connect('mongodb+srv://team:1234@new.jlzd2ba.mongodb.net/', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
})
```

API Endpoints

A summary of the main API endpoints and their purposes for the flight booking application:

User Authentication

- **POST /api/user/register** - Registers a new user.
- **POST /api/user/login** - Authenticates a user and returns a token.

User Management

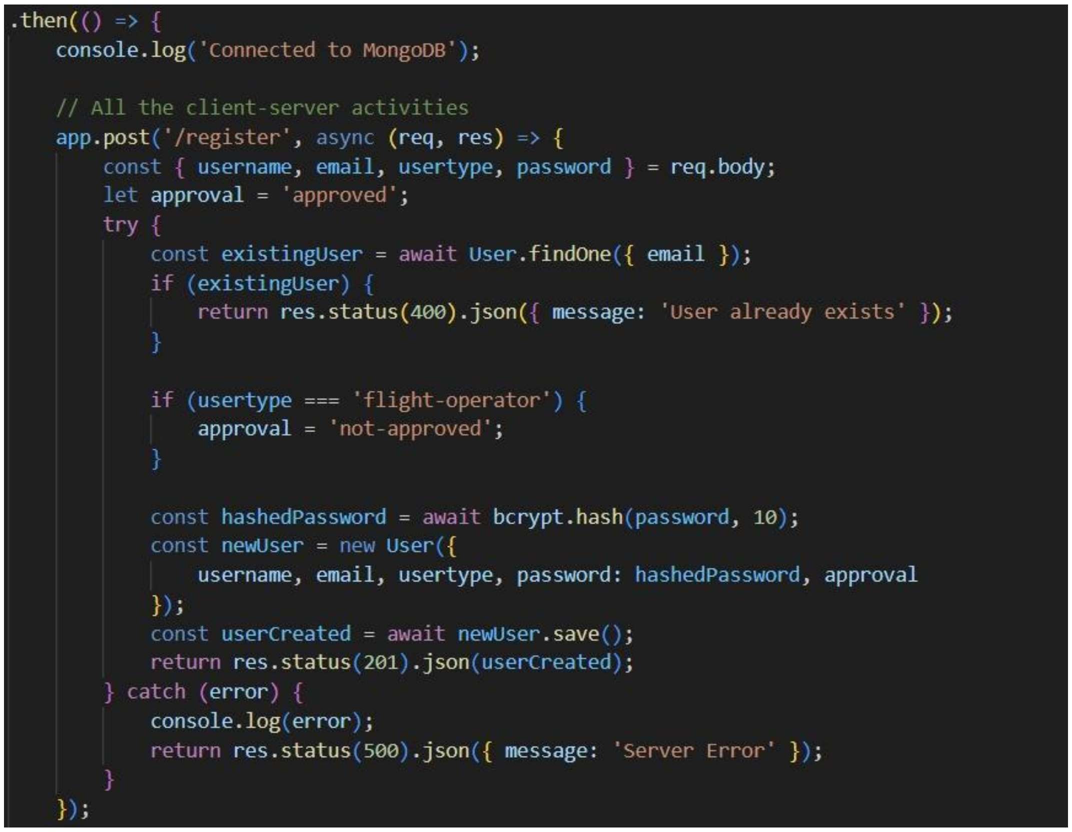
- **GET /api/user/{id}** - Retrieves user information by ID.

