**DESIGN DOCUMENT**

**J&T EXPRESS PARCEL TRACKING SYSTEM WITH GPS MOBILE ACCESS**

A Software Engineering Project Presented to the Faculty of

Datamex College of Saint Adeline in Valenzuela Branch

MR. GABRIEL THOMAS TORNEROS

**By:**ANGELES, ADRIAN R.

BSIT-3B

# INTRODUCTION

This chapter will explain the purpose, overview of the software system,

and scope of the design document

## Main Purpose

To provide a comprehensive blueprint for the design, development, deployment, and maintenance of the system. It serves as a central reference for developers, designers, system administrators, and stakeholders, ensuring that all aspects of the system are clearly defined, understood, and implemented consistently.

## Overview of the software system

A parcel tracking system to have mobile-supported and desktop responsiveness on every website page including the customer and courier website, login for admin panel website, and literal admin panel website. The colors are primary based on the client choice of J&T Express which is Red and White which is being used to this web-based system. As it shows every time the pages have started, the logo and the spinning wheel should appear then fade away for exit transition. After loading the screen, it will show the content of specific website page and often displays a top header except the actual admin panel which uses left sidebar which represents all buttons to change panel. The customer and courier website principles should be applied about user friendliness and accuracy to see their parcels status online regardless of which devices they are using, and the browser should not be outdated.

## Scope of the design document

The purpose of this design document is to define the boundaries and objectives of the system while specifying the tools and technologies that will be used to achieve a modern and aesthetically pleasing website. To implement an interactive and visually appealing design, the system will utilize two key libraries:

* Leaflet.js – A lightweight, open-source JavaScript library that allows developers to create interactive, mobile-friendly maps for web applications. Its core strength lies in its simplicity, speed, and extensibility, making it ideal for projects that require mapping features without the overhead of heavier frameworks
* chart.js – A widely used open-source JavaScript library for creating interactive and visually appealing charts on web pages. It provides a simple API for rendering lines, bar, radar, pie, doughnut, polar area, bubble, and scatter charts, along with support for animations, tooltips, and responsive resizing

## Software Tools

To ensure the Parcel Tracking System is efficient, secure, and visually appealing, a selection of modern software tools and frameworks will be used. These tools support various aspects of development, including frontend design, backend management, deployment, and security, allowing the system to deliver a robust and user-friendly experience. The software tools we will be using:

* Netlify – platform designed for running modern composable web applications. It supports developers and enterprises in building, deploying, and managing web applications efficiently, from corporate sites to complex e-commerce solutions
* reCAPTCHA – A security service offered by Google that helps protect websites and web applications from spam, abuse, and fraudulent activities. It works by distinguishing between human users and automated bots, preventing malicious actions such as fake registrations, automated form submissions, and brute-force attacks
* Supabase Database – An open-source backend-as-a-service (BaaS) platform designed to simplify the development of web and mobile applications. It provides developers with a fully managed backend infrastructure, enabling them to focus on building applications without worrying about server management or scaling complexities

Supabase is built on PostgreSQL, a powerful and reliable database, and offers a suite of integrated tools and features to streamline application development

* Tailwind CSS – A utility-first CSS framework that simplifies web design by providing pre-defined utility classes. It allows developers to style elements directly in HTML, enabling rapid and responsive UI development

# SYSTEM ARCHITECTURE

This chapter explains about the overview of the system architecture like high-level components and their interactions, deployment architecture, and communication protocols and interfaces

The J&T Express Parcel Tracking System with GPS Mobile Access is designed to provide a seamless, responsive, and interactive web experience across desktop and mobile devices. The system architecture defines the main components, their interactions, deployment setup, and communication protocols, ensuring a robust and scalable solution

## High-Level Components and Their Interactions

The system consists of the following key components:

