

Overview

This documentation outlines the backend structure and functionality of the Bus Booking System. The backend is developed using Node.js and MongoDB with Mongoose for data modeling. The application allows users to manage bus bookings, check occupancy status, and ensures data integrity.

Table of Contents

1. Getting Started
2. Data Models
 - o Bus Model
 - o Booking Model
3. Controllers
 - o Bus Controller
 - o Booking Controller
4. API Endpoints
 - o Get Bus Occupancy Color
5. Conclusion

Getting Started

To set up the backend for the Bus Booking System, follow these steps:

1. Clone the repository.
2. Navigate to the project directory.
3. Install the required dependencies:

```
npm install
```

4. Start the server:

```
npm start
```

Data Models

Bus Model

The Bus model defines the schema for buses in the system. It includes details such as bus name, total seats, occupancy, operation days, and route information.

```
// Schema for each seat in the bus
const seatSchema = new mongoose.Schema({
  seatNumber: {
    type: String,
    required: true,
  },
  isBooked: { type: Boolean, default: false }, // Track if the seat is booked
  bookedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User', default: null }, // User who booked the seat
  lockUntil: { type: Date, default: null }, // Optional: Lock seat temporarily for booking
});
```

```
// Main Bus Schema
const busSchema = new mongoose.Schema({
  busName: { type: String, required: true },
  totalSeats: {
    type: Number,
    required: true,
    validate: {
      validator: function(value) {
        return value > 0; // Ensure total seats is greater than 0
      },
      message: 'Total seats must be a positive number.'
    }
  },
  seats: [seatSchema], // Array of seat objects
  currentOccupancy: {
    type: Number,
    default: 0,
    min: 0 // Ensure occupancy is non-negative
  },
  operationDays: [
    type: String,
    enum: ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], // Validate days
  ],
  route: {
    source: { type: String, required: true },
    destination: { type: String, required: true },
    distance: {
      type: Number,
      required: true,
      min: 0 // Distance must be non-negative
    },
    eta: { type: String, required: true, match: /^[\d{2}:\d{2}]$/ }, // Validate ETA (hh:mm format)
  },
}, { timestamps: true }); // Add createdAt and updatedAt timestamps

// Ensure that buses with the same name and route are unique
busSchema.index({ busName: 1, 'route.source': 1, 'route.destination': 1 }, { unique: true });
```

Booking Model

The Booking model manages the details of each booking, associating users with specific seats on a bus.

```
const bookingSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  busId: { type: mongoose.Schema.Types.ObjectId, ref: 'Bus', required: true },
  seatNumber: { type: String, required: true }, // Store the seat number booked by the user
  bookedAt: { type: Date, default: Date.now }, // Timestamp for the booking
});

module.exports = mongoose.model('Booking', bookingSchema);
```

User Model

The User schema is defined using Mongoose and provides a structured format for storing user data in the MongoDB database. The schema includes necessary fields for user identification, role management, and security features.

```
const crypto = require('crypto');
const { v1: uuidv1 } = require('uuid'); // Ensure you have uuid installed for salt generation

// Define the schema for a User model
const userSchema = new mongoose.Schema({
  username: { type: String, required: true }, // Username is required
  email: { type: String, required: true, unique: true }, // Email is required and must be unique
  role: {
    type: String,
    enum: ['customer', 'coach', 'operations'], // Role must be one of these options
    required: true
  },
  createdAt: { type: Date, default: Date.now }, // Default to the current date if not provided
  encry_password: {
    type: String,
    required: false
  },

  salt: String,
  role: [
    type: Number,
    default: 0
  ]
});
```

```
// Custom methods
userSchema.methods = {
  securePassword: function(plainpassword) {
    if (!plainpassword) return "";
    try {
      return crypto.createHmac('sha256', this.salt).update(plainpassword).digest('hex');
    } catch (err) {
      console.error("Error in securePassword:", err);
      return null; // Handle error appropriately
    }
  },
  authenticate: function(plainpassword) {
    return this.securePassword(plainpassword) === this.encry_password;
  }
};

// Creating a virtual field to handle password
userSchema.virtual("password")
  .set(function(password) {
    this._password = password;
    this.salt = uuidv1(); // Generate a unique salt for the user
    this.encry_password = this.securePassword(password); // Encrypt the password
  })
  .get(function() {
    return this._password; // Return the raw password
  });
}
```

Controllers

Bus Controller

This document provides an overview of the controller methods for managing buses and bookings in the Bus Booking System. Each method includes functionality for adding, updating, retrieving, and managing bus occupancy and bookings.