Flight Booking

- **GET /api/flights** - Retrieves all available flights.
- **POST /api/flights/book** - Books a flight.
- **GET /api/flights/{id}** - Retrieves flight details by ID.
- **PUT /api/flights/{id}/cancel** - Cancels a booked flight by ID.

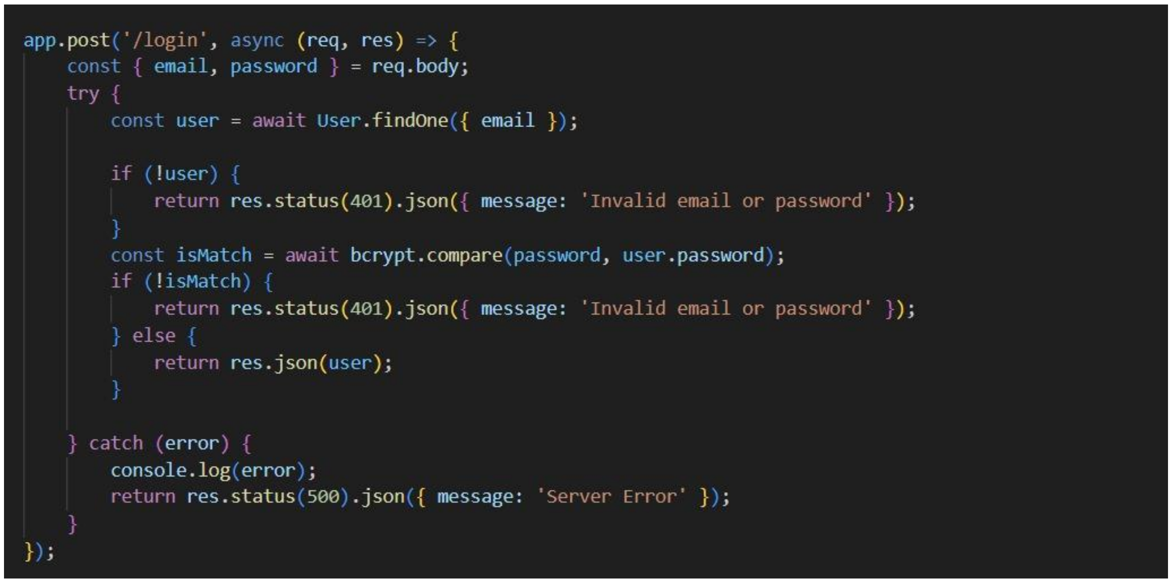
User Authentication

- **POST /api/user/register** - Registers a new user.



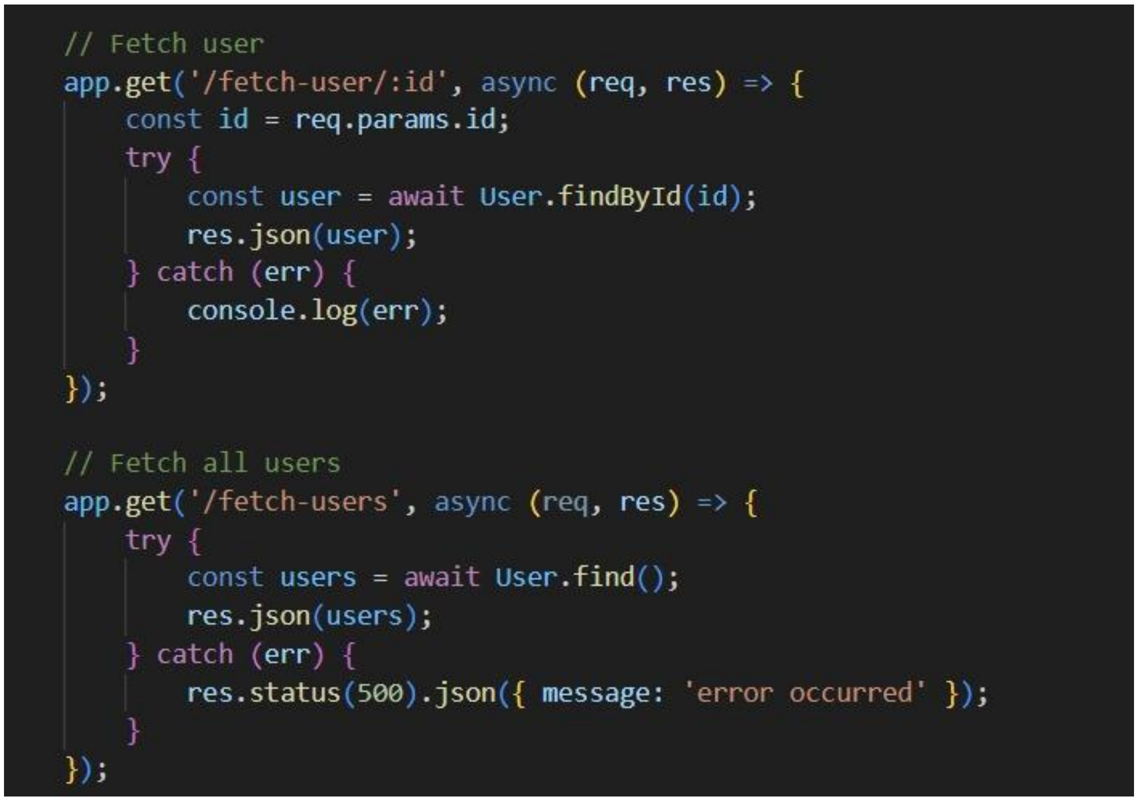
```
.then(() => {
    console.log('Connected to MongoDB');

