# 🗺️ Complete Codebase Map - Transportation Management System

**DEBUGGING GUIDE FOR IT TEAM**

This document provides a complete mapping of every feature, function, and code file in the Transportation Management System. Use this to quickly locate and fix any issues.

---

# 🛠️ BACKEND CODEBASE (Node.js/Express)

## 📁 Server Entry Point

**File**: `backend/index.js`

- **Purpose**: Main server startup and configuration
- **Key Functions**:
  - `app.listen(PORT)` - Starts server on port 5002
  - `mongoose.connect(URI)` - Connects to MongoDB
  - CORS configuration for frontend communication
- **Routes Mounted**:
  - `/api` → auth routes
  - `/api/drivers` → driver routes
  - `/api/vehicles` → vehicle routes
  - `/api/tripRequest` → trip request routes
- **Debug**: Check server startup logs, MongoDB connection errors

## 📁 Authentication System

**File**: `backend/routes/auth.js`

- **Purpose**: User registration, login, and session management
- **Key Functions**:
  - `POST /api/signup` - Employee registration with OTP
  - `POST /api/verify-otp` - OTP verification and account creation
  - `POST /api/login` - User authentication

    - `GET /api/me` - Get current user info
    - `POST /api/logout` - User logout
- **Helper Functions**:
    - `generateOtp()` - Creates 6-digit OTP
    - `pendingSignups` - In-memory store for OTP verification
- **Debug**: Check OTP generation, email sending, JWT token creation

**File**: `backend/middleware/secureRoute.js`

- **Purpose**: Protects routes requiring authentication
- **Key Functions**:
    - `secureRoute()` - Middleware function
    - JWT token verification
    - User loading from database
- **Debug**: Check JWT token validity, cookie parsing

**File**: `backend/jwt/generateToken.js`

- **Purpose**: JWT token generation and cookie management
- **Key Functions**:
    - `createTokenAndSaveCookie(userId, res)` - Creates JWT and sets HTTP-only cookie
- **Debug**: Check token expiration, cookie settings

## 📁 Trip Request Management

**File**: `backend/routes/trip.request.js`

- **Purpose**: Complete trip request lifecycle management
- **Key Functions**:
    - `POST /api/tripRequest` - Create new trip request
    - `GET /api/tripRequest` - Get all trips (filtered by user role)
    - `POST /api/tripRequest/:id/approve` - Approve and assign resources
    - `POST /api/tripRequest/:id/reject` - Reject with remarks
    - `POST /api/tripRequest/:id/complete` - Mark as completed
- **Debug**: Check request validation, resource assignment, notification sending

**File**: `backend/models/trip.request.model.js`

- **Purpose**: Trip request database schema
- **Key Fields**:
    - `purpose` (Official/Personal)
    - `designation` (Unit Head, Management, etc.)
    - `destination`, `pickupPoint`, `startDate`, `endDate`
    - `status` (Pending/Approved/Rejected/Completed)
    - `vehicleDetails` (assigned driver and vehicle info)
- **Debug**: Check data validation, field types, relationships

## 📁 Driver Management

**File**: `backend/routes/driver.route.js`

- **Purpose**: Driver CRUD operations
- **Key Functions**:
    - `GET /api/drivers` - Get all drivers
    - `POST /api/drivers` - Create new driver
    - `DELETE /api/drivers/:id` - Delete driver
    - `PATCH /api/drivers/:id/toggleUnavailable` - Toggle availability
- **Debug**: Check driver validation, status updates

**File**: `backend/models/driver.model.js`

- **Purpose**: Driver database schema
- **Key Fields**:
    - `driverName`, `age`, `phoneNo`, `licenseNo`
    - `status` (assigned/available)
    - `temporarilyUnavailable` (boolean)
- **Debug**: Check age validation, license uniqueness

## 📁 Vehicle Management

**File**: `backend/routes/vehicle.route.js`

- **Purpose**: Vehicle CRUD operations
- **Key Functions**:
    - `GET /api/vehicles` - Get all vehicles

    - `POST /api/vehicles` - Create new vehicle
    - `DELETE /api/vehicles/:id` - Delete vehicle
    - `PATCH /api/vehicles/:id/toggleStatus` - Toggle out-of-service
- **Debug**: Check vehicle validation, capacity handling