Table of Contents

1. Add a New Bus
2. Update an Existing Bus
3. Delete a Bus
4. Get Bus by ID
5. Get Bus Details
6. Get Available Buses
7. Get Seat Availability
8. Book a Seat
9. Cancel a Booked Seat
10. Lock a Seat Temporarily
11. Get All Buses
12. Get Bus Occupancy Color
13. Conclusion

Add a New Bus

Method: addBus

This method allows administrators to add a new bus to the system.

```
exports.addBus = async (req, res) => {
  try {
    const newBus = new Bus(req.body); // Assuming req.body contains bus details
    await newBus.save();
    res.status(201).json({ message: 'Bus added successfully!', bus: newBus });
  } catch (error) {
    res.status(500).json({ error: 'Failed to add bus.', details: error.message });
  }
};
```

Request: POST /buses

Response: Returns a success message and the newly added bus details.

Update an Existing Bus

Method: updateBus

This method updates the details of an existing bus based on its ID.

```
exports.updateBus = async (req, res) => {
  try {
    const { busId } = req.params; // Get busId from URL parameters
    const updatedBus = await Bus.findByIdAndUpdate(busId, req.body, { new: true });

    if (!updatedBus) {
      return res.status(404).json({ error: 'Bus not found.' });
    }

    res.status(200).json({ message: 'Bus updated successfully!', bus: updatedBus });
  } catch (error) {
    res.status(500).json({ error: 'Failed to update bus.', details: error.message });
  }
};
```

Request: PUT /buses/:busId

Response: Returns a success message and the updated bus details.

Delete a Bus

Method: deleteBus

This method deletes a bus from the system using its ID.

```
exports.deleteBus = async (req, res) => {
  try {
    const { busId } = req.params; // Get busId from URL parameters
    const deletedBus = await Bus.findByIdAndDelete(busId);

    if (!deletedBus) {
      return res.status(404).json({ error: 'Bus not found.' });
    }

    res.status(200).json({ message: 'Bus deleted successfully!' });
  } catch (error) {
    res.status(500).json({ error: 'Failed to delete bus.', details: error.message });
  }
};
```

Request: DELETE /buses/:busId

Response: Returns a success message upon successful deletion.

Get Bus by ID

Method: getBusById

This middleware retrieves a bus by its ID and attaches it to the request object for further use in other routes.

```
exports.getBusById = async (req, res, next) => {
  try {
    const { busId } = req.params; // Get busId from URL parameters

    const bus = await Bus.findById(busId);
    if (!bus) {
      return res.status(404).json({ error: 'Bus not found.' });
    }

    req.bus = bus; // Attach bus to req.bus for further use
    next();
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch bus.', details: error.message });
  }
};
```

Request: GET /buses/:busId

Response: No direct response; it attaches the bus object to the request for further processing.

Get Bus Details

Method: getBusByIdDetails

This method retrieves detailed information about a bus, including its seat availability.

```
exports.getBusByIdDetails = async (req, res) => {
  try {
    const { busId } = req.params; // Get busId from URL parameters

    const bus = await Bus.findById(busId).populate('seats'); // Populate seats if needed
    if (!bus) {
      return res.status(404).json({ error: 'Bus not found.' });
    }

    res.status(200).json(bus);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch bus details.', details: error.message });
  }
};
```

Request: GET /buses/:busId/details

Response: Returns the bus details, including seat availability.

Get Available Buses

Method: getAvailableBuses

This method retrieves all available buses based on the source and destination with pagination.

```
exports.getAvailableBuses = async (req, res) => {
  try {
    const { source, destination, page = 1, limit = 10 } = req.body;

    const buses = await Bus.find({
      'route.source': source,
      'route.destination': destination,
    })
      .skip((page - 1) * limit)
      .limit(Number(limit));

    if (!buses.length) {
      return res.status(404).json({ message: 'No buses available for this route.' });
    }

    res.status(200).json(buses);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch available buses.', details: error.message });
  }
};
```

- **Request:** POST /buses/available
- **Response:** Returns an array of available buses based on the provided criteria.

Get Seat Availability

Method: `getSeatAvailability`

This method returns the availability status of seats for a specific bus.

```
exports.getSeatAvailability = async (req, res) => {
  try {
    const { busId } = req.params; // Get busId from URL parameters

    const bus = await Bus.findById(busId);
    if (!bus) {
      return res.status(404).json({ error: 'Bus not found.' });
    }

    const seatAvailability = bus.seats.map(seat => ({
      seatNumber: seat.seatNumber,
      isBooked: seat.isBooked,
      bookedBy: seat.bookedBy,
    }));
  }

  res.status(200).json(seatAvailability);
} catch (error) {
  res.status(500).json({ error: 'Failed to fetch seat availability.', details: error.message });
};
```

Request: GET /buses/:busId/seats

Response: Returns an array of seat availability for the specified bus.