    // All the client-server activities
    app.post('/register', async (req, res) => {
        const { username, email, usertype, password } = req.body;
        let approval = 'approved';
        try {
            const existingUser = await User.findOne({ email });
            if (existingUser) {
                return res.status(400).json({ message: 'User already exists' });
            }

            if (usertype === 'flight-operator') {
                approval = 'not-approved';
            }

            const hashedPassword = await bcrypt.hash(password, 10);
            const newUser = new User({
                username, email, usertype, password: hashedPassword, approval
            });
            const userCreated = await newUser.save();
            return res.status(201).json(userCreated);
        } catch (error) {
            console.log(error);
            return res.status(500).json({ message: 'Server Error' });
        }
    });
```

- **POST /api/user/login -** Authenticates a user and returns a token



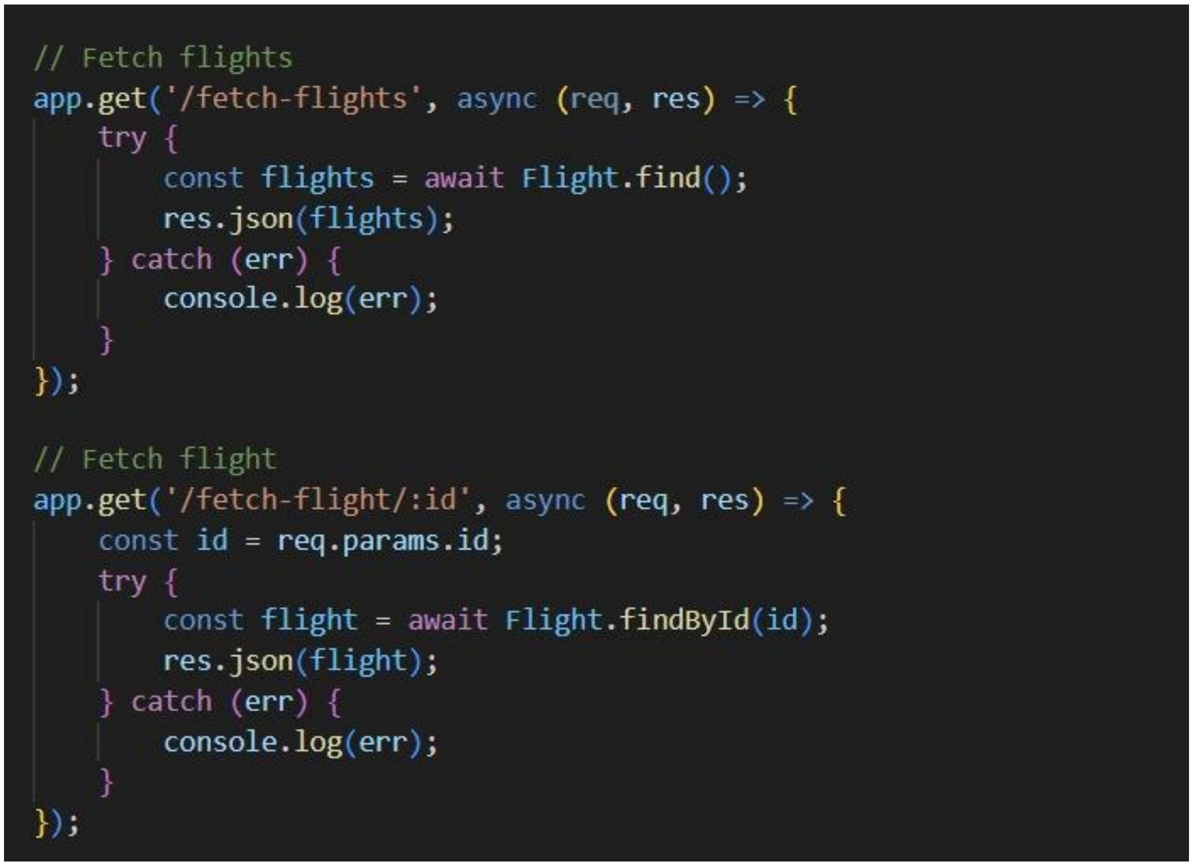
```
app.post('/login', async (req, res) => {
    const { email, password } = req.body;
    try {
        const user = await User.findOne({ email });

        if (!user) {
            return res.status(401).json({ message: 'Invalid email or password' });
        }
        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) {
            return res.status(401).json({ message: 'Invalid email or password' });
        } else {
            return res.json(user);
        }

