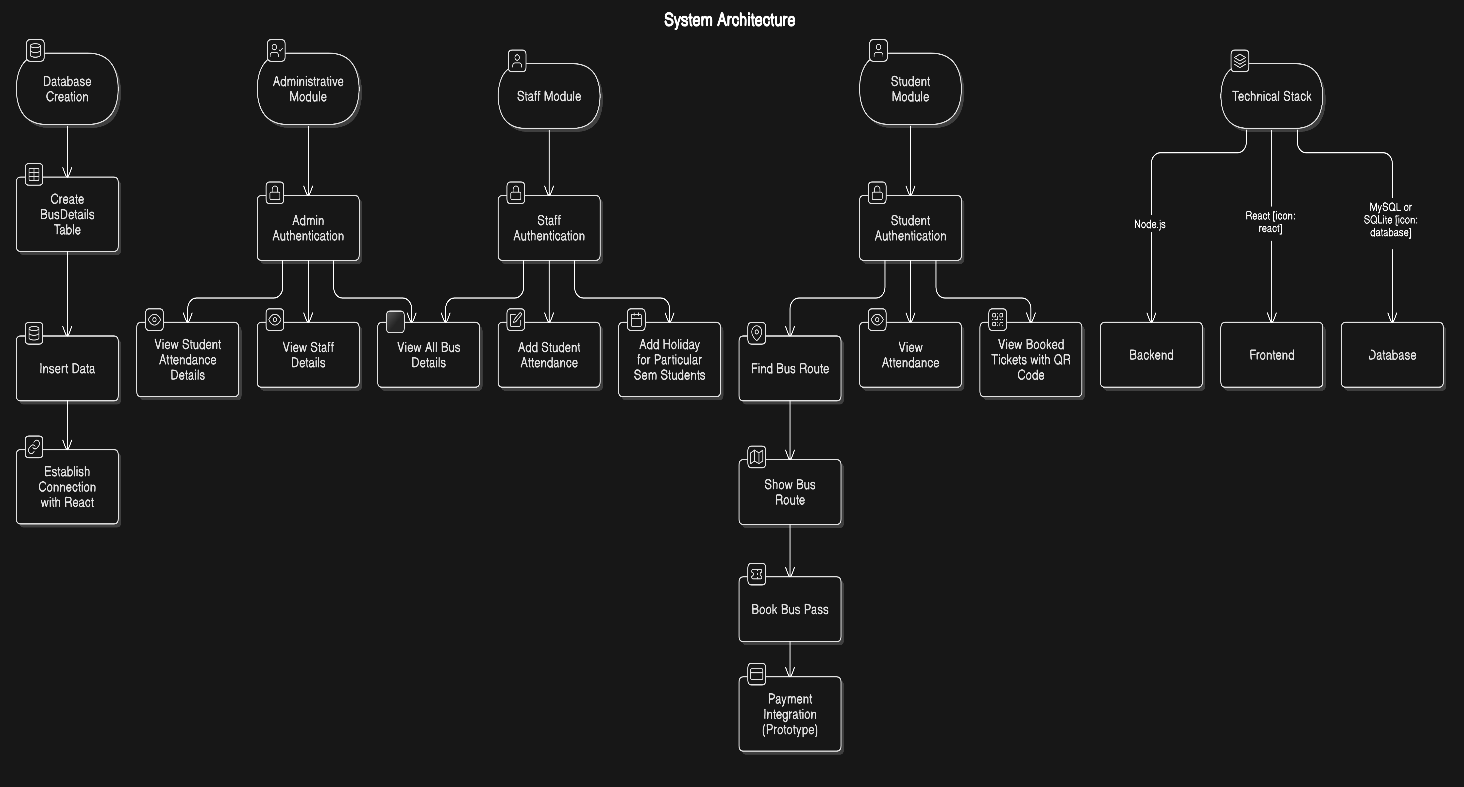
**Buss Pass Management System**

1. System Architecture (Block Diagram):



2. Modules and Algorithms

1. Student Module
2. Admin Module
3. Staff Module

*Student Module*

**ALGORITHM - Login**

1. Start

2. Component Setup

2.1 Import React, useState, useNavigate, and Link.

2.2 Define the LoginPage functional component.

2.3 Initialize state variables: email, password, loading.

2.4 Setup useNavigate for redirection.

3. Form Handling

3.1 Define handleLogin function.

3.2 Prevent form submission if email or password is empty.

3.3 Set loading state to true.

4. API Request

4.1 Send a POST request to "/auth/login" with email and password.

4.2 Parse the response JSON.

4.3 Stop loading.

5. Handle Response

5.1 If login is successful:

5.1.1 Store auth token, role, and user ID in session storage.

5.1.2 Redirect user based on role:

- Student → "/findBus"

- Admin → "/admin/viewallbus"

- Staff → "/staff/busview"