1. Front-End (User Interface)
   * Provides responsive web interfaces for customers, couriers, and administrators.
   * Built with Tailwind CSS for a modern, mobile-friendly design
   * Integrate Leaflet.js to display parcel locations on interactive maps and Chart.js to visualize parcel statistics, trends, and delivery performance
   * Sends requests to the back end using HTTP/HTTPS and consumes RESTful APIs
2. Back-End (Server)
   * Hosted on Supabase, providing managed server infrastructure.
   * Handles authentication, session management, and business logic for tracking, updating, and reporting parcels
   * Integrates reCAPTCHA to prevent automated abuse.
   * Processes requests from front-end clients and interacts with the database for CRUD operations
3. Database
   * Stores all system data, including parcels, parcel updates, customers, couriers, admins, and branches
   * Includes triggers and timestamp functions for automatic update tracking.
   * Enforces Row-Level Security to protect sensitive data and control access based on roles
4. Third-Party Services
   * Netflify: Hosts the front-end of the Parcel Tracking System for fast, reliable, and globally accessible delivery. Supports deployment by uploading HTML, CSS, and JavaScript files or linking to a Git repository for automatic continuous deployment. Provides a global CDN, free SSL certificates, custom domain management, and optional serverless functions for added functionality.

## Deployment Architecture

The system uses client-server architecture with two main deployment options:

1. Local Deployment (Development & Testing)

* The front-end files (HTML, CSS, and JavaScript) can be run directly on a personal computer using a web browser
* Visual Studio Code (VS Code) with the Live Server extension can be used to serve the front-end files locally. This allows developers to preview changes in real-time whenever files are updated, without manually refreshing the browser
* Suitable for development and testing, allowing developers to preview changes instantly without uploading them to a server
* The back end can also be run locally if connected to a local PostgreSQL database (Supabase provides a local development option)
* Limitation: Users outside the local network cannot access the system. Performance depends on the developer’s computer

1. Cloud Deployment (Production)

* The front-end is hosted on Netlify, providing fast, reliable, and globally accessible delivery
* The system connects to a centralized Supabase backend, which handles:
  + Database management for parcels, parcel updates, customers, couriers, admins, and branches
  + Authentication including email/password login and session management for customers, couriers, and admins
  + Storage as buckets to store parcel images, signatures, and other media securely.
  + Real-time updates, enabling live features such as updating parcel status or a chat/message box without page reloads
* This deployment allows users to access the system from any device or location, ensuring scalability, security, and real-time interactivity
* Netlify supports automatic deployment from Git repositories, global CDN distribution, SSL certificates, and optional serverless functions for added functionality

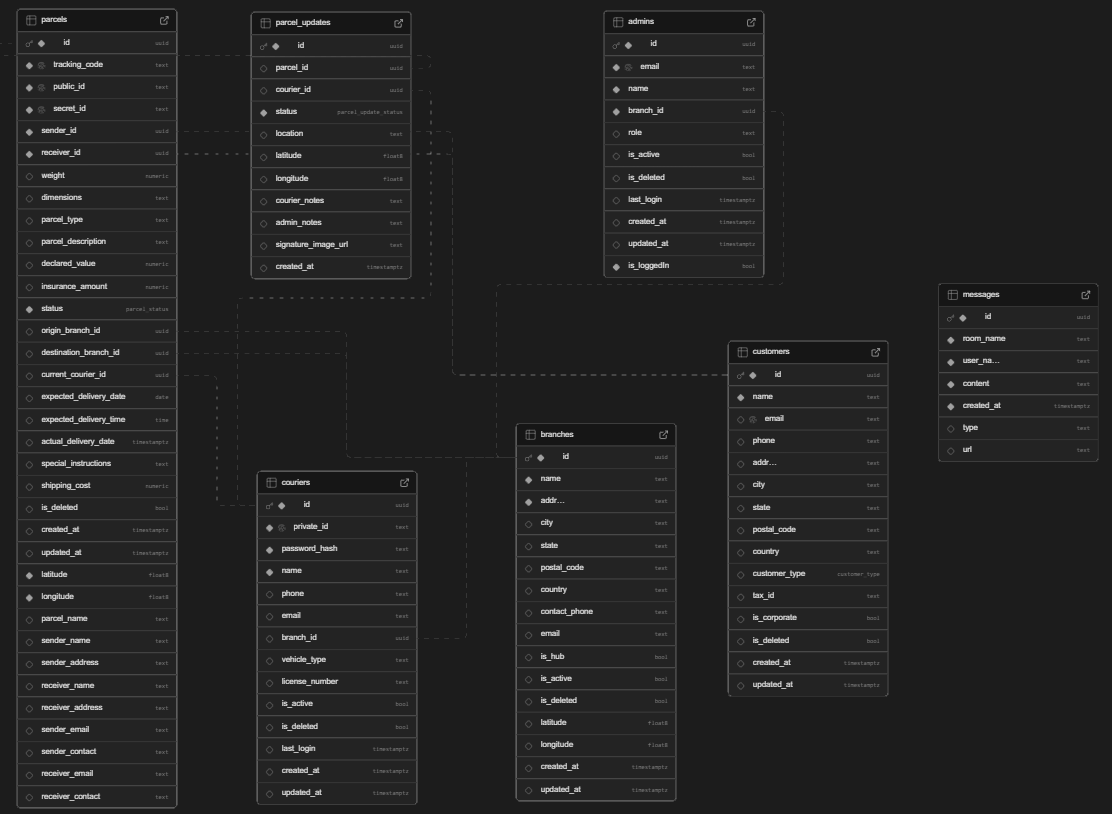
## Communitcation protocols and inferfaces