Book a Seat

Method: bookSeat

This method allows a user to book a seat on a bus.

```
exports.bookSeat = async (req, res) => {
  try {
    const { seatNumber } = req.body; // Get seatNumber from the request body
    const { userId } = req.params; // Get userId from the request parameters
    const bus = req.bus; // Use the bus information attached to req
    const seat = bus.seats.find(seat => seat.seatNumber === seatNumber);

    if (!seat) return res.status(404).json({ error: 'Seat not found.' });
    if (seat.isBooked) return res.status(400).json({ error: 'Seat already booked.' });

    // Book the seat
    seat.isBooked = true;
    seat.bookedBy = userId;
    bus.currentOccupancy += 1;

    const newBooking = new Booking({
      userId,
      busId: bus._id, // Reference the booked bus
      seatNumber,
    });

    const booking = await newBooking.save();

    await bus.save();
    res.status(200).json({ message: `Seat ${seatNumber} booked successfully!`, booking });
  } catch (error) {
    res.status(500).json({ error: 'Failed to book seat.', details: error.message });
  }
};
```

Request: POST /buses/:busId/book-seat/:userId

Response: Returns a success message and booking details upon successful seat booking.

Cancel a Booked Seat

Method: cancelSeatBooking

This method allows a user to cancel a previously booked seat.

```
exports.cancelSeatBooking = async (req, res) => {
  try {
    const { seatNumber } = req.body; // Get seatNumber from the request body
    const bus = req.bus; // Use the bus information attached to req

    const seat = bus.seats.find(seat => seat.seatNumber === seatNumber);

    if (!seat || !seat.isBooked) {
      return res.status(400).json({ error: 'Seat is not booked.' });
    }

    // Cancel the booking
    seat.isBooked = false;
    seat.bookedBy = null;
    bus.currentOccupancy -= 1;

    await bus.save();
    res.status(200).json({ message: `Seat ${seatNumber} booking canceled successfully!` })
  } catch (error) {
    res.status(500).json({ error: 'Failed to cancel booking.', details: error.message });
  }
};
```

Request: POST /buses/:busId/cancel-seat

Response: Returns a success message upon successful cancellation.

Lock a Seat Temporarily

Method: lockSeat

This method allows a user to temporarily lock a seat for booking.

```
exports.lockSeat = async (req, res) => {
  try {
    const { seatNumber } = req.body; // Get seatNumber from the request body
    const bus = req.bus; // Use the bus information attached to req
    const lockTime = 5 * 60 * 1000; // 5 minutes

    const seat = bus.seats.find(seat => seat.seatNumber === seatNumber);

    if (!seat) return res.status(404).json({ error: 'Seat not found.' });
    if (seat.isBooked || seat.lockUntil > new Date()) {
      return res.status(400).json({ error: 'Seat is already locked or booked.' });
    }

    // Lock the seat temporarily
    seat.lockUntil = new Date(Date.now() + lockTime);
    await bus.save();

    res.status(200).json({ message: `Seat ${seatNumber} locked for 5 minutes.` });
  } catch (error) {
    res.status(500).json({ error: 'Failed to lock seat.', details: error.message });
  }
};
```

Request: POST /buses/:busId/lock-seat

Response: Returns a message indicating the seat is locked for a specified duration.

Get All Buses

Method: getAllBuses

This method retrieves all bus records from the database, accessible only to admins.

```
exports.getAllBuses = async (req, res) => {
  try {
    const buses = await Bus.find();
    res.status(200).json(buses);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch all buses.', details: error.message });
  }
};
```

Request: GET /buses

Response: Returns a list of all buses in the system.

Get Bus Occupancy Color

Method: getBusOccupancyColor

This method calculates the occupancy color of a bus based on its current occupancy percentage.

```
exports.getBusOccupancyColor = async (req, res) => {
  try {
    const { busId } = req.params; // Get the bus ID from the request parameters

    const bus = await Bus.findById(busId);
    if (!bus) {
      return res.status(404).json({ error: 'Bus not found.' });
    }

    // Calculate occupancy percentage
    const totalSeats = bus.totalSeats;
    const currentOccupancy = bus.currentOccupancy;

    const occupancyPercentage = (currentOccupancy / totalSeats) * 100;

    // Determine color based on occupancy percentage
    let color;
    if (occupancyPercentage <= 60) {
      color = 'Green'; // 60% or less
    } else if (occupancyPercentage > 60 && occupancyPercentage < 90) {
      color = 'Yellow'; // Between 60% and 90%
    } else {
      color = 'Red'; // 90% or more
    }

    res.status(200).json({
      busId,
      occupancyPercentage: occupancyPercentage.toFixed(2), // Round to two decimal places
      color
    });
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch bus occupancy.', details: error.message });
  }
};
```

Request: GET /buses/:busId/occupancy-color

Response: Returns the occupancy percentage and corresponding color indicating seat availability.