**File**: `backend/models/vehicle.model.js`

- **Purpose**: Vehicle database schema
- **Key Fields**:
    - `vehicleName` , `capacity` , `vehicleNo` , `vehicleColor`
    - `status` (Assigned/Available)
    - `outOfService` (boolean)
- **Debug**: Check vehicle number uniqueness, capacity validation

## 📁 Employee Management

**File**: `backend/models/employee.model.js`

- **Purpose**: Employee database schema
- **Key Fields**:
    - `employeeId` , `name` , `email` , `department`
    - `password` (bcrypt hashed)
    - `role` (employee/admin)
- **Debug**: Check email domain validation, password hashing

## 📁 Notification System

**File**: `backend/utils/mailer.js`

- **Purpose**: Email notifications using Nodemailer
- **Key Functions**:
    - `sendMail(to, subject, text)` - Sends email
- **Configuration**:
    - Uses Gmail SMTP
    - Environment variables: `ADMIN_USER` , `ADMIN_PASS`
- **Debug**: Check SMTP credentials, email delivery

**File**: `backend/utils/twiliosms.js`

- **Purpose**: SMS notifications using Twilio
- **Key Functions**:
  - `sendSMS(to, message)` - Sends SMS
- **Configuration**:
  - Environment variables: `TWILIO_SID`, `TWILIO_AUTH_TOKEN`, `TWILIO_PHONE_NUMBER`
- **Debug**: Check Twilio credentials, phone number format

## 📁 Automated Operations

**File**: `backend/Scheduler/scheduler.js`

- **Purpose**: Automated trip completion and resource management
- **Key Functions**:
  - Cron job runs every minute
  - Finds completed trips and releases resources
  - Updates driver and vehicle status
- **Debug**: Check cron job execution, resource release logic

---

# 🎨 FRONTEND CODEBASE (React)

## 📁 Application Entry

**File**: `frontend/src/main.jsx`

- **Purpose**: React app entry point
- **Key Functions**:
  - Renders `<App />` component
  - Imports CSS files

**File**: `frontend/src/App.jsx`

- **Purpose**: Main routing and authentication logic
- **Key Functions**:
  - `fetchCurrentUser()` - Checks authentication status

    - Route protection based on user role
    - Redirects to appropriate dashboard
- **Routes**:
    - `/login` → Login component
    - `/signup` → Signup component
    - `/employee-dashboard` → Employee dashboard
    - `/admin-dashboard` → Admin dashboard
- **Debug**: Check authentication state, route protection

## 📁 Authentication Components

**File**: `frontend/src/Components/Login.jsx`

- **Purpose**: User login interface
- **Key Functions**:
    - `handleSubmit()` - Login form submission
    - Supports both email and employee ID login
- **State Variables**:
    - `email`, `employeeId`, `password`, `errors`
- **Debug**: Check form validation, API calls, error handling

**File**: `frontend/src/Components/Signup.jsx`

- **Purpose**: Employee registration with OTP
- **Key Functions**:
    - `handleSubmit()` - Initial signup request
    - `handleOtpSubmit()` - OTP verification
- **State Variables**:
    - `showOtpInput`, `otp`, `loading`, `errors`
- **Debug**: Check form validation, OTP flow, API responses

## 📁 State Management (Zustand Stores)

**File**: `frontend/src/store/useAuthStore.js`

- **Purpose**: Authentication state management
- **Key Functions**:

    - `setUser(user)` - Set current user
    - `logout()` - Clear user session
    - `fetchCurrentUser()` - Get user from server
    - `signup()` - Handle registration
- **State**: `user`, `isLoggedIn`
- **Debug**: Check authentication state, API calls

**File**: `frontend/src/store/useTripRequestStore.js`

- **Purpose**: Trip request state management
- **Key Functions**:
    - `fetchTripRequests()` - Get all requests
    - `addTripRequest()` - Create new request
    - `approveTripRequest()` - Approve request
    - `rejectTripRequest()` - Reject request
    - `completeTripRequest()` - Mark as completed
- **State**: `tripRequests`, `loading`, `error`
- **Debug**: Check API calls, state updates, error handling

**File**: `frontend/src/store/useVehicleStore.js`

- **Purpose**: Vehicle state management
- **Key Functions**:
    - `fetchVehicles()` - Get all vehicles
    - `addVehicle()` - Create vehicle
    - `deleteVehicle()` - Delete vehicle
- **State**: `vehicles`, `loading`, `error`
- **Debug**: Check CRUD operations, state synchronization