It defines how different components of the system exchange data and interact with each other. Protocols set the rules for transmitting information, while interfaces define the inputs, outputs, and expected behaviors for each interaction.

* HTTP/HTTPS – The primary protocol used by the front-end (web browser) to communicate with the back end (Supabase). HTTPS ensures that all data is encrypted and secure during transmission
* WebSockets – Supabase Enables real-time communication. When a parcel’s status is updated or a chat message is sent, the back end can push updates instantly to the front-end without requiring the user to refresh the page
* RESTful APIs – The front-end interacts with the back end through specific endpoints (Ex.: fetching parcel status, creating updates, or logging in). Each API endpoint defines the required inputs and the returned outputs, standardizing communication between components
* Database Queries (PostgreSQL) – The back end uses SQL queries to retrieve, insert, update, or delete data in the Supabase PostgreSQL database. These queries serve as an interface between the application logic and the data layer
* Supabase Storage API – Provides a structured interface for managing media files, including uploading, downloading, and deleting parcel images, signatures, and other media stored in buckets
* Supabase Authentication API – Handles user authentication, session management, and role-based access control, ensuring that only authorized users can access specific system features

# DATABASE DESIGN

This part will illustrate Entity-relationship diagram, explaining about database tables, fields, and relationships and Data normalization techniques used



*Figure 1: Entity-relationship diagram of database schema in Supabase*

The system’s database is structured to support efficient parcel tracking, customer management, and courier operations. The ERD would show the following key entities and relationships:

* Branches: Stores branch information; related to admins, couriers, and parcels’ origin/destination
* Customers: Stores customer details; each parcel has a sender and receiver reference to customers
* Admins: Managers managing branches; linked to branches via branch\_id
* Couriers: Delivery personnel; linked to branches and assigned to parcels
* Parcels: Central entity representing shipments; linked to customers, branches, and couriers
* Parcel Updates: Logs status changes of parcels; linked to parcels and couriers

## Database Tables, Fields, and Relationships

1. Branches

* Fields: id, name, address, city, state, postal\_code, country, contact\_phone, email, is\_hub, is\_active, latitude, longitude, created\_at, updated\_at.
* Primary key: id
* Relationships: referenced by admins, couriers, and parcels

1. Customers

* Fields: id, name, email, phone, address, customer\_type, tax\_id, is\_corporate, is\_deleted, created\_at, updated\_at.
* Primary key: id
* Relationships: linked as sender\_id and receiver\_id in parcels

1. Admins

* Fields: id, email, password\_hash, name, branch\_id, role, is\_active, is\_loggedIn, last\_login, created\_at, updated\_at.
* Primary key: id
* Relationships: belongs to one branch (branch\_id)

1. Couriers

* Fields: id, private\_id, password\_hash, name, phone, branch\_id, vehicle\_type, license\_number, is\_active, last\_login, created\_at, updated\_at.
* Primary key: id
* Relationships: belongs to a branch, handles multiple parcels, and logs parcel\_updates.

