Courier Management System

Final Report

Prepared by: MD AL-AMIN SARKER

1. Introduction

The Courier Management System (CMS) is a full stack MERN application designed to streamline
parcel booking, assignment, tracking, and delivery for three user roles — Customer, Delivery Agent,
and Admin. The objective of this project is to deliver a minimal yet production ready platform that
demonstrates solid software engineering practices, RESTful API design, authentication, rolenbased
access control, and a modern React frontend.

2. Technology Stack

• Frontend: React 18, Vite, Tailwind CSS, ShadCN/UI
• State management: React Context + useReducer hooks
• Backend: Node.js 20, Express 5, JWT authentication, Mongoose 8
• Database: MongoDB Atlas
• Tooling: ESLint, Prettier, nodemon, dotenv
• Testing & Docs: Postman

3. System Architecture

The application follows a classic client–server model. The React SPA (client) communicates with the
Express API (server) over HTTPS. Key layers: • **Presentation** – React components, Tailwind utility
classes, protected routes handled by ReactnRouter. • **Business Logic** – Controllers coordinate
request validation, service calls, and responses. • **Persistence** – Mongoose models abstract
MongoDB collections. • **Authentication** – JWT access tokens (15 min) + refresh tokens (7 days)
stored as HTTP only cookies, with role claims (customer, agent, admin). • **Realtime (optional)** –
Socket.io channel not wired yet but stubbed for delivery status pushes.

4. REST API Overview

The list below summarises primary endpoints. Detailed request/response samples are included in the

accompanying Postman collection.
• PATCH /api/parcels/:trackingId/assign-agent
• POST /api/parcels/bulk-assign
• GET /api/parcels/unassigned
• GET /api/users
• GET /api/users/:userId
• PATCH /api/users/:userId/role
• GET /api/agents
• GET /api/stats
• GET /api/export/users
• GET /api/dashboard
• GET /api/daily/:date
• GET /api/agent-performance
• GET /api/reports
• GET /api/revenue
• GET /api/delivery
• GET /api/agent/dashboard
• GET /api/agent/daily/:date
• GET /api/agent/performance
• POST /api/pickup
• PATCH /api/:trackingId/status
• GET /api/track/:trackingId
• GET /api/
• GET /api/:trackingId/history
• PATCH /api/:trackingId/location
• PATCH /api/:trackingId/cancel
• GET /api/user/:userId?
• GET /api/track/:trackingId/history
• POST /api/register
• POST /api/login
• GET /api/me
• POST /api/logout
• POST /api/refresh
See Appendix A for the full endpoint catalogue.

5. Database Schema

The MongoDB schema is kept deliberately concise: • **User** — firstName, lastName, email,
passwordHash, role {customer|agent|admin}, phone, address. • **Parcel** — trackingId,

customer (ref User), agent (ref User|null), pickupAddress, deliveryAddress, weight, fee, status
(enum), timeline (subdoc). • **BookingHistory** — parcel (ref Parcel), status, changedBy (ref User),
timestamp. • **Analytics** — date, dailyRevenue, deliveredCount, failedCount, agentPerformance (arrays). Mongoose automatically creates the necessary indexes for relations and
uniqueness (e.g., trackingId).

6. Frontend WalkThrough

The SPA is structured by feature folders (Dashboard, Parcels, Users). A `ProtectedRoute` HOC
guards private screens. Key screens: 1. **Customer** – New Booking, My Parcels, Profile. 2.
Agent – Assigned Parcels, Update Status, Earnings. 3. **Admin** – User Management, Parcel
Assignment, Analytics Dashboard. Tailwind CSS provides the design system; ShadCN components
accelerate form and table scaffolding.

7. Setup & Deployment

Local development
npm i # root install workspaces
cd server && npm run dev
cd client && npm run dev
Create a `.env` from `env.example` and supply:
MONGODB_URI=
JWT_SECRET=
PORT=5000
Production
The server is ready for Docker:
docker build -t cms-api ./server
docker run -d -p 80:5000 --env-file server/.env cms-api
The client builds to static assets with `npm run build` and can be served by Nginx or Vercel.

8. Testing & Documentation

An exported Postman collection
(`Courier_Management_System_API.postman_collection.json`)
covers authentication, parcel, and admin flows. Run `npm test` in `/server` to execute unit tests
silhouettes (Jest). Swaggernstyle JSDoc comments have been started for future OpenAPI
generation.

9. Conclusion & Future Work

Although the MERN ecosystem was completely new terrain for me—my background is in Flutter—I embraced the challenge head-on. In just three or four days I immersed myself in web development fundamentals, shifting from familiar Firebase-powered apps to crafting a custom Node/Express backend. I won't pretend I did it alone: I leaned on guidance from experienced MERN-stack friends (including my brother for the server layer). Their insights, combined with my determination, accelerated my learning curve and gave me a clear roadmap. This sprint has proven that I can quickly adapt, absorb new technologies, and deliver results. With more time, I'm confident I'll master this stack and contribute at an expert level.

The CMS achieves its minimum viable objectives within the given timeframe, demonstrating secure
role based operations and clean separation of concerns. Future enhancements may include: •
Google Map Integration • Email/SMS notifications via Twilio. • CI/CD pipeline
(GitHub Actions) and unit/integration test coverage expansion.