    } catch (error) {
        console.log(error);
        return res.status(500).json({ message: 'Server Error' });
    }
});
```

- **GET /api/user/{id}** - Retrieves user information by ID.



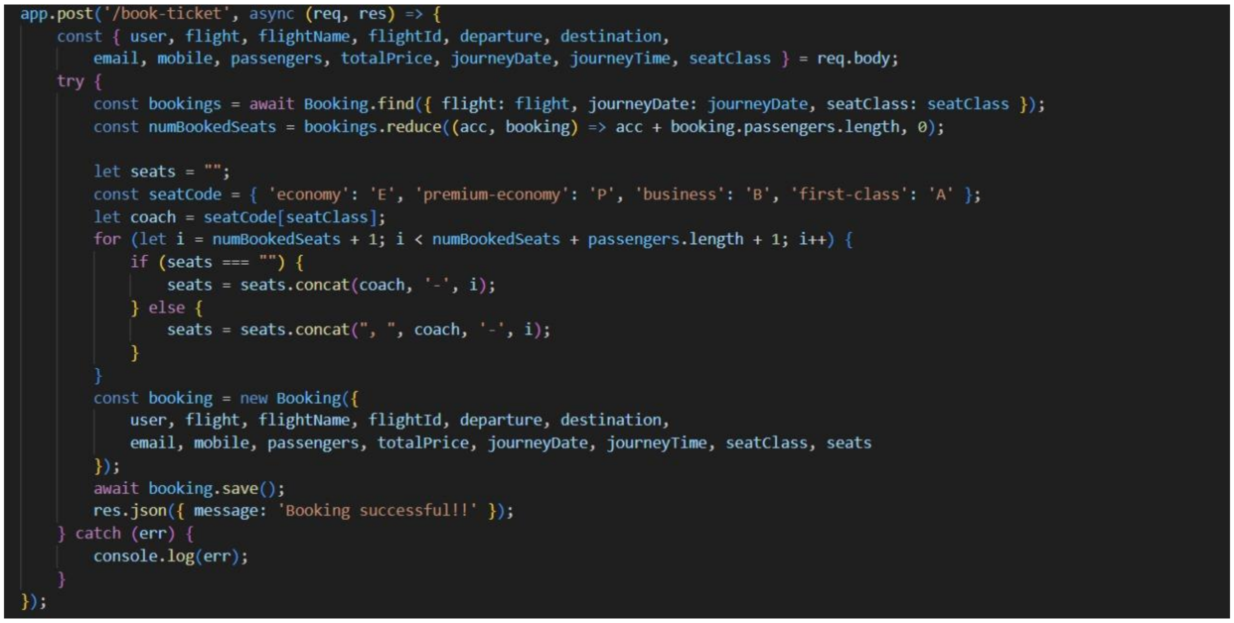
```
// Fetch user
app.get('/fetch-user/:id', async (req, res) => {
    const id = req.params.id;
    try {
        const user = await User.findById(id);
        res.json(user);
    } catch (err) {
        console.log(err);
    }
});

// Fetch all users
app.get('/fetch-users', async (req, res) => {
    try {
        const users = await User.find();
        res.json(users);
    } catch (err) {
        res.status(500).json({ message: 'error occurred' });
    }
});
```

- **GET /api/flights** - Retrieves all available flights.



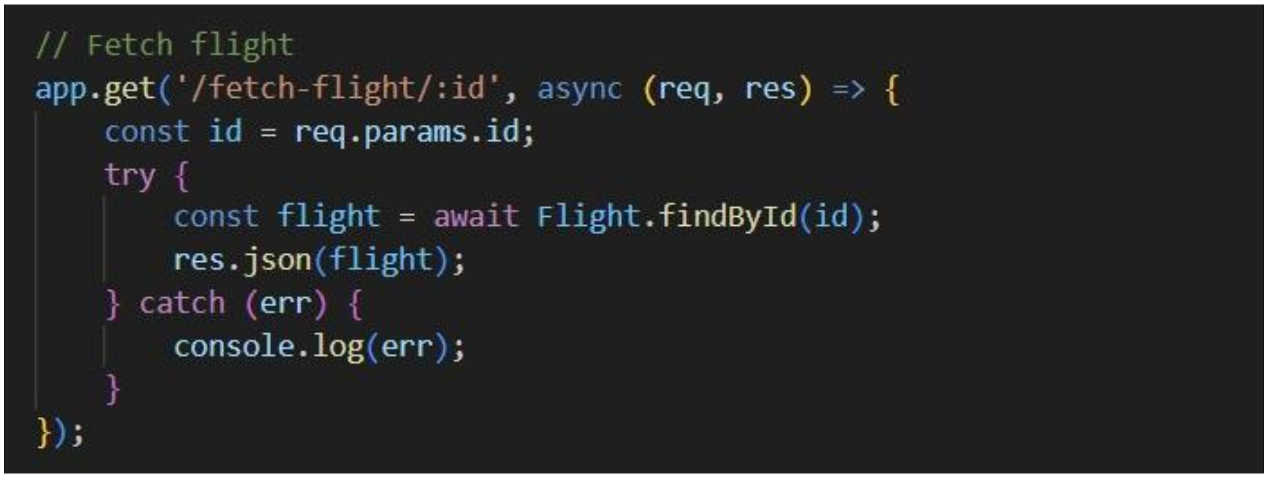
```javascript
// Fetch flights
app.get('/fetch-flights', async (req, res) => {
    try {
        const flights = await Flight.find();
        res.json(flights);
    } catch (err) {
        console.log(err);
    }
});