5.2 If login fails, display an error message.

6. Rendering

6.1 Display a login form with:

- Email input

- Password input

- Sign-in button (displays "Logging in..." when loading)

6.2 Provide a sign-up link for new users.

7. Stop

**ALGORITHM - Sign Up**

1. Start

2. Component Setup

2.1 Import React, useState, useNavigate, and Link.

2.2 Define the SignUpPage functional component.

2.3 Initialize state variables: name, email, password, role, courseName, semester.

2.4 Setup useNavigate for redirection.

2.5 Define roles, courses, and semesters arrays.

3. Form Handling

3.1 Define handleSignUp function.

3.2 Prevent form submission if fields are empty.

3.3 Construct a JSON payload with user inputs.

3.4 Log the payload for debugging.

4. API Request

4.1 Send a POST request to "/auth/signup" with user details.

4.2 Parse the response JSON.

4.3 If sign-up is successful, show success alert and navigate to login page.

4.4 If sign-up fails, display an error message.

5. Rendering

5.1 Display a sign-up form with:

- Name input

- Email input

- Role dropdown

- Semester dropdown

- Course dropdown

- Password input

5.2 Include a sign-up button.

5.3 Provide a link for users to sign in if they already have an account.

6. Stop

**ALGORITHM - Find Bus Page**

**1. Start**

**2. Component Setup**

**2.1** Import required dependencies: React, useState, useEffect, useNavigate.  
**2.2** Define the FindBus functional component.  
**2.3** Initialize state variables:

* from, to, date → Stores user-selected journey details.
* busId, price, routeId, stopId → Stores selected bus details.
* fromSuggestions, toSuggestions → Stores autocomplete suggestions.
* tripTime → Stores trip time based on the current hour. **2.4** Use useNavigate for redirection.

**3. Auto-Select Default Values**

**3.1** On component mount (useEffect):

* **Determine trip time**:
  + If current time < 3 PM → Set from = "FISAT", tripTime = "morning".
  + Otherwise → Set to = "FISAT", tripTime = "evening".
* **Set travel date**:
  + Before 3 PM → Set date as today.
  + After 3 PM → Set date as tomorrow.

**4. Fetch Bus Stops (Autocomplete)**

**4.1** Define fetchStops(query, setSuggestions) function:

* **Check input length**: If query length < 2, return.
* **Retrieve auth token** from session storage.
* **Send API request** to fetch stops matching query.
* **If successful**:
  + Extract unique stop names.
  + Set busId, price, routeId, stopId based on first result.
* **Handle errors** if API call fails.

**5. Handle Find Bus**

**5.1** Define handleFindNow function:

* **Validate bus selection**:
  + If busId is not set → Show alert: "Please select a bus first".
* **Retrieve auth token** from session storage.
* **Send API request** to fetch seat availability.
* **If successful**:
  + Navigate to /viewBus, passing bus and seat data.
* **If failed**:
  + Show alert: "Failed to fetch seat status".

6. Rendering UI

6.1 Header:

* Displays "Student Bus Portal" logo and navigation links (Home, Attendance, etc.).

6.2 Main Content:

* **Search Form**:
  + From input (Autocomplete with suggestions).
  + To input (Autocomplete with suggestions).
  + Date (Read-only, auto-filled).
  + **"Find Now"** button to fetch available buses.

6.3 Footer:

* Contact details (Email, Phone).
* Quick links (Privacy Policy, Terms of Service).

**7. Stop**

**ALGORITHM - View Bus Details**

1. **Start**
2. **Component Setup**  
   2.1 Import necessary modules: useLocation, useNavigate from react-router-dom, and icons from react-icons/fa.  
   2.2 Define the ViewBus functional component.  
   2.3 Extract seatData, busId, price, routeId, and stopId from location.state.  
   2.4 Initialize useNavigate for redirection.  
   2.5 Log extracted values for debugging.
3. **Data Validation**  
   3.1 If seatData is not available, display an error message and stop rendering further components.
4. **UI Rendering**  
   4.1 Display a **header** containing:  
   - A bus emoji and portal title.  
   - Navigation links: Home, Attendance, Payments, Bus Pass, and Routes.  
   4.2 Display **main content** including:  
   - Bus Number.  
   - Route Name.  
   - Trip Time.  
   - Travel Date.  
   - Vacant Seats.  
   - Day Pass Booked Count.  
   - Day Scholar Users Count.  
   - Seats from Off-Class Students.  
   4.3 Provide a "See Route" button to navigate to viewbusroute with relevant state data.
5. **Navigation Handling**  
   5.1 On clicking the "See Route" button, navigate to /viewbusroute, passing seatData, busId, price, routeId, and stopId as state.
6. **Footer**  
   6.1 Display **Contact Information**: Email and Phone Number.  
   6.2 Provide **Quick Links**: Privacy Policy and Terms of Service.
7. **Stop**

You're right! Let me update the algorithm to explicitly mention the API calls.

**ALGORITHM - Bus Route Details**

1. **Start**
2. **Component Setup**  
   2.1 Import necessary modules (useEffect, useState, useLocation, useNavigate).  
   2.2 Define the BusRoute functional component.  
   2.3 Extract seatData, busId, price, routeId, and stopId from location.state.  
   2.4 Initialize states:  
   - bus (stores bus details).  
   - route (stores route details).  
   - loading (indicates data fetching status).  
   - studentCount (for attendance input).  
   - isSubmitting (to prevent multiple submissions).  
   - isDropdownOpen, isDropdownOpen1 (for menu toggling).  
   2.5 Log extracted data for debugging.
3. **Fetch Bus Details**  
   3.1 Retrieve authToken from sessionStorage.  
   3.2 **Make API Call to Fetch Bus Details**  
   - Send a **GET request** to http://localhost:5000/buses/${busId}.  
   - Include Authorization: Bearer ${token} in headers.  
   - Parse and store response data in bus state.  
   - If the request fails, log an error message.  
   - Update loading state accordingly.
4. **Fetch Route Details (After Bus Data is Available)**  
   4.1 Check if bus.routeId exists.  
   4.2 Retrieve authToken again.  
   4.3 **Make API Call to Fetch Route Details**  
   - Send a **GET request** to http://localhost:5000/routes/${bus.routeId}.  
   - Include Authorization: Bearer ${token} in headers.  
   - Parse and store response data in route state.  
   - If the request fails, log an error message.
5. **UI Rendering**  
   5.1 Display a **loading message** if data is still being fetched.  
   5.2 If bus data is missing, show an error message.  
   5.3 Render the **header** with:  
   - Portal title.  
   - Navigation menu with dropdowns for "View" and "Add" sections.  
   - Logout button (clears session and redirects to login).  
   5.4 Display **bus details**:  
   - Bus number.  
   - Active status.  
   5.5 Render **route visualization**:  
   - Display all stops in a horizontal layout.  
   - Mark the first stop as "From" and the last as "To".  
   5.6 Provide a **"Book Now"** button:  
   - On click, navigate to the checkout page.
6. **Footer**  
   6.1 Display **Contact Information**: Email and Phone Number.  
   6.2 Provide **Quick Links**: Privacy Policy and Terms of Service.
7. **Stop** 🚏

Here’s the **algorithmic breakdown** of the Checkout component, following the structured approach:

**ALGORITHM - Checkout Page**

1. **Start**
2. **Component Setup**  
   2.1 Import necessary modules (useEffect, useState, useLocation, useNavigate).  
   2.2 Define the Checkout functional component.  
   2.3 Extract booking details (seatData, busId, price, routeId, stopId) from location.state.  
   2.4 Initialize states:  
   - userDetails (stores user details).  
   - paymentOption (tracks selected payment method).  
   2.5 Retrieve userId and authToken from sessionStorage.  
   2.6 Log relevant values for debugging.
3. **Fetch User Details**  
   3.1 If userId exists:  
   - **Make API Call**: GET http://localhost:5000/users/${userId}.  
   - Parse response and store user details in userDetails state.  
   - If the request fails, log an error message.
4. **Handle Payment Selection**  
   4.1 Update paymentOption state when user selects a payment method.
5. **Confirm Booking**  
   5.1 Define confirmBooking(bookingId).  
   5.2 **Make API Call**:  
   - PUT http://localhost:5000/day-passes/confirm/${bookingId}.  
   - Include Authorization: Bearer ${token} in headers.  
   - If successful, display a confirmation alert.  
   - If unsuccessful, show an error message.
6. **Handle Payment Process**  
   6.1 Validate seatData and price.  
   6.2 Construct bookingData object:  
   - travelDate, tripTime, stopId, routeId, and price.  
   6.3 **Make API Call**:  
   - POST http://localhost:5000/day-passes/booking.  
   - Include Authorization: Bearer ${token} in headers.  
   - Send bookingData in the request body.  
   6.4 If successful:  
   - Extract bookingId from the response.  
   - Call confirmBooking(bookingId).  
   - Navigate to /paymentstatus.  
   6.5 If unsuccessful, display an error alert.
7. **UI Rendering**  
   7.1 Display **Header**:  
   - "Student Bus Portal" title.  
   - Navigation menu with links (Home, Attendance, Payments, Bus Pass, Routes).  
   7.2 Show **Booking Details**:  
   - User name, email, phone, and drop-off address.  
   7.3 Render **Payment Options**:  
   - Radio buttons for "Card" and "UPI".  
   - Conditional input fields for each method.  
   7.4 Display **Order Summary**:  
   - Ticket price.  
   - Total price.  
   7.5 Render **Pay Now** button.
8. **Handle Footer Section**  
   8.1 Display **Contact Information**: Email and phone number.  
   8.2 Provide **Quick Links**: Privacy Policy and Terms of Service.
9. **Stop** 🚏