🚚 Transportation Management System

A comprehensive full-stack web application for managing vehicle requests, driver assignments, and fleet operations within an organization. Built with Node.js/Express (backend) and React (frontend) with real-time updates, automated scheduling, and multi-channel notifications.

📋 System Overview

This Transportation Management System (TMS) is designed for organizations to efficiently manage their fleet operations. It provides separate interfaces for employees to request transportation and administrators to manage the entire fleet, including vehicles, drivers, and trip requests.

🎯 Core Functionality

- **Employee Portal**: Submit and track transportation requests
- **Admin Portal**: Comprehensive fleet management and request approval system
- **Real-time Updates**: Live status tracking and notifications
- **Automated Operations**: Scheduled trip completion and resource management
- **Multi-channel Notifications**: Email and SMS alerts for status changes
- **Role-based Access Control**: Secure authentication with JWT tokens

---

🏗️ Architecture

Backend Architecture

- **Framework**: Node.js with Express.js
- **Database**: MongoDB with Mongoose ODM
- **Authentication**: JWT-based with secure HTTP-only cookies
- **Real-time Updates**: Polling-based with 2-5 second intervals
- **Scheduling**: Node-cron for automated trip completion
- **Notifications**: Nodemailer (email) and Twilio (SMS)

Frontend Architecture

- **Framework**: React 19 with Vite build tool
- **State Management**: Zustand for lightweight state management
- **Routing**: React Router DOM for navigation
- **Styling**: Tailwind CSS with responsive design
- **Icons**: Lucide React for modern iconography
- **HTTP Client**: Axios for API communication

🗄️ Database Models

Employee Model

```
{
  employeeId: String (required, unique, uppercase),
  name: String (required),
  email: String (required, unique, @adityabirla.com domain),
  department: Enum ['HR', 'ER', 'MECHANICAL', 'CIVIL', 'IT', 'CHEMICAL'],
  password: String (bcrypt hashed, min 6 chars),
  phoneNo: String (optional),
  role: String (default: 'employee'),
  timestamps: true
}
```

Driver Model

```
{
  driverName: String (required),
  age: Number (18-65),
  phoneNo: String (required),
  licenseNo: String (required, unique, uppercase),
  status: Enum ['assigned', 'available'] (default: 'available'),
  temporarilyUnavailable: Boolean (default: false),
  timestamps: true
}
```

Vehicle Model

```
{
  vehicleName: String (required),
  capacity: Number (required),
  vehicleNo: String (unique, uppercase),
  vehicleColor: String (required),
  status: Enum ['Assigned', 'Available'] (default: 'Available'),
  outOfService: Boolean (default: false),
  timestamps: true
}
```

Trip Request Model

```
{
  purpose: Enum ['Official', 'Personal'],
  designation: Enum ['Unit Head', 'Functional Head', 'Department Head', 'Sectional H
ead', 'Management', 'Staff', 'Worker'],
  destination: String (required),
  pickupPoint: String (required),
  startDate: Date (required),
  startTime: String (required),
  endDate: Date (required),
  numberOfPassengers: Number (1-20),
  remarks: String (optional),
  status: Enum ['Pending', 'Approved', 'Rejected', 'Completed'] (default: 'Pendin
g'),
  createdBy: ObjectId (ref: Employee),
  vehicleDetails: {
    driverId: ObjectId (ref: Employee),
    vehicleId: ObjectId (ref: Vehicle),
    driverName: String,
    phoneNo: String,
    licenseNo: String,
    vehicleNo: String,
```

```
    vehicleName: String,
    vehicleColor: String,
    isOutside: Boolean (default: false),
    outsideVehicle: { vehicleNo: String, vehicleName: String },
    outsideDriver: { driverName: String, phoneNo: String }
  },
  timestamps: true
}
```

🔌 API Endpoints

Authentication Routes (`/api`)

- `POST /signup` - Employee registration with OTP verification
- `POST /verify-otp` - OTP verification for account creation
- `POST /login` - User authentication (email/employeeId + password)
- `GET /me` - Get current user information
- `POST /logout` - User logout and token invalidation

Trip Request Routes (`/api/tripRequest`)

- `POST /` - Create new trip request
- `GET /` - Get all trip requests (filtered by user role)
- `POST /:id/approve` - Approve trip request and assign resources
- `POST /:id/reject` - Reject trip request with remarks
- `POST /:id/complete` - Mark trip as completed

Driver Management Routes (`/api/drivers`)

- `GET /` - Get all drivers
- `POST /` - Create new driver
- `DELETE /:id` - Delete driver
- `PATCH /:id/toggleUnavailable` - Toggle driver availability

Vehicle Management Routes (`/api/vehicles`)

- `GET /` - Get all vehicles
- `POST /` - Create new vehicle
- `DELETE /:id` - Delete vehicle
- `PATCH /:id/toggleStatus` - Toggle vehicle out-of-service status