1. Parcels

* Fields: id, tracking\_code, public\_id, secret\_id, sender\_id, receiver\_id, sender\_name, receiver\_contact, parcel\_name, latitude, longitude, weight, dimensions, parcel\_type, status, origin\_branch\_id, destination\_branch\_id, current\_courier\_id, expected\_delivery\_date, actual\_delivery\_date, special\_instructions, shipping\_cost, is\_deleted, created\_at, updated\_at.
* Primary key: id
* Relationships: references customers (sender and receiver), branches (origin and destination), and couriers.

1. Parcel Updates

* Fields: id, parcel\_id, courier\_id, status, location, latitude, longitude, courier\_notes, admin\_notes, signature\_image\_url, created\_at.
* Primary key: id

## Data Normalization

A process in database design that organizes data to reduce redundancy and improve data integrity. It ensures that the database is efficient, consistent, and easier to maintain. We have three laws of Normal Forms:

1NF (First Normal Form)

* Rule: Each column must contain atomic (indivisible) values, and there should be no repeating groups or arrays in a single column
* Schema Example
  + customers table - name, email, phone, address are all single, atomic values
  + parcels table - parcel\_name, weight, dimensions, etc., are stored in individual columns, not as lists
* Benefit: Makes querying and filtering simpler. For example, you can easily select parcels by weight or city without needing to parse combined fields

2NF (Second Normal Form)

* Rule: The table must be in 1NF, and all non-key attributes must depend fully on the primary key, not just part of it
* Schema Example
  + parcels table has id as the primary key
  + Attributes like tracking\_code, status, sender\_id, receiver\_id, expected\_delivery\_date all depend entirely on the parcel id, not on a subset of a composite key (you don’t have a composite key here, so it naturally satisfies 2NF).
  + admins table - role, branch\_id, is\_active all fully depend on id

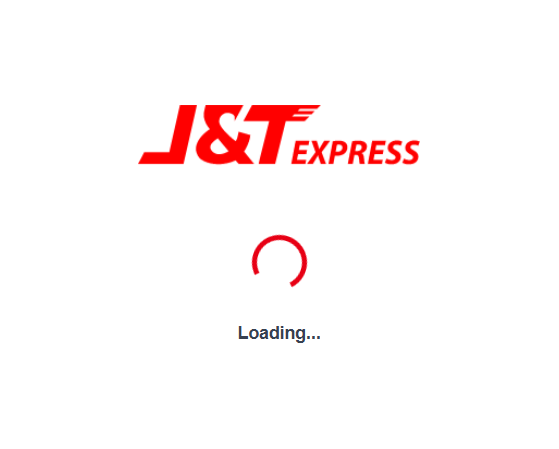
3NF (Third Normal Form)

* Rule: The table must be in 2NF, and there should be no transitive dependency non-key attributes cannot depend on other non-key attributes
* Database Schema Example
  + parcels table: Instead of storing sender/receiver addresses directly in the parcel table, you store them in customers and reference them via sender\_id and receiver\_id
  + This prevents the transitive dependency: parcel → sender\_id → sender\_address
* Benefit: Avoids data duplication and ensures that updating a customer’s address automatically updates all related parcels

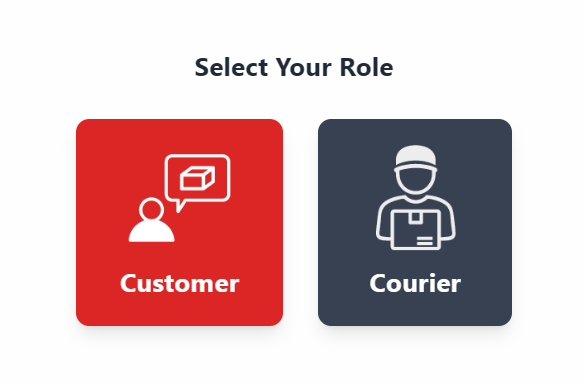
Enums (Data Consistency)

