School Transport Backend Documentation

1. Authentication

Register User
URL: /api/auth/register
Method: POST
Body: { "name": "John Doe", "email": "john@example.com", "phone": "1234567890", "password": "password123", "role": "passenger" }
Response: { "_id": "userId", "name": "John Doe", "email": "john@example.com", "role": "passenger", "token": "JWT_TOKEN" }

Login User
URL: /api/auth/login
Method: POST
Body: { "email": "john@example.com", "password": "password123" }
Response: { "_id": "userId", "name": "John Doe", "email": "john@example.com", "role": "passenger", "token": "JWT_TOKEN" }

2. Users (Admin Only)

Get All Users: GET /api/users/
Get Single User: GET /api/users/:id
Update User: PUT /api/users/:id
Delete User: DELETE /api/users/:id

3. Trips (Admin Only)

Create Trip: POST /api/trips/
Body: { "driver": "driverId", "routeName": "Route A", "stops": [{ "name": "Stop 1", "latitude": 12.34, "longitude": 56.78, "passengers": ["passengerId1"] }] }
Get All Trips: GET /api/trips/
Get Single Trip: GET /api/trips/:id
Update Trip: PUT /api/trips/:id
Delete Trip: DELETE /api/trips/:id

4. Messages

Send Message: POST /api/messages/
Body: { "receiverId": "userId", "text": "Hello" }

Get Messages: GET /api/messages/:userId

5. Notifications

Send Notification: POST /api/notifications/
Body: { "title": "Bus Alert", "body": "Bus arriving in 5 minutes", "receivers": ["passengerId1", "passengerId2"] }

Get Notifications: GET /api/notifications/

6. Socket.IO Events

Authentication: io({ auth: { token: "JWT_TOKEN" } });

Chat:
Send: chat:send { "receiverId": "userId", "text": "Hello" }
Receive: chat:receive { "_id": "messageId", "sender": "userId", "receiver": "userId", "text": "Hello", "createdAt": "2025-11-11T06:00:00.000Z" }

Driver Location:
Send: driver:location:update { "tripId": "tripId", "latitude": 12.34, "longitude": 56.78 }
Receive: driver:location { "driverId": "driverId", "latitude": 12.34, "longitude": 56.78 }

7. Middleware & Security

- protect: JWT authentication
- authorizeRoles: Role-based access (admin, driver, passenger)
- Helmet: Security headers
- Compression: Response compression
- Rate-limit: Prevent abuse (200 requests/min)

8. Cluster & Scalability

- Master process handles multiple workers (os.cpus().length - 1)
- Workers automatically restart if they crash
- Redis adapter for Socket.IO scaling
- Real-time tracking and chat work across workers

9. Notes

- All models are MongoDB/Mongoose
- DriverLocation updates are upserted for real-time tracking
- Messages and notifications stored in DB for admin reporting
- Socket.IO for bidirectional real-time communication
- FCM pushes notifications to passengers/drivers