🎨 Frontend Components

Core Components

- **App.jsx**: Main routing and authentication logic
- **Login.jsx**: User authentication interface
- **Signup.jsx**: Employee registration with OTP verification
- **EmployeeDashboard.jsx**: Employee portal with request management
- **AdminDashboard.jsx**: Admin portal with comprehensive fleet management

Employee Components

- **NewRequest.jsx**: Trip request creation form
- **ActiveRequest.jsx**: Display pending and approved requests
- **PastRequest.jsx**: Display completed and rejected requests

Admin Components

- **Header.jsx**: Admin dashboard header with user info
- **Navbar.jsx**: Navigation sidebar with tab switching
- **ActiveRequest.jsx**: Manage pending trip requests
- **PastRequest.jsx**: View and manage completed/rejected requests
- **VehicleManage.jsx**: Vehicle CRUD operations
- **DriverManage.jsx**: Driver CRUD operations

State Management (Zustand Stores)

- **useAuthStore.js**: Authentication state and user management
- **useTripRequestStore.js**: Trip request CRUD operations
- **useVehicleStore.js**: Vehicle management operations

- **useDriverStore.js**: Driver management operations

🔧 Key Features

🔐 Authentication & Security

- **JWT-based Authentication**: Secure token-based authentication
- **HTTP-only Cookies**: XSS protection with secure cookie settings
- **Password Hashing**: BCrypt encryption for password security
- **Role-based Access**: Employee and Admin role separation
- **Domain Restriction**: Only @adityabirla.com emails allowed
- **Environment-based Configuration**: Secure credential management through environment variables

📧 Notification System

- **Email Notifications**: Nodemailer integration with configurable admin credentials
- **SMS Notifications**: Twilio integration for mobile alerts
- **Multi-channel Alerts**: Both email and SMS for critical updates
- **Automated Notifications**: Triggered on request approval/rejection
- **Transport Head Alerts**: Dedicated email notifications to transport head for new requests
- **Configurable Recipients**: Environment-based email configuration for flexibility

🤖 Automated Operations

- **Scheduled Trip Completion**: Node-cron job runs every minute
- **Resource Management**: Automatic vehicle/driver status updates
- **Background Processing**: Non-blocking notification delivery
- **Status Synchronization**: Real-time resource availability updates

📊 Reporting & Management

- **Excel Export**: ExcelJS integration for data export
- **Real-time Dashboard**: Live updates every 2-5 seconds
- **Status Tracking**: Comprehensive request lifecycle management
- **Resource Monitoring**: Vehicle and driver availability tracking

🚙 Fleet Management

- **Vehicle Assignment**: Intelligent resource allocation
- **Driver Management**: Availability and assignment tracking
- **Outside Resources**: Support for external vehicles and drivers
- **Capacity Planning**: Passenger capacity management

🛠️ Tech Stack

Backend Dependencies

```
{
  "express": "^5.1.0",
  "mongoose": "^8.16.1",
  "jsonwebtoken": "^9.0.2",
  "bcrypt": "^6.0.0",
  "nodemailer": "^7.0.5",
  "twilio": "^5.8.0",
  "node-cron": "^4.2.0",
  "cors": "^2.8.5",
  "cookie-parser": "^1.4.7",
  "dotenv": "^17.0.1"
}
```

Frontend Dependencies

```
{
  "react": "^19.1.0",
  "react-router-dom": "^7.6.3",
  "zustand": "^5.0.6",
  "axios": "^1.10.0",
  "exceljs": "^4.4.0",
  "lucide-react": "^0.525.0",
  "tailwindcss": "^3.4.17",
  "vite": "^7.0.0"
```

```
}
```

🚀 Getting Started

Prerequisites

- Node.js (v14 or higher)
- MongoDB Atlas account
- Gmail account (for email notifications)
- Twilio account (for SMS notifications)

Environment Configuration

Create `.env` file in the backend directory:

```
MONGODB_URI=your_mongodb_atlas_connection_string
JWT_TOKEN=your_jwt_secret_key
[email protected]
ADMIN_PASS=your_gmail_app_password
[email protected]
PORT=5002
(Optional for SMS notifications)
TWILIO_SID=your_twilio_account_sid
TWILIO_AUTH_TOKEN=your_twilio_auth_token
TWILIO_PHONE_NUMBER=your_twilio_phone_number
```

Installation & Running

Local Development

```
# Install all dependencies
npm run install-all
```

3. Configure environment variables

- Copy the provided `.env.example` file in the `backend` directory to `.env` :

```
cd backend
cp .env.example .env
```

- Edit `.env` to add your MongoDB Atlas URI, JWT secret, and desired PORT:

```
MONGODB_URI=your_mongodb_atlas_connection_string
JWT_SECRET=your_very_secret_key_here
PORT=5002
```

How to generate a MongoDB Atlas URI:

1. Go to MongoDB Atlas and sign up or log in.
2. Create a new project and cluster (free tier is fine).
3. In your cluster, click “Connect” > “Connect your application”.
4. Copy the provided connection string (it looks like `mongodb+srv://<username>:<password>@cluster0.xxxxx.mongodb.net/?retryWrites=true&w=majority`).
5. Replace `<username>`, `<password>`, and any placeholder values with your actual credentials and database name.
6. Paste this string as the value for `MONGODB_URI` in your `.env` file.