**File**: `frontend/src/store/useDriverStore.js`

- **Purpose**: Driver state management
- **Key Functions**:
    - `fetchDrivers()` - Get all drivers
    - `addDriver()` - Create driver
    - `deleteDriver()` - Delete driver
- **State**: `drivers`, `loading`, `error`

- **Debug**: Check CRUD operations, state updates

## 📁 Employee Dashboard

**File**: `frontend/src/Components/EmployeeDashboard.jsx`

- **Purpose**: Main employee interface
- **Key Functions**:
  - `handleSubmit()` - Create new trip request
  - `handleViewDetails()` - Show request details
  - `handleLogout()` - User logout
- **State Variables**:
  - `showForm`, `formData`, `requests`, `selectedTrip`
- **Debug**: Check form submission, data fetching, UI state

**File**: `frontend/src/Components/EmployeeComponents/NewRequest.jsx`

- **Purpose**: Trip request creation form
- **Key Functions**:
  - Form validation and submission
  - Date and time handling
- **Debug**: Check form validation, API integration

**File**: `frontend/src/Components/EmployeeComponents/ActiveRequest.jsx`

- **Purpose**: Display active trip requests
- **Key Functions**:
  - Show pending and approved requests
  - Real-time status updates
- **Debug**: Check data display, status filtering

**File**: `frontend/src/Components/EmployeeComponents/PastRequest.jsx`

- **Purpose**: Display completed/rejected requests
- **Key Functions**:
  - Show request history
  - Display final status and remarks

- **Debug**: Check data filtering, status display

## 📁 Admin Dashboard

**File**: `frontend/src/Components/AdminDashboard.jsx`

- **Purpose**: Main admin interface with tabs
- **Key Functions**:
  - `handleAddVehicle()` - Add new vehicle
  - `handleAddDriver()` - Add new driver
  - `handleApprove()` - Approve trip requests
  - `handleReject()` - Reject trip requests
  - `handleCompleteTrip()` - Mark trips as completed
- **State Variables**:
  - `activeTab`, `showAddVehicle`, `showAddDriver`
  - `selectedRequest`, `assignmentData`
- **Debug**: Check tab switching, form handling, API calls

**File**: `frontend/src/Components/AdminComponents/ActiveRequest.jsx`

- **Purpose**: Manage active trip requests
- **Key Functions**:
  - Display pending requests
  - Approval/rejection interface
  - Resource assignment
- **Debug**: Check request display, approval flow

**File**: `frontend/src/Components/AdminComponents/PastRequest.jsx`

- **Purpose**: View completed/rejected requests
- **Key Functions**:
  - Request history display
  - Export functionality
- **Debug**: Check data display, export features

**File**: `frontend/src/Components/AdminComponents/VehicleManage.jsx`

- **Purpose**: Vehicle management interface
- **Key Functions**:
    - Add/delete vehicles
    - Toggle out-of-service status
    - Vehicle list display
- **Debug**: Check CRUD operations, status updates

**File**: `frontend/src/Components/AdminComponents/DriverManage.jsx`

- **Purpose**: Driver management interface
- **Key Functions**:
    - Add/delete drivers
    - Toggle availability
    - Driver list display
- **Debug**: Check CRUD operations, availability updates

**File**: `frontend/src/Components/AdminComponents/Header.jsx`

- **Purpose**: Admin dashboard header
- **Key Functions**:
    - User info display
    - Logout functionality
- **Debug**: Check user display, logout flow

**File**: `frontend/src/Components/AdminComponents/Navbar.jsx`

- **Purpose**: Admin navigation sidebar
- **Key Functions**:
    - Tab switching
    - Navigation state management
- **Debug**: Check navigation, active tab highlighting

## 📁 Configuration Files

**File**: `frontend/vite.config.js`

- **Purpose**: Vite build configuration

- **Key Settings**:
    - React plugin configuration
    - Build optimization
- **Debug**: Check build process, development server

**File**: `frontend/tailwind.config.js`

- **Purpose**: Tailwind CSS configuration
- **Key Settings**:
    - Content paths
    - Custom theme settings
- **Debug**: Check CSS compilation, styling issues