* Rule: Enums are predefined sets of values that a column can take. They help maintain data integrity
* Schema Example
  + parcel\_status ENUM: 'Pending', 'In Transit', 'Out for Delivery', 'Delivered', 'Returned', 'Cancelled'. Only these statuses are allowed in the parcels.status field
  + parcel\_update\_status ENUM: 'Picked Up', 'In Transit', 'Sorting', 'Out for Delivery', 'Delivered', 'Failed Delivery', 'Returned to Sender'. This ensures parcel updates are consistent
  + customer\_type ENUM: 'Individual', 'Business', 'Corporate', 'Government'. Only valid customer types are stored in customers.customer\_type
* Benefit: Prevents errors like typos or inconsistent values and simplify querying (Ex.: filtering all ‘Delivered’ parcels)

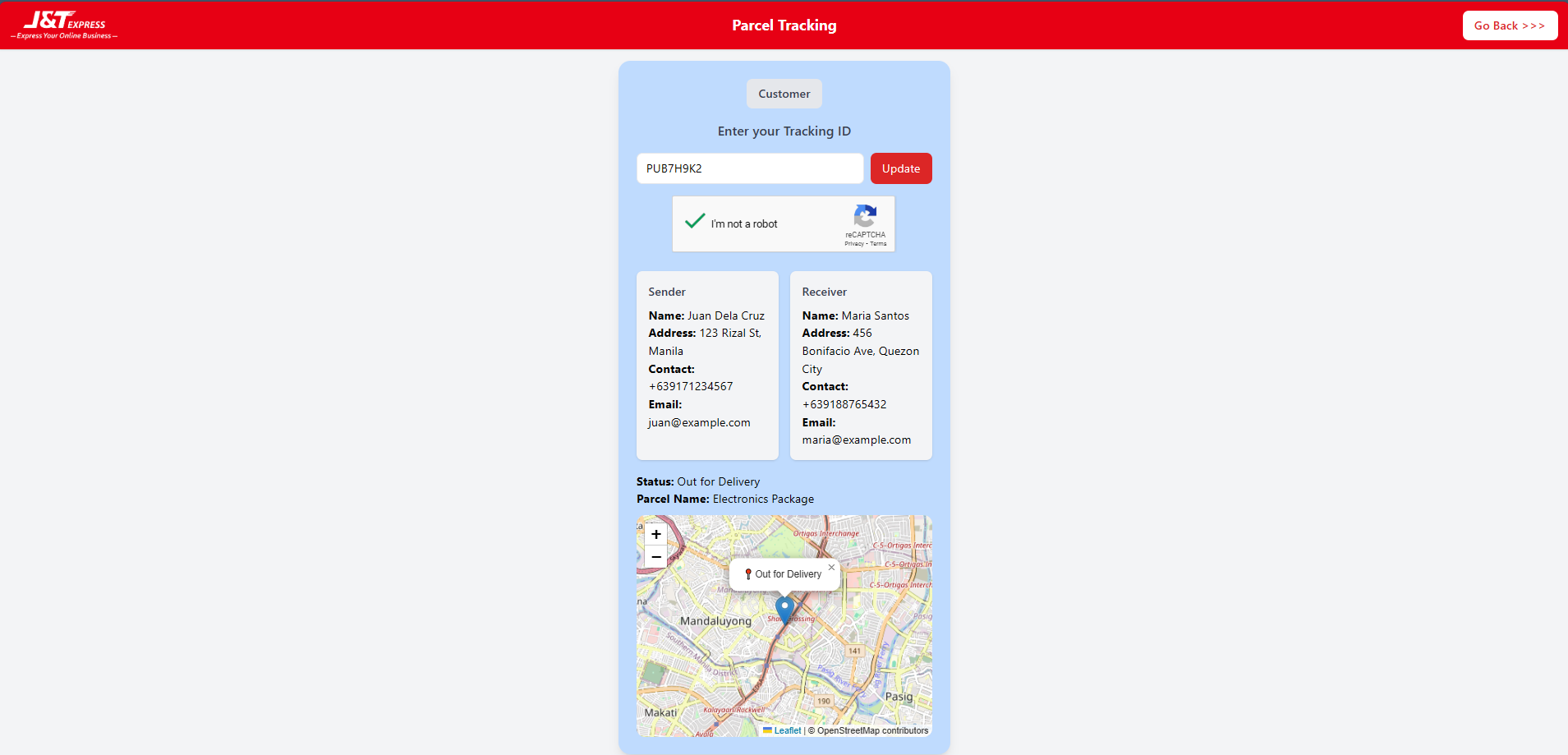
# USER INTERFACE DESIGN

****

*Figure 2: Loading Layout wireframe of the web-based application*

****

*Figure 3: Choose Role Layout wireframe of the web-based application*

****

*Figure 4: Customer Tracking Page Wireframe of the Web-based Application*

**A screenshot of a login form

AI-generated content may be incorrect.**