Booking Controller:

This document outlines the controller methods for managing bookings in the Bus Booking System. Each method includes functionality for creating, retrieving, updating, and deleting bookings, as well as additional methods for querying bookings based on various criteria.

Table of Contents

1. Get Booking by ID
2. Add a New Booking
3. Get All Bookings for a User
4. Update a Booking
5. Delete a Booking
6. Get Bookings for a Specific Date

Get Booking by ID

Method: `getBookingById`

This middleware retrieves a specific booking by its ID and populates associated bus details.

```
exports.getBookingById = async (req, res, next, id) => {
  try {
    const booking = await Booking.findById(id).populate('busId', 'busName route');
    if (!booking) return res.status(404).json({ error: 'Booking not found.' });
    req.booking = booking;
    next();
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch booking.', details: error.message });
  }
};
```

- **Request:** Typically used in routes that require booking details.
- **Response:** No direct response; attaches the booking object to the request for further processing.

Add a New Booking

Method: addBooking

This method creates a new booking for a specified bus and seat.

```
exports.addBooking = async (req, res) => {
  try {
    const { busId, seatNumber } = req.body;
    const userId = req.params.userId;

    // Check if the seat is available
    const bus = await Bus.findOneAndUpdate(
      { _id: busId, 'seats.seatNumber': seatNumber, 'seats.isBooked': false },
      { $set: { 'seats.$isBooked': true }, $inc: { currentOccupancy: 1 } },
      { new: true }
    );

    if (!bus) {
      return res.status(400).json({ error: 'Seat is already booked or bus not found.' });
    }

    // Create a new booking
    const booking = new Booking({ userId, busId, seatNumber });
    await booking.save();

    res.status(201).json({ message: 'Booking added successfully!', booking });
  } catch (error) {
    res.status(500).json({ error: 'Failed to add booking.', details: error.message });
  }
};
```

- **Request:** POST /bookings/:userId
- **Response:** Returns a success message and the newly created booking details.

Get All Bookings for a User

Method: getAllBookings

This method retrieves all bookings associated with a specific user.

```
exports.getAllBookings = async (req, res) => {
  try {
    const userId = req.params.userId;
    const bookings = await Booking.find({ userId }).populate('busId', 'busName route');

    if (!bookings.length) {
      return res.status(404).json({ message: 'No bookings found for this user.' });
    }

    res.status(200).json(bookings);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch user bookings.', details: error.message });
  }
};
```

Request: GET /bookings/user/:userId

Response: Returns an array of bookings for the specified user.

Update a Booking

Method: updateBooking

This method updates the details of an existing booking.

```
exports.updateBooking = async (req, res) => {
  try {
    const { bookingId } = req.params;
    const updates = req.body;

    const updatedBooking = await Booking.findByIdAndUpdate(bookingId, updates, { new: true })
      'busId',
      'busName route'
    );

    if (!updatedBooking) {
      return res.status(404).json({ error: 'Booking not found.' });
    }
  }
};
```

Request: PUT /bookings/:bookingId

Response: Returns a success message and the updated booking details.

```
res.status(200).json({ message: 'Booking updated successfully!', booking: updatedBooking })
} catch (error) {
  res.status(500).json({ error: 'Failed to update booking.', details: error.message });
}
};
```

Delete a Booking

Method: `deleteBooking`

This method deletes a specified booking and updates the associated seat status.

```
exports.deleteBooking = async (req, res) => {
  try {
    const { bookingId } = req.params;

    // Find the booking to be deleted
    const booking = await Booking.findById(bookingId);
    if (!booking) return res.status(404).json({ error: 'Booking not found.' });

    // Update the seat status on the bus
    const bus = await Bus.findOne({ _id: booking.busId, 'seats.seatNumber': booking.seatNumber });
    if (!bus) return res.status(404).json({ error: 'Bus not found.' });

    const seat = bus.seats.find(seat => seat.seatNumber === booking.seatNumber);
    if (seat) {
      seat.isBooked = false; // Cancel the seat booking
      seat.bookedBy = null; // Clear the bookedBy field
      bus.currentOccupancy -= 1; // Decrease current occupancy
      await bus.save(); // Save the updated bus
    }

    // Delete the booking record
    await Booking.findByIdAndUpdateAndDelete(bookingId);

    res.status(200).json({ message: 'Booking deleted successfully!' });
  } catch (error) {
    res.status(500).json({ error: 'Failed to delete booking.', details: error.message });
  }
}
```

Request: `DELETE /bookings/:bookingId`

Response: Returns a success message upon successful deletion.