// Fetch flight
app.get('/fetch-flight/:id', async (req, res) => {
    const id = req.params.id;
    try {
        const flight = await Flight.findById(id);
        res.json(flight);
    } catch (err) {
        console.log(err);
    }
});
```

- **POST /api/flights/book -** Books a flight.



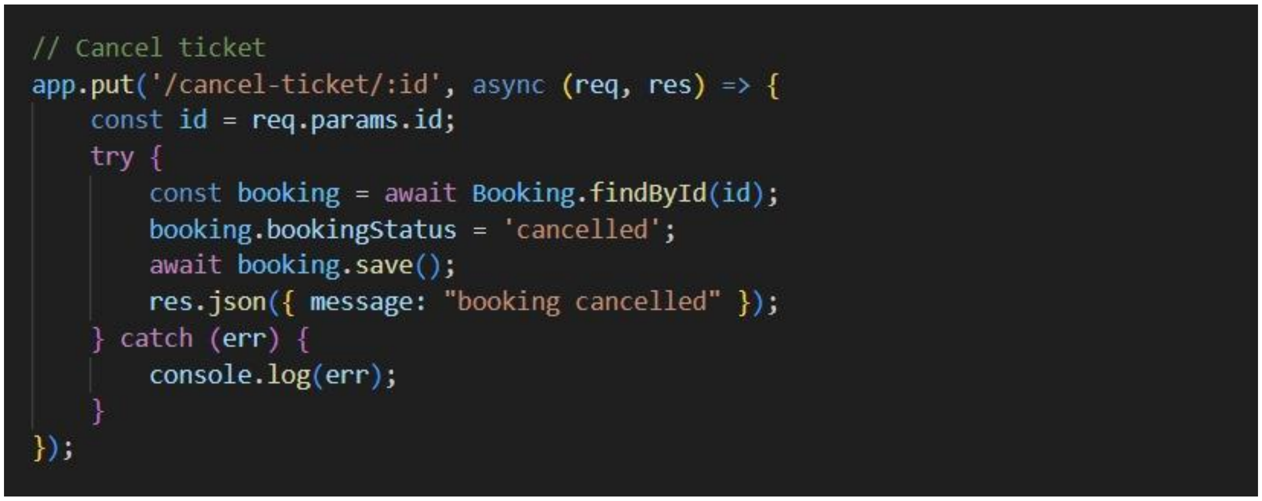
```javascript
app.post('/book-ticket', async (req, res) => {
    const { user, flight, flightName, flightId, departure, destination,
        email, mobile, passengers, totalPrice, journeyDate, journeyTime, seatClass } = req.body;
    try {
        const bookings = await Booking.find({ flight: flight, journeyDate: journeyDate, seatClass: seatClass });
        const numBookedSeats = bookings.reduce((acc, booking) => acc + booking.passengers.length, 0);

        let seats = "";
        const seatCode = { 'economy': 'E', 'premium-economy': 'P', 'business': 'B', 'first-class': 'A' };
        let coach = seatCode[seatClass];
        for (let i = numBookedSeats + 1; i < numBookedSeats + passengers.length + 1; i++) {
            if (seats === "") {
                seats = seats.concat(coach, '-', i);
            } else {
                seats = seats.concat(", ", coach, '-', i);
            }
        }
        const booking = new Booking({
            user, flight, flightName, flightId, departure, destination,
            email, mobile, passengers, totalPrice, journeyDate, journeyTime, seatClass, seats
        });
        await booking.save();
        res.json({ message: 'Booking successful!!' });
    } catch (err) {
        console.log(err);
    }
});
```

- **GET /api/flights/{id}** - Retrieves flight details by ID



