SafeDrive App - Complete Backend Setup Instructions

Prerequisites

- Node.js 18+ installed
- PostgreSQL database
- Git
- Code editor (VS Code recommended)

Step 1: Backend Setup

1.1 Create Backend Project

```bash
mkdir safedrive-backend
cd safedrive-backend
npm init -y
```

1.2 Install Backend Dependencies

```bash
# Core dependencies
npm install express cors helmet express-rate-limit dotenv
npm install @prisma/client prisma zod axios
npm install bcryptjs jsonwebtoken

# TypeScript dependencies
npm install -D typescript @types/node @types/express
npm install -D @types/cors @types/bcryptjs @types/jsonwebtoken
npm install -D ts-node nodemon concurrently

# Development tools
npm install -D eslint prettier @typescript-eslint/eslint-plugin
```

1.3 Setup TypeScript Configuration

Create `tsconfig.json`:

```json
```

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "lib": ["ES2020"],
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "resolveJsonModule": true
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "dist"]
}
```

1.4 Setup Package.json Scripts

Add to `package.json`:

```json
{
  "scripts": {
    "dev": "nodemon src/server.ts",
    "build": "tsc",
    "start": "node dist/server.js",
    "prisma:generate": "prisma generate",
    "prisma:push": "prisma db push",
    "prisma:studio": "prisma studio"
  }
}
```

1.5 Create Environment File

Create `.env`:

```env
```

```
# Database
DATABASE_URL="postgresql://username:password@localhost:5432/safedrive_db"

# Server
PORT=5000
NODE_ENV=development
FRONTEND_URL=http://localhost:3000

# Security
JWT_SECRET=your-super-secret-jwt-key-here
BCRYPT_ROUNDS=12

# External APIs (for future use)
WAZE_API_KEY=your-waze-api-key
GOOGLE_MAPS_API_KEY=your-google-maps-api-key
PROGRESSIVE_API_KEY=your-progressive-api-key
```

1.6 Initialize Prisma

```
bash

npx prisma init
```

1.7 Setup Database Schema

Replace `prisma/schema.prisma` with the provided schema from earlier.

1.8 Create Database and Tables

```
bash

# Generate Prisma client
npx prisma generate

# Push schema to database
npx prisma db push

# Optional: Open Prisma Studio to view database
npx prisma studio
```

1.9 Create Project Structure

```
bash
```

```
mkdir -p src/routes src/middleware src/services src/types
```

1.10 Add Additional Route Files

Create the remaining route files:

src/routes/vehicles.ts:

```typescript
```

```typescript
import express from 'express';
import { PrismaClient } from '@prisma/client';
import { z } from 'zod';

const router = express.Router();
const prisma = new PrismaClient();

const createVehicleSchema = z.object({
  userId: z.string(),
  make: z.string().optional(),
  model: z.string().optional(),
  year: z.string().optional(),
  vin: z.string().optional(),
  licensePlate: z.string().optional(),
  color: z.string().optional()
});

router.post('/', async (req, res) => {
  try {
    const validatedData = createVehicleSchema.parse(req.body);
    const vehicle = await prisma.vehicle.create({ data: validatedData });
    res.status(201).json(vehicle);
  } catch (error) {
    res.status(500).json({ error: 'Failed to create vehicle' });
  }
});

router.get('/user/:userId', async (req, res) => {
  try {
    const { userId } = req.params;
    const vehicles = await prisma.vehicle.findMany({ where: { userId } });
    res.json(vehicles);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch vehicles' });
  }
});

export default router;
```

src/routes/insurance.ts:

```typescript
```

```typescript
import express from 'express';
import { PrismaClient } from '@prisma/client';
import { z } from 'zod';

const router = express.Router();
const prisma = new PrismaClient();

const createInsuranceSchema = z.object({
  userId: z.string(),
  provider: z.string().optional(),
  policyNumber: z.string().optional(),
  groupNumber: z.string().optional(),
  effectiveDate: z.string().optional(),
  expirationDate: z.string().optional(),
  coverageType: z.string().optional(),
  deductible: z.string().optional()
});

router.post('/', async (req, res) => {
  try {
    const validatedData = createInsuranceSchema.parse(req.body);
    const insurance = await prisma.insurance.create({ data: validatedData });
    res.status(201).json(insurance);
  } catch (error) {
    res.status(500).json({ error: 'Failed to create insurance record' });
  }
});

router.get('/user/:userId', async (req, res) => {
  try {
    const { userId } = req.params;
    const insurance = await prisma.insurance.findMany({ where: { userId } });
    res.json(insurance);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch insurance records' });
  }
});

export default router;
```

src/routes/alerts.ts:

```typescript
```

```
import express from 'express';
import { PrismaClient } from '@prisma/client';

const router = express.Router();
const prisma = new PrismaClient();

router.get('/user/:userId', async (req, res) => {
  try {
    const { userId } = req.params;
    const { unreadOnly } = req.query;

    const where: any = { userId };
    if (unreadOnly === 'true') {
      where.isRead = false;
    }

    const alerts = await prisma.alert.findMany({
      where,
      orderBy: { timestamp: 'desc' }
    });
    res.json(alerts);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch alerts' });
  }
});

router.put('/:id/read', async (req, res) => {
  try {
    const { id } = req.params;
    const alert = await prisma.alert.update({
      where: { id },
      data: { isRead: true }
    });
    res.json(alert);
  } catch (error) {
    res.status(500).json({ error: 'Failed to mark alert as read' });
  }
});

export default router;
```

Step 2: Frontend Integration

2.1 Create Frontend Project

```bash
# In a new terminal/directory
pnpm create vite safedrive-frontend -- --template react-swc-ts
cd safedrive-frontend
pnpm install
```

2.2 Install Frontend Dependencies

```bash
pnpm add lucide-react axios
pnpm add -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

2.3 Configure Tailwind CSS

Update `tailwind.config.js`:

```javascript
export default {
  content: ["./index.html", "./src/**/*.{js,ts,jsx,tsx}"],
  darkMode: 'class',
  theme: { extend: {} },
  plugins: [],
}
```

Update `src/index.css`:

```css
@tailwind base;
@tailwind components;
@tailwind utilities;
```

2.4 Create Environment File

Create `.env` in frontend:

```env
VITE_API_BASE_URL=http://localhost:5000/api
```

2.5 Add API Service

Create `src/services/api.ts` with the provided API service code.

2.6 Update Your SafeDrive Component

Replace your existing SafeDriveApp component to use the API service:

```typescript
```

```tsx
import { useEffect, useState } from 'react';
import apiService from './services/api';

// Add at the top of your component:
const [userId] = useState('user_123'); // Replace with actual user ID
const [currentTripId, setCurrentTripId] = useState<string | null>(null);

// Replace your existing functions:
const startTrip = async () => {
  try {
    const tripData = {
      userId,
      vehicleId: vehicleInfo.make ? 'vehicle_id' : undefined,
      mapProvider,
      startTime: new Date().toISOString()
    };

    const trip = await apiService.startTrip(tripData);
    setCurrentTripId(trip.tripId);
    setIsTracking(true);
    setCurrentSpeed(25);
    setAlerts([]);
    // ... rest of your existing logic
  } catch (error) {
    console.error('Failed to start trip:', error);
  }
};

const exportData = async () => {
  if (!currentTripId) return;

  try {
    const result = await apiService.exportTrip(currentTripId, insuranceInfo.provider || 'Unknown');
    console.log('Export successful:', result);

    setAlerts(prev => [...prev, {
      id: Date.now(),
      type: 'success',
      message: 'Complete profile and trip data exported',
      timestamp: new Date().toLocaleTimeString()
    }]);
  } catch (error) {
    console.error('Export failed:', error);
```

```
  }
};
```

Step 3: Running the Application

3.1 Start Backend Server

```bash
cd safedrive-backend
npm run dev
```

Server will run on http://localhost:5000

3.2 Start Frontend Development Server

```bash
cd safedrive-frontend
pnpm dev
```

Frontend will run on http://localhost:3000

3.3 Test API Connection

Visit http://localhost:5000/api/health to verify backend is running.

Step 4: Database Management

4.1 View Database

```bash
cd safedrive-backend
npx prisma studio
```

4.2 Reset Database (if needed)

```bash
npx prisma db push --force-reset
```

Step 5: Production Deployment

5.1 Build Backend

```bash
npm run build
npm start
```

5.2 Build Frontend

```bash
pnpm build
pnpm preview
```

Step 6: Testing API Endpoints

Test with curl or Postman:

Create User:

```bash
curl -X POST http://localhost:5000/api/users \
  -H "Content-Type: application/json" \
  -d '{"email":"test@example.com","firstName":"John","lastName":"Doe"}'
```

Start Trip:

```bash
curl -X POST http://localhost:5000/api/trips/start \
  -H "Content-Type: application/json" \
  -d '{"userId":"USER_ID","startTime":"2024-01-01T10:00:00Z"}'
```

Troubleshooting

Common Issues:

1. **Database Connection**: Verify PostgreSQL is running and credentials are correct
2. **CORS Errors**: Ensure frontend URL is in CORS configuration
3. **Port Conflicts**: Change ports in .env if 5000/3000 are occupied

4. **TypeScript Errors**: Run `npx prisma generate` after schema changes

Development Tips:

- Use Prisma Studio for database inspection
- Check browser Network tab for API call details
- Monitor backend console for error logs
- Use TypeScript strict mode for better error catching

This setup provides a complete, production-ready backend with full CRUD operations, real-time trip tracking, and insurance data export capabilities.