Get Bookings for a Specific Date

Method: `getBookingsForDate`

This method retrieves all bookings for a specific date, providing a start and end range for filtering.

```
exports.getBookingsForDate = async (req, res) => {
  try {
    const { date } = req.query;

    // Convert the date from the query to a start and end range
    const startDate = new Date(date);
    const endDate = new Date(startDate);
    endDate.setDate(startDate.getDate() + 1); // Set to the next day

    const bookings = await Booking.find({
      bookedAt: { $gte: startDate, $lt: endDate } // Filter for bookings within the date range
    }).populate('busId', 'busName route');

    if (!bookings.length) {
      return res.status(404).json({ message: 'No bookings found for this date.' });
    }

    res.status(200).json(bookings);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch bookings.', details: error.message });
  }
};
```

User Authentication Controller

This document outlines the controller methods responsible for user authentication in the Bus Booking System. It includes methods for user signup and signin, middleware for route protection, and validation logic to ensure data integrity.

Table of Contents

1. User Signup
2. User Signin
3. Middleware for Authentication
 - o Check if User is Signed In
 - o Check if User is Authenticated
 - o Check if User is Admin

User Signup

Method: `signup`

This method handles the registration of a new user. It validates the input data, creates a new user record in the database, generates a JWT token, and optionally stores it in a cookie.

```
exports.signup = async (req, res) => {
  const errors = validationResult(req);

  // Return error if validation fails
  if (!errors.isEmpty()) {
    return res.status(422).json({
      error: errors.array()[0].msg,
    });
  }

  try {
    // Create a new user
    const user = new User(req.body);
    const savedUser = await user.save(); // Save user to DB

    // Generate a JWT token
    const token = jwt.sign({ _id: savedUser._id }, process.env.SECRET);

    // Store token in cookie
    res.cookie("token", token, { expire: new Date() + 9999 });

    // Return token and basic user info
    const { _id, name, email, role } = savedUser;
    return res.json({
      token,
      user: { _id, name, email, role },
    });
  } catch (error) {
    console.error("Error saving user:", error);
    return res.status(400).json({
      error: "Not able to save user in DATABASE",
    });
  }
};
```

- **Request:** POST /signup
- **Response:** Returns a JSON object containing the JWT token and the user's basic information upon successful signup.

User Signin

Method: signin

This method authenticates an existing user by validating their credentials and generating a JWT token for session management.

```
exports.signin = (req, res) => {
    const errors = validationResult(req);
    const { email, password } = req.body;

    // Return error if validation fails
    if (!errors.isEmpty()) {
        return res.status(422).json({
            error: errors.array()[0].msg
        });
    }

    // Find user by email
    User.findOne({ email }).then((user, err) => {
        if (err || !user) {
            return res.status(400).json({
                error: "User email does not exist"
            });
        }

        // Authenticate the user by password
        if (!user.authenticate(password)) {
            return res.status(401).json({
                error: "Email and password do not match"
            });
        }

        // Generate JWT token
        const token = jwt.sign({ _id: user._id }, process.env.SECRET);

        // Store token in cookie
        res.cookie("token", token, { expire: new Date() + 9999 });

        // Send response with user details and token
        const { _id, name, email, role } = user;
        return res.json({ token, user: { _id, name, email, role } });
    });
};
```

Request: POST /signin

Response: Returns a JSON object containing the JWT token and the user's details upon successful signin.

Middleware for Authentication

Middleware functions are used to protect routes and enforce authentication rules.

Check if User is Signed In

Method: `isSignedIn`

This middleware checks if the user is signed in by verifying the JWT token in the request.

```
exports.isSignedIn = expressJwt({
  secret: process.env.SECRET,
  algorithms: ["HS256"],
  userProperty: "auth"
});
```

- **Usage:** Applied to routes that require user authentication.
- **Response:** Automatically denies access to routes if the user is not signed in.

Check if User is Authenticated

Method: `isAuthenticated`

This middleware verifies whether the authenticated user matches the requested profile.

```
exports.isAuthenticated = (req, res, next) => {
  const check = req.profile && req.auth && req.profile._id == req.auth._id;
  if (!check) {
    return res.status(403).json({
      error: "Access Denied"
    });
  }
  next();
};
```

Usage: Applied to routes where user-specific actions are allowed.

Response: Denies access if the user is not the owner of the profile.