```
// Fetch flight
app.get('/fetch-flight/:id', async (req, res) => {
    const id = req.params.id;
    try {
        const flight = await Flight.findById(id);
        res.json(flight);
    } catch (err) {
        console.log(err);
    }
});
```

- **PUT /api/flights/{id}/cancel -** Cancels a booked flight by ID.

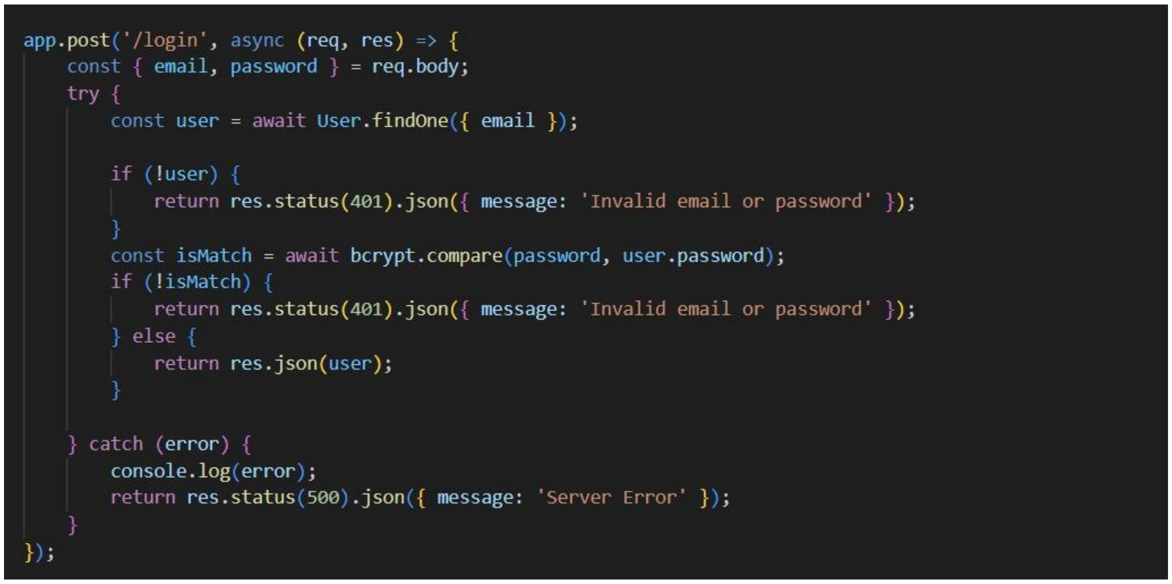


```
// Cancel ticket
app.put('/cancel-ticket/:id', async (req, res) => {
    const id = req.params.id;
    try {
        const booking = await Booking.findById(id);
        booking.bookingStatus = 'cancelled';
        await booking.save();
        res.json({ message: "booking cancelled" });
    } catch (err) {
        console.log(err);
    }
});
```

Integration with Frontend

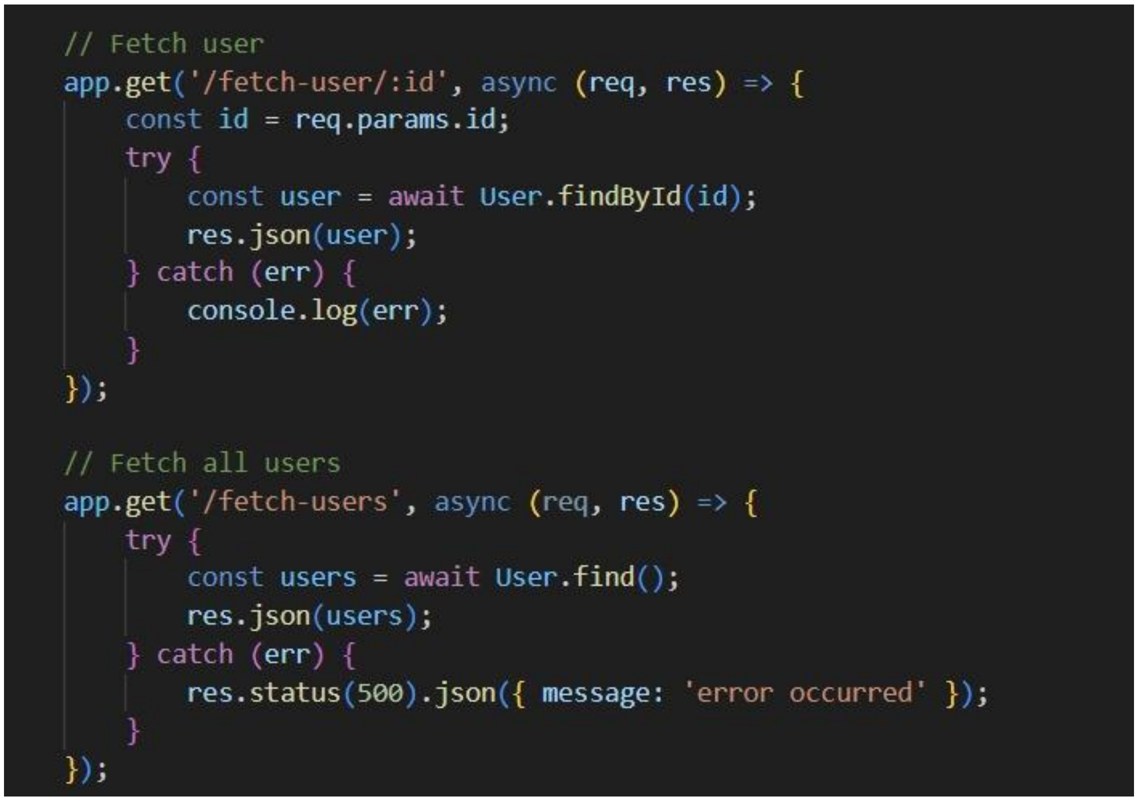
The backend communicates with the frontend via RESTful APIs. Key points of integration include:

- **User Authentication:** Tokens are passed between frontend and backend to handle authentication.