JWT_SECRET configuration:

- `JWT_SECRET` can be **any random string**. It is used to sign and verify authentication tokens.
- Example values:
  - `JWT_SECRET=supersecret`
  - `JWT_SECRET=myrandomstring123`
  - `JWT_SECRET=ThisIsAReallyLongAndRandomSecretKey!@#123`
- For production, use a long, unpredictable string.

PORT configuration:

- The default port is `5002`. You can change it in your `.env` file if needed.

4. Start the application

```
# Build and run with Docker Compose
docker-compose up --build

# Access the application
```

```
# Frontend: http://localhost:5173
# Backend API: http://localhost:5002
```

🎛️ User Interfaces

Employee Dashboard Features

- **New Request Creation**: Comprehensive trip request form
- **Active Request Tracking**: Real-time status updates
- **Request History**: Complete request lifecycle view
- **Status Notifications**: Email and SMS alerts
- **Responsive Design**: Mobile-friendly interface

Admin Dashboard Features

- **Request Management**: Approve/reject trip requests
- **Resource Assignment**: Vehicle and driver allocation
- **Fleet Management**: Add/remove vehicles and drivers
- **Status Monitoring**: Real-time resource availability
- **Data Export**: Excel report generation
- **Outside Resource Support**: External vehicle/driver management

---

🔄 System Workflow

Trip Request Lifecycle

1. **Request Creation**: Employee submits trip request
2. **Transport Head Notification**: Dedicated email notification sent to transport head (configurable via TRANSPORT_HEAD_EMAIL)
3. **Request Review**: Admin reviews and assigns resources
4. **Approval/Rejection**: Admin approves or rejects with remarks
5. **User Notification**: Employee receives email/SMS notification with assignment details
6. **Trip Execution**: Vehicle and driver assigned (internal or external resources)
7. **Automatic Completion**: System marks trip as completed after end date
8. **Resource Release**: Vehicle and driver status reset to available

Resource Management

- **Vehicle Assignment**: Automatic status update to ‘Assigned’
- **Driver Assignment**: Automatic status update to ‘assigned’
- **Capacity Validation**: Passenger count vs vehicle capacity
- **Availability Tracking**: Real-time resource status monitoring
- **Conflict Prevention**: Prevents double assignment

🔒 Security Features

Authentication Security

- **JWT Token Expiration**: 10-day token validity
- **Secure Cookie Settings**: HTTP-only, secure, same-site strict
- **Password Requirements**: Minimum 6 characters with bcrypt hashing
- **Session Management**: Automatic token validation

Data Security

- **Input Validation**: Comprehensive request validation
- **SQL Injection Prevention**: Mongoose ODM protection
- **XSS Protection**: HTTP-only cookies and input sanitization
- **CSRF Protection**: Same-site cookie policy

📈 Performance Optimizations

Backend Optimizations

- **Database Indexing**: Optimized queries with proper indexing
- **Connection Pooling**: MongoDB connection management
- **Background Processing**: Non-blocking notification delivery
- **Error Handling**: Comprehensive error management

Frontend Optimizations

- **State Management**: Efficient Zustand state updates
- **Real-time Polling**: Optimized 2-5 second intervals
- **Component Optimization**: React best practices
- **Bundle Optimization**: Vite build optimization

🐛 Troubleshooting

Common Issues

- **MongoDB Connection**: Verify connection string and network access
- **Email Notifications**: Check ADMIN_USER and ADMIN_PASS environment variables
- **Transport Head Notifications**: Verify TRANSPORT_HEAD_EMAIL environment variable
- **Port Conflicts**: Ensure ports 5002 and 5173 are available
- **CORS Issues**: Verify frontend URL in backend CORS configuration
- **Environment Variables**: Ensure all required environment variables are properly set
- **SMS Notifications**: Verify Twilio credentials and phone numbers

Debug Mode

- **Backend Logging**: Console logs for API operations
- **Frontend Console**: Browser developer tools for client-side debugging
- **Network Monitoring**: Check API requests and responses
- **Database Queries**: MongoDB Atlas query monitoring

📄 License

MIT License - See LICENSE file for details

🤝 Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Test thoroughly
5. Submit a pull request

📞 Support

For technical support or questions, please contact the developer or create an issue in the repository.