*Figure 5: Login Form for Admin Panel Page of Wireframe*

**A screen shot of a graph

AI-generated content may be incorrect.**

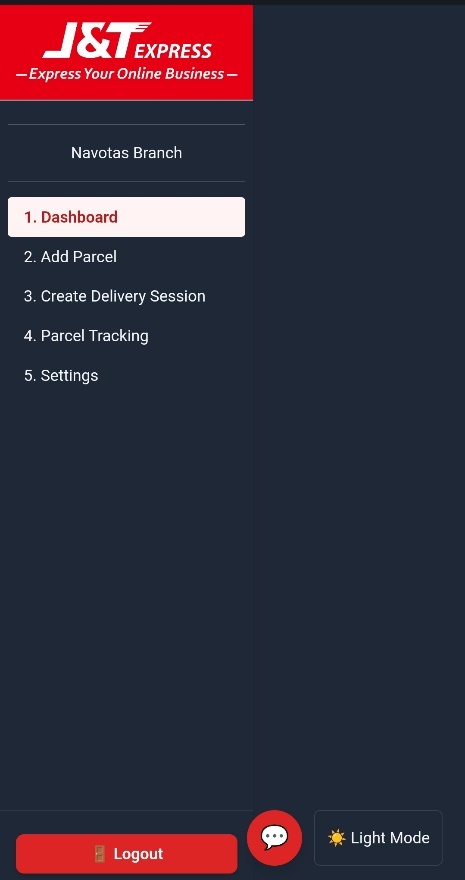
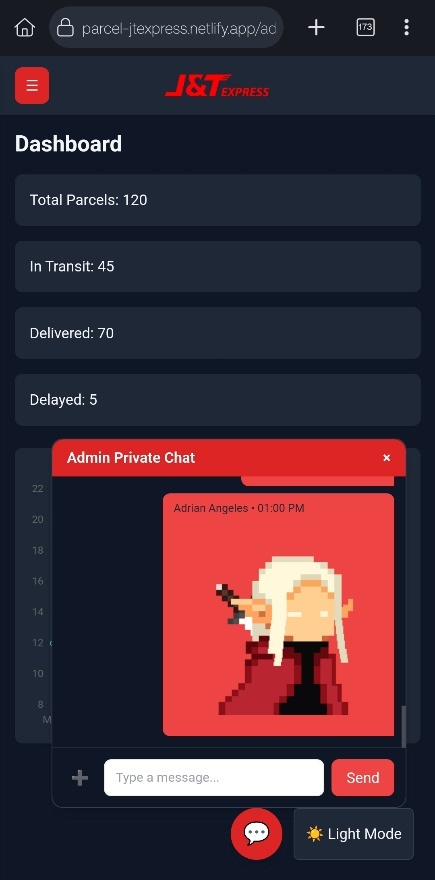
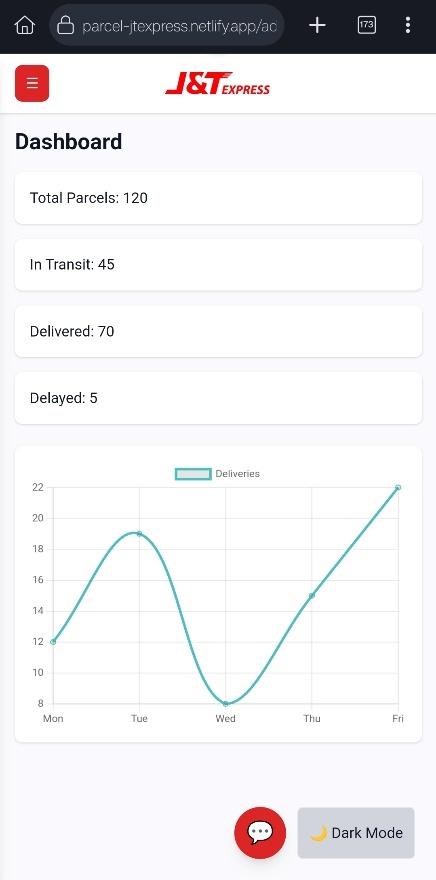
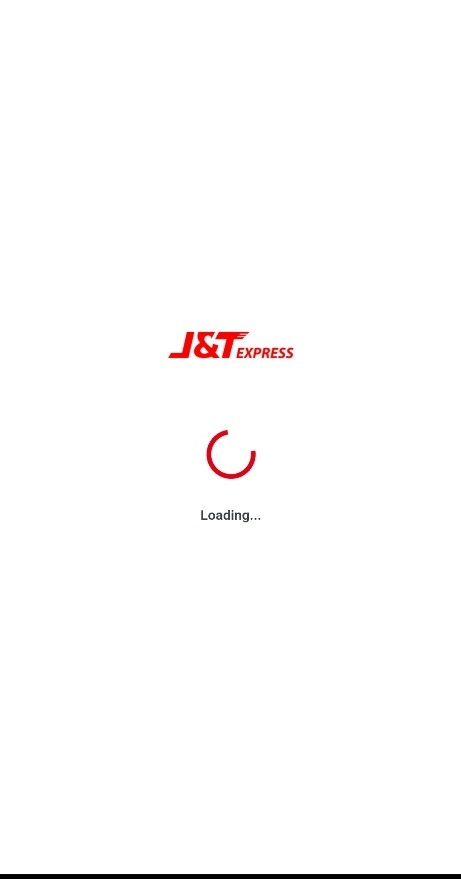
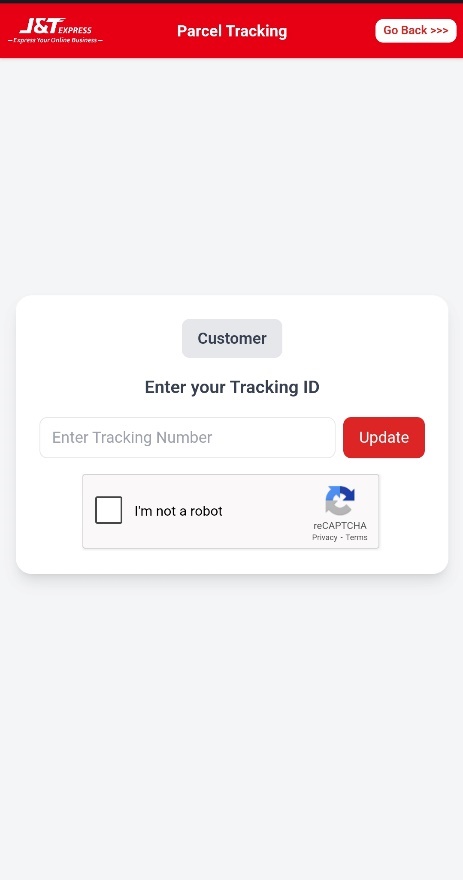
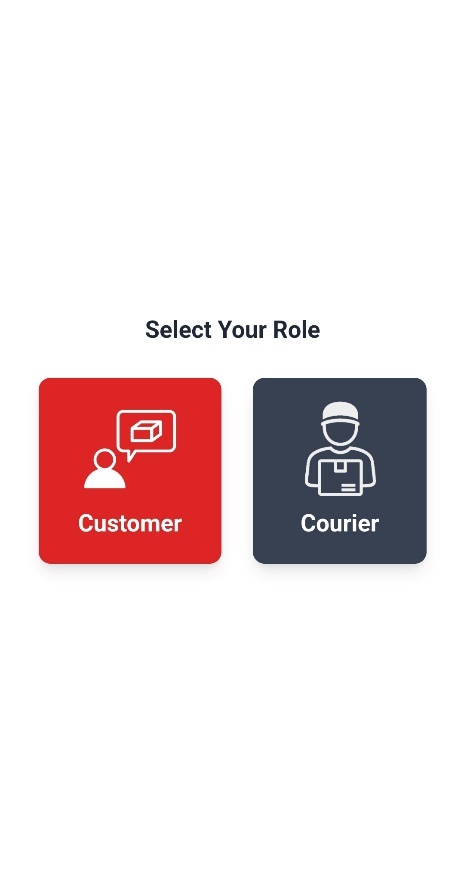
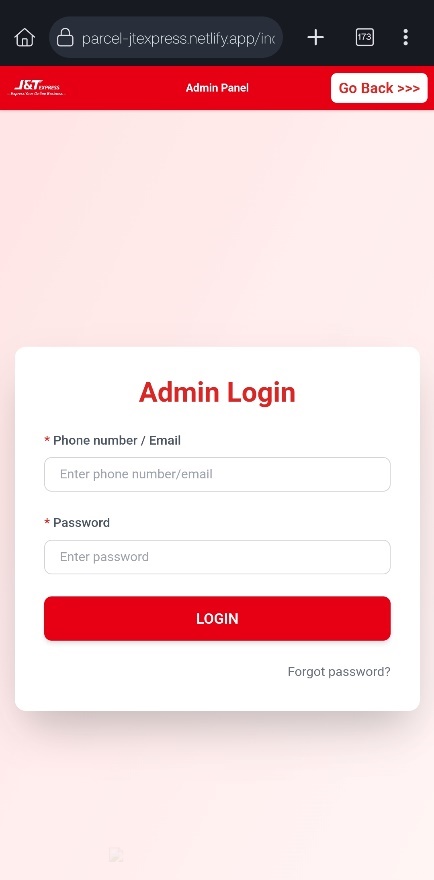
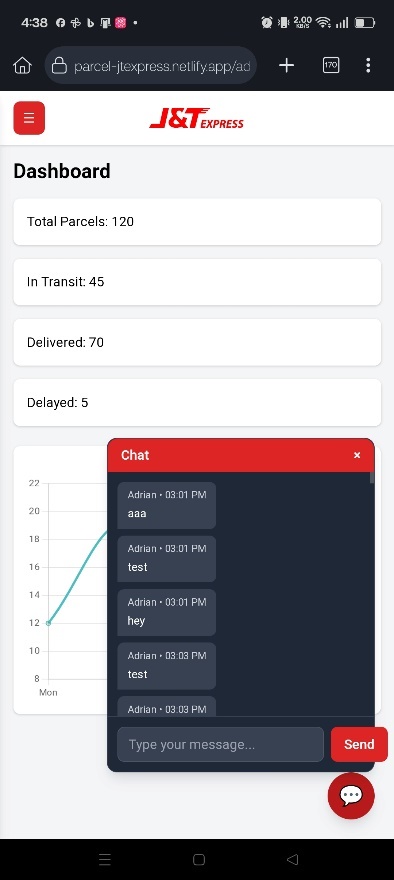
*Figure 6: Main Dashboard for Admin Panel Page of Wireframe*