```
app.post('/login', async (req, res) => {
    const { email, password } = req.body;
    try {
        const user = await User.findOne({ email });

        if (!user) {
            return res.status(401).json({ message: 'Invalid email or password' });
        }
        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) {
            return res.status(401).json({ message: 'Invalid email or password' });
        } else {
            return res.json(user);
        }

    } catch (error) {
        console.log(error);
        return res.status(500).json({ message: 'Server Error' });
    }
});
```

- **Data Fetching:** Frontend components make API calls to fetch necessary data for display and interaction.

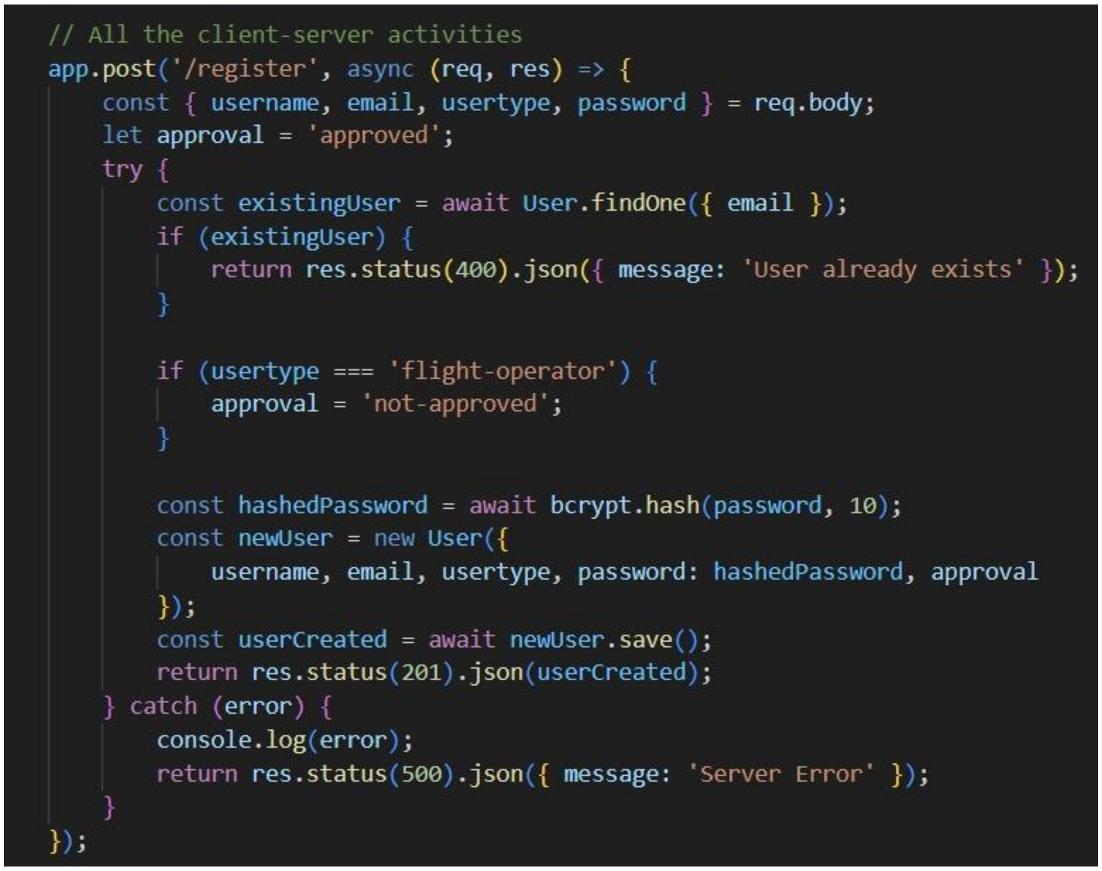