Check if User is Admin

Method: isAdmin

This middleware checks if the authenticated user has admin privileges.

```
exports.isAdmin = (req, res, next) => {
  if (req.profile.role == 0) {
    return res.status(403).json({
      error: "You are not an admin"
    });
  }
  next();
};
```

- **Usage:** Applied to routes that require admin access.
- **Response:** Denies access if the user is not an admin.

Testing through Postman:

The screenshot shows a POST request to `http://localhost:5000/api/signup`. The request body is a JSON object with fields: `"email": "gaurabsahay2468@gmail.com"`, `"username": "Gaurav Sahay"`, and `"password": "iiita123"`. The response is a 200 OK status with a response time of 87 ms and a response size of 693 B. The response body is a JSON object containing a token and a user object. The token is a long string of characters, and the user object contains the user's ID, email, and role (0).

```
POST http://localhost:5000/api/signup

Params: Authorization: Headers (10) Body (raw) Scripts: Tests: Settings: Cookies: Beautify: none form-data x-www-form-urlencoded raw binary GraphQL JSON

1 {
2   "email" : "gaurabsahay2468@gmail.com",
3   "username" : "Gaurav Sahay",
4   "password" : "iiita123"
5 }
```

Body: Cookies (1) Headers (9) Test Results: 200 OK 87 ms 693 B

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NzE5mZjMzhjNDBlYzFmNTJjZDY0ZGEiLCJpYXQiOjE3Mjk3NTcxMjN9.NlWWJ0YCfmqq0R6HToVAr99D5htN-kc_dSBuKtjNyii",
3   "user": {
4     "_id": "6719ffc38c40ec1f52cd64da",
5     "email": "gaurabsahay2468@gmail.com",
6     "role": 0
7   }
8 }
```

Signup

The screenshot shows a POST request to `http://localhost:5000/api/signin`. The request body is a JSON object with `"email": "gaurabsahay2468@gmail.com"` and `"password": "iiita123"`. The response status is `200 OK` with a response time of 70 ms and a response size of 693 B. The response body is a JSON object containing a token and a user object.

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJfaWQiOiI2NzE5ZmZjMzhjNDBlYzFmNTJjZDV0ZGEiLCJpYXQiOjE3Mjk3NTcyNjZ9.  
ZkZAZ86RAwxCxrezEnh5-Hc8JlxQD147S7ZhnhAgh4s",  
3   "user": {  
4     "_id": "6719ffc38c40ec1f52cd64da",  
5     "email": "gaurabsahay2468@gmail.com",  
6     "role": 0  
7   }  
8 }
```

Signin

The screenshot shows a POST request to `http://localhost:5000/api/admin/6719ffc38c40ec1f52cd64da/add-bus`. The request body is a JSON object with bus details like name, total seats, seat numbers, operation days, route, source, destination, distance, and ETA.

```
1 {  
2   "busName": "Luxury Coach 1",  
3   "totalSeats": 50,  
4   "seats": [  
5     { "seatNumber": "1A" },  
6     { "seatNumber": "1B" },  
7     { "seatNumber": "1C" },  
8     { "seatNumber": "1D" },  
9     { "seatNumber": "2A" },  
10    { "seatNumber": "2B" }  
11  ],  
12  "operationDays": ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"],  
13  "route": {  
14    "source": "City A",  
15    "destination": "City B",  
16    "distance": 150,  
17    "eta": "02:30" // ETA in hh:mm format  
18  }  
19 }
```

Add new Bus (Request)

```
{
  "message": "Bus added successfully!",
  "bus": {
    "busName": "Luxury Coach 1",
    "totalSeats": 50,
    "seats": [
      {
        "seatNumber": "1A",
        "isBooked": false,
        "bookedBy": null,
        "lockUntil": null,
        "_id": "671a0189acd960bd46fcced2"
      },
      {
        "seatNumber": "1B",
        "isBooked": false,
        "bookedBy": null,
        "lockUntil": null,
        "_id": "671a0189acd960bd46fcced3"
      },
      {
        "seatNumber": "1C",
        "isBooked": false,
        "bookedBy": null,
        "lockUntil": null,
        "_id": "671a0189acd960bd46fcced4"
      }
    ]
  }
}
```

Add new Bus (Response)

Params	Authorization	Headers (11)	Body	Scripts	Tests	Settings	Cookies
Headers 9 hidden							
		Key	Value	Description	...	Bulk Edit	Presets ▾
	<input checked="" type="checkbox"/>	Content-Type	application/json				
	<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9eyJfaWQiOiI2NzE5ZmZjMzhjNDBIYzFmNTJjZDY0ZGEiLCJpyXQIjOjE3Mjk3NTcyNjZ9.ZkZAZ86RAwucxrezEnh5-	Description			
		Key					