**File**: `frontend/postcss.config.js`

- **Purpose**: PostCSS configuration
- **Key Settings**:
    - Tailwind CSS plugin
    - Autoprefixer
- **Debug**: Check CSS processing

**File**: `frontend/eslint.config.js`

- **Purpose**: ESLint configuration
- **Key Settings**:
    - React rules
    - Code formatting rules
- **Debug**: Check code quality, linting errors

---

# 🔍 COMPREHENSIVE DEBUGGING GUIDE

## Authentication Issues

**Symptoms**: Login fails, OTP not received, session lost

**Check These Files**:

1. `backend/routes/auth.js` - Login/signup logic

2. `backend/middleware/secureRoute.js` - Route protection
3. `backend/jwt/generateToken.js` - Token generation
4. `frontend/src/store/useAuthStore.js` - Frontend auth state
5. `frontend/src/Components/Login.jsx` - Login form
6. `frontend/src/Components/Signup.jsx` - Signup form
7. `backend/utils/mailer.js` - OTP email delivery

## Trip Request Issues

**Symptoms**: Can't create requests, approval fails, status not updating

**Check These Files**:

1. `backend/routes/trip.request.js` - Request API logic
2. `backend/models/trip.request.model.js` - Data validation
3. `frontend/src/store/useTripRequestStore.js` - Frontend state
4. `frontend/src/Components/EmployeeComponents/NewRequest.jsx` - Request creation
5. `frontend/src/Components/AdminComponents/ActiveRequest.jsx` - Request management
6. `backend/Scheduler/scheduler.js` - Auto-completion

## Vehicle/Driver Management Issues

**Symptoms**: Can't add vehicles/drivers, status not updating

**Check These Files**:

1. `backend/routes/vehicle.route.js` - Vehicle API
2. `backend/routes/driver.route.js` - Driver API
3. `backend/models/vehicle.model.js` - Vehicle validation
4. `backend/models/driver.model.js` - Driver validation
5. `frontend/src/store/useVehicleStore.js` - Vehicle state
6. `frontend/src/store/useDriverStore.js` - Driver state
7. `frontend/src/Components/AdminComponents/VehicleManage.jsx` - Vehicle UI
8. `frontend/src/Components/AdminComponents/DriverManage.jsx` - Driver UI

## Notification Issues

**Symptoms**: No email/SMS notifications

**Check These Files**:

1. `backend/utils/mailer.js` - Email configuration
2. `backend/utils/twiliosms.js` - SMS configuration
3. Environment variables in `.env` files
4. `backend/routes/trip.request.js` - Notification triggers

## UI/UX Issues

**Symptoms**: Components not rendering, styling problems

**Check These Files**:

1. `frontend/src/App.jsx` - Routing logic
2. `frontend/src/Components/` - Component files
3. `frontend/tailwind.config.js` - CSS configuration
4. `frontend/src/index.css` - Global styles

## API/Network Issues

**Symptoms**: Frontend can't connect to backend

**Check These Files**:

1. `backend/index.js` - Server startup
2. `frontend/src/store/` - API calls
3. Environment variables for URLs
4. CORS configuration in backend

## Database Issues

**Symptoms**: Data not saving, connection errors

**Check These Files**:

1. `backend/index.js` - MongoDB connection
2. `backend/models/` - Schema definitions
3. Environment variables for database URI
4. MongoDB Atlas configuration

---

# 🛠️ DEBUGGING COMMANDS

## Backend Debugging

```
# Check server logs
pm2 logs tms-backend

# Check specific error
pm2 logs tms-backend --lines 100

# Restart backend
pm2 restart tms-backend

# Check MongoDB connection
# Look for connection errors in logs
```

## Frontend Debugging

```
# Check build errors
cd frontend
npm run build

# Check development server
npm run dev

# Check browser console (F12)
# Look for JavaScript errors, API call failures
```

## Database Debugging

```
# Check MongoDB connection
# Use MongoDB Compass to connect and verify data

# Check environment variables
echo $MONGODB_URI
```

---

# 📞 SUPPORT ESCALATION

1. **Check this codebase map** for the relevant files
2. **Use debugging commands** to identify the issue
3. **Check environment variables** and configuration
4. **Review recent changes** to the codebase
5. **Contact developer** with specific error details and file locations

**Remember**: Most issues can be traced to specific files using this map. Start with the symptoms and work backwards through the relevant code sections.