```
// Fetch user
app.get('/fetch-user/:id', async (req, res) => {
    const id = req.params.id;
    try {
        const user = await User.findById(id);
        res.json(user);
    } catch (err) {
        console.log(err);
    }
});

// Fetch all users
app.get('/fetch-users', async (req, res) => {
    try {
        const users = await User.find();
        res.json(users);
    } catch (err) {
        res.status(500).json({ message: 'error occurred' });
    }
});
```

Error Handling and Validation

Describe the error handling strategy and validation mechanisms:

- **Error Handling:** Centralized error handling using middleware.

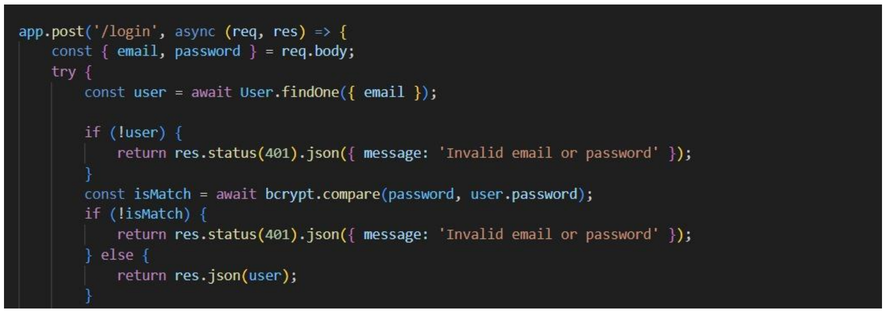


```javascript
// All the client-server activities
app.post('/register', async (req, res) => {
    const { username, email, usertype, password } = req.body;
    let approval = 'approved';
    try {
        const existingUser = await User.findOne({ email });
        if (existingUser) {
            return res.status(400).json({ message: 'User already exists' });
        }

        if (usertype === 'flight-operator') {
            approval = 'not-approved';
        }

        const hashedPassword = await bcrypt.hash(password, 10);
        const newUser = new User({
            username, email, usertype, password: hashedPassword, approval
        });
        const userCreated = await newUser.save();
        return res.status(201).json(userCreated);
    } catch (error) {
        console.log(error);
        return res.status(500).json({ message: 'Server Error' });
    }
});
```

Security Considerations

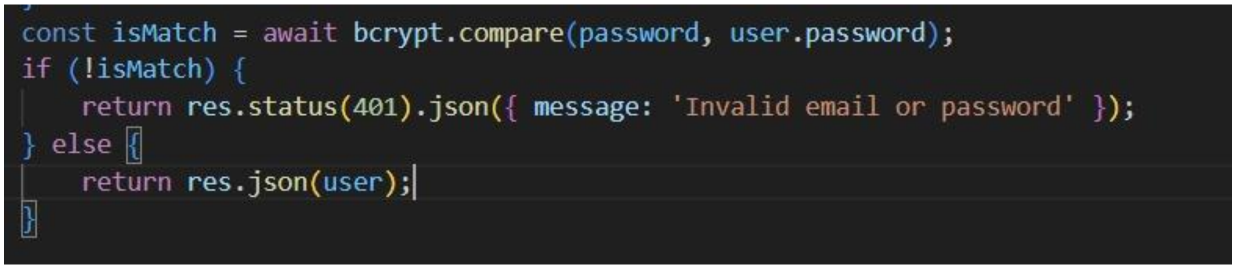
Outline the security measures implemented:

Authentication: Secure token-based authentication.



```javascript
app.post('/login', async (req, res) => {
    const { email, password } = req.body;
    try {
        const user = await User.findOne({ email });

        if (!user) {
            return res.status(401).json({ message: 'Invalid email or password' });
        }
        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) {
            return res.status(401).json({ message: 'Invalid email or password' });
        } else {
            return res.json(user);
        }
```

- **Data Encryption:** Encrypt sensitive data at rest and in transit.



```
const isMatch = await bcrypt.compare(password, user.password);
if (!isMatch) {
    return res.status(401).json({ message: 'Invalid email or password' });
} else {
    return res.json(user);
}
```