Using JWT Token for authorization

HTTP BookMyBus / Update Bus

PUT <http://localhost:5000/api/admin/671a0189acd960bd46feced1/6719ffc38c40ec1f52cd64da/update-bus>

Params Authorization Headers (11) **Body** Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1  {
2    "busName": "Luxury Coach 2",
3    "totalSeats": 55,
4    "seats": [
5      { "seatNumber": "1A" }, // Include existing or new seats
6      { "seatNumber": "1B" },
7      { "seatNumber": "1C" },
8      { "seatNumber": "1D" },
9      { "seatNumber": "2A" },
10     { "seatNumber": "2B" }
11   ],
12   "operationDays": ["Monday", "Tuesday", "Thursday"],
13   "route": {
14     "source": "City A",
15     "destination": "City C",
16     "distance": 200,
17     "eta": "03:00"
18   }
19 }
```

Update Bus API (Request)

Body Cookies (1) Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2    "route": {
3      "source": "City A",
4      "destination": "City B",
5      "distance": 150,
6      "eta": "02:30"
7    },
8    "_id": "671a0189acd960bd46feced1",
9    "busName": "Luxury Coach 1",
10   "totalSeats": 50,
11   "seats": [
12     {
13       "seatNumber": "1A",
14       "isBooked": false,
15       "bookedBy": null,
16       "lockUntil": null,
17       "_id": "671a0189acd960bd46feced2"
18     },
19     {
20       "seatNumber": "1B",
21       "isBooked": false,
22       "bookedBy": null,
23       "lockUntil": null
24     }
25   ]
26 }
```

Update Bus API (Response)

The screenshot shows the Postman interface with the following details:

- HTTP Method:** DELETE
- URL:** http://localhost:5000/api/admin/671a03ebf31b8d9f131e901c/6719ffc38c40ec1f52cd64da/delete-bus
- Headers:** (11)
 - Content-Type: application/json
 - Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...
- Body:** (Pretty, Raw, Preview, Visualize, JSON)
 - Pretty JSON response:

```
1 {  
2   "route": {  
3     "source": "City A",  
4     "destination": "City B",  
5     "distance": 150,  
6     "eta": "02:30"  
7   },  
8   "_id": "671a03ebf31b8d9f131e901c",  
9   "busName": "Luxury Coach 2",  
10  "totalSeats": 50,  
11  "seats": [  
12    {  
13      "seatNumber": "00A",  
14      "isBooked": false,  
15      "bookedBy": null,  
16    }  
17  ]  
18}
```
- Response:** 200 OK (55 ms, 1.21 KB)

Delete Bus API

The screenshot shows the Postman interface with the following details:

- HTTP Method:** POST
- URL:** http://localhost:5000/api/671a0189acd960bd46fcced1/6719ffc38c40ec1f52cd64da/book-seat
- Params:** (11)
 - Query Params table:

Key	Value	Description	Bulk Edit
Key	Value	Description	
- Body:** (Pretty, Raw, Preview, Visualize, JSON)
 - Pretty JSON response:

```
1 {  
2   "message": "Seat 1A booked successfully!"  
3 }
```
- Response:** 200 OK (269 ms, 309 B)

Seat Booking API

```

▶ _id: ObjectId('671a0189acd960bd46fcced1')
  busName : "Luxury Coach 1"
  totalSeats : 50
  ▶ seats : Array (6)
    ▶ 0: Object
      seatNumber : "1A"
      isBooked : true
      lockUntil : null
      _id : ObjectId('671a0189acd960bd46fcced2')
    ▶ 1: Object
    ▶ 2: Object
    ▶ 3: Object
    ▶ 4: Object
    ▶ 5: Object
      currentOccupancy : 1
  ▶ operationDays : Array (5)
  ▶ route : Object

```

MongoDB document after booking a seat(1A)

HTTP BookMyBus / Lock Seat

POST <http://localhost:5000/api/671a0189acd960bd46fcced1/6719ffc38c40ec1f52cd64da/lock-seat> Send

Params Authorization Headers (11) Body Scripts Tests Settings Cookies Beautify

Body (1) Headers (8) Test Results 400 Bad Request 147 ms 321 B e.g. ...

Pretty Raw Preview Visualize JSON

```

1 {
2   "seatNumber": "1A"
3 }

```

```

1 {
2   "error": "Seat is already locked or booked."
3 }

```

Lock Seat API

The screenshot shows the Postman interface for a DELETE request to cancel a seat. The URL is `http://localhost:5000/api/671a0189acd960bd46fccccd1/6719ffc38c40ec1f52cd64da/cancel-seat`. The request body is a JSON object with a single key-value pair: `"seatNumber": "1A"`. The response status is 200 OK, with a response time of 166 ms and a response size of 319 B. The response body contains the message: `"message": "Seat 1A booking canceled successfully!"`.

Cancel a booked seat

The screenshot shows the Postman interface for a GET request to get available buses. The URL is `http://localhost:5000/api/available-buses`. The request body is a JSON object with two key-value pairs: `"source": "City A"` and `"destination": "City B"`. The response status is 200 OK, with a response time of 166 ms and a response size of 319 B.

Get Available Buses (Request)

```
[  
]  
{  
    "route": {  
        "source": "City A",  
        "destination": "City B",  
        "distance": 150,  
        "eta": "02:30"  
    },  
    "_id": "671a0189acd960bd46fcccc1",  
    "busName": "Luxury Coach 1",  
    "totalSeats": 50,  
    "seats": [  
        {  
            "bookedBy": null,  
            "seatNumber": "1A",  
            "isBooked": false,  
            "lockUntil": null,  
            "_id": "671a0189acd960bd46fcccc2"  
        },  
        {  
            "seatNumber": "1B",  
            "isBooked": false,  
            "bookedBy": null  
        }  
    ]  
}
```

Get Available buses (response)

HTTP BookMyBus / Get Seats

GET http://localhost:5000/api/671a0189acd960bd46fcccc1/seats

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	... Bulk Edit
Key	Value	Description	(i)

Body Cookies (1) Headers (8) Test Results 200 OK • 102 ms • 588 B | ⌂ ⌂ ⌂

Pretty Raw Preview Visualize JSON

```
1 [  
2 {  
3     "seatNumber": "1A",  
4     "isBooked": false,  
5     "bookedBy": null  
6 },  
7 {  
8     "seatNumber": "1B",  
9     "isBooked": false,  
10    "bookedBy": null  
11 },  
12 {  
13     "seatNumber": "1C",  
14     "isBooked": false,  
15     "bookedBy": null  
16 }
```

Postbot Ctrl Alt P

Get seats in a bus

HTTP BookMyBus / Get Bus Details

Save Share

GET http://localhost:5000/api/671a0189acd960bd46fcccc1/getDetails Send

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	...

Body Cookies (1) Headers (8) Test Results 200 OK • 164 ms • 1.2 KB e.g. ...

Pretty Raw Preview Visualize JSON

```
1 {
2   "route": {
3     "source": "City A",
4     "destination": "City B",
5     "distance": 150,
6     "eta": "02:30"
7   },
8   "_id": "671a0189acd960bd46fcccc1",
9   "busName": "Luxury Coach 1",
10  "totalSeats": 50,
11  "seats": [
12    {
13      "bookedBy": null,
14      "seatNumber": "1A",
15      "isBooked": false,
16      "lockUntil": null
17    }
18  ]
19}
```

Postbot Runner Start Proxy Cookies Vault Trash

Get details of a Bus

POST signIn POST Book Seat GET Get User Bookings GET Get All Bookings for a given date No environment

HTTP BookMyBus / Get All Bookings for a given date Save Share

GET http://localhost:5000/api/admin/6719ffc38c40ec1f52cd64da/booking/getBydate?date=2024-10-24 Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> date	2024-10-24		
Key	Value	Description	

Body Cookies (1) Headers (8) Test Results 200 OK • 153 ms • 1.1 KB | e.g. ...

Pretty Raw Preview Visualize JSON

```
1 [  
2 {  
3   "_id": "671a1c3f71ca418aeece5537b",  
4   "userId": "6719ffc38c40ec1f52cd64da",  
5   "busId": {  
6     "route": {  
7       "source": "City A",  
8       "destination": "City B",  
9       "distance": 150,  
10      "eta": "02:30"  
11    },  
12    "_id": "671a0189acd960bd46fcced1",  
13    "busName": "Luxury Coach 1"  
14  },  
15  "seatNumber": "2A",  
16  "bookedAt": "2024-10-24T10:06:55.305Z"
```

* Postbot Runner Start Proxy Cookies Vault Trash

Get All details of buses on a given date

HTTP BookMyBus / Get Bus Status Save Share

GET http://localhost:5000/api/671a0189acd960bd46fcced1/getStatus Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (8) Test Results 200 OK • 90 ms • 348 B | e.g. ...

Pretty Raw Preview Visualize JSON

```
1 {  
2   "busId": "671a0189acd960bd46fcced1",  
3   "occupancyPercentage": "8.00",  
4   "color": "Green"  
5 }
```

Get the seat filled status of a Bus