**A screen shot of a computer

AI-generated content may be incorrect.**

*Figure 7: Main Dashboard for Admin Panel Page of Wireframe*

**Mobile User Interface Design**

****

*Figure 8-15: Mobile User Interface Wireframe of Web-based Application*

# COMPONENT DESIGN

This chapter will tackle key system components/modules, Interface specifications, dependency management and interaction between components

## Key System Components / Modules

1. Authentication Module
   * Handles user login, logout, and session validation
   * Ensures only authorized users (admins/customers) can access restricted features
   * Maintains authentication tokens using cookies
2. Admin Management Module
   * Allows administrators to manage parcels, customers, and system records
   * Updates admin login status in the database (is\_loggedIn, last\_login)
3. Parcel Tracking Module
   * Enables searching and retrieving parcel details using tracking numbers
   * Displays parcel status updates (Ex.: “In Transit”, “Delivered”)
4. Customer Management Module
   * Stores customer details (name, address, contact number)
   * Links customer records with parcels for proper delivery handling
5. Database Module (Supabase)
   * Provides cloud-hosted storage for all system data (parcels, customers, admins).
   * Ensure consistency and data security
   * Implements authentication and role-based access
6. User Interface Module
   * Provides the front-end design for login, parcel tracking, and admin dashboard.
   * Implements responsive design principles for usability

## Interface Specifications

1. Authentication Module

* Functions:
  + