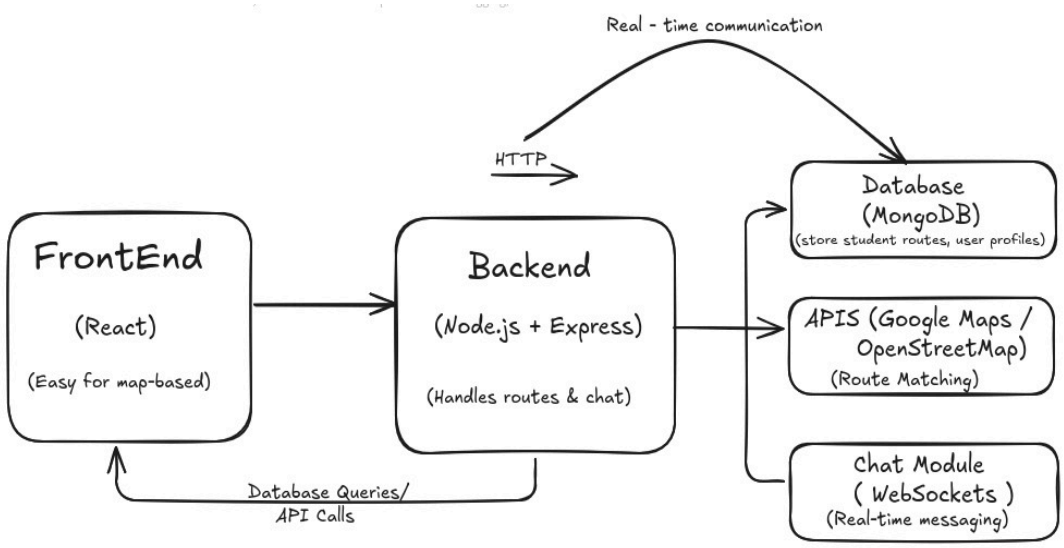
Problem Statements:- Student Commute Optimizer (Full Stack)

How It Works:-

- **Frontend**: The student-facing interface uses a simple map where users input their home and destination. The map shows their route and nearby students traveling in a similar direction. Students can click on an icon to chat directly with other users. All student identities are anonymous, using a unique, non-duplicable username.
- **Backend**: The backend system compares the routes of multiple students to find overlaps or proximity in their travel paths. It then suggests potential matches for carpooling or ride-sharing. The system is designed to hide the student's identity under a unique, non-duplicable username.

Architecture Diagram:-

(Frontend ↔ Backend ↔ Database + APIs + Chat Module)



Technology Stack Choices & Justification:-

1. Frontend: React

Why: React is used for its **component-based** architecture, which is ideal for building the map-centric UI and its various interactive elements like student icons and chat widgets. Its **rich ecosystem** of libraries makes map integration and state management straightforward, while its virtual DOM ensures a **performant and responsive** user experience.

Pseudo-code:

```
// Main App Component

const App = () => {
  // State for map and user data
  const [userLocation, setUserLocation] = useState({});
  const [nearbyStudents, setNearbyStudents] = useState([]);

  // Function to fetch nearby students from the backend
  const fetchNearbyStudents = async () => {
    const response = await fetch('/api/find-matches');
    const data = await response.json();
    setNearbyStudents(data);
  };

  return (
    <div>
      <MapComponent location={userLocation}>
        {nearbyStudents.map(student => (
          <StudentIcon key={student.id} data={student} onClick={openChat} />
        ))}
      </MapComponent>
      <ChatComponent />
    </div>
  );
};
```

---

2. Backend: Node.js + Express

Why: Node.js with Express is a solid choice because it allows for a **unified JavaScript stack** from frontend to backend. Its **non-blocking I/O** model efficiently handles multiple concurrent connections, which is essential for real-time features like route matching and chat.

Pseudo-code:

```
// Express Backend
const express = require('express');
const app = express();
const { findMatches } = require('./routeMatcher');

// API endpoint to find student matches
app.get('/api/find-matches', async (req, res) => {
  const { start, end } = req.query;
```

```
  const matches = await findMatches(start, end);
  res.json(matches);
});

// WebSocket setup for chat module
const http = require('http').createServer(app);
const io = require('socket.io')(http);

io.on('connection', (socket) => {
  socket.on('sendMessage', (message) => {
    io.emit('receiveMessage', message);
  });
});
```

---

3. Database: MongoDB

Why: MongoDB is used because its flexible **NoSQL document model** easily stores user profiles and dynamic route data. Most importantly, it has powerful **geospatial querying capabilities** that are central to finding students traveling along similar paths.

Pseudo-code:

```
// MongoDB query to find nearby students
const findNearbyStudents = async (userRoute) => {
  const matches = await db.collection('students').aggregate([
    {
      $geoNear: {
        near: { type: "Point", coordinates: userRoute.startCoordinates },
        distanceField: "dist.calculated",
        maxDistance: 5000 // 5km radius}
    },
    {
      $match: {
        // Additional logic to find students with similar end destinations
        'destination.coordinates': {
          $nearSphere: {
            $geometry: {
              type: "Point",
              coordinates: userRoute.endCoordinates
            },
            $maxDistance: 5000
          }
        }
```

```
    }
   }
  ]).toArray();
  return matches;
};
```

---

4. Maps API: Google Maps / OpenStreetMap

Why: These APIs are essential for **geocoding** user inputs, **calculating routes**, and **rendering interactive maps** on the frontend, which are core functionalities of the application.

Pseudo-code:

```
// Using a Directions API to get a route
const getRoute = async (start, end) => {
  const response = await
fetch(`https://maps.googleapis.com/maps/api/directions/json?origin=${start}&destination=${end}
&key=YOUR_API_KEY`);
  const data = await response.json();
  return data.routes[0].overview_polyline.points; // Returns the encoded route line
};
```

---

5. Chat: WebSockets

Why: WebSockets provide a **persistent, full-duplex connection** between the client and server. This enables **real-time communication** with **low latency**, which is crucial for the instant messaging functionality.

Pseudo-code:

```
// Frontend (React) Chat Component
useEffect(() => {
  const socket = io('http://localhost:3000');

  socket.on('receiveMessage', (message) => {
    // Append message to chat window
  });
  return () => socket.disconnect();
}, []);
const sendMessage = (message) => {
  socket.emit('sendMessage', { text: message, sender: 'user_id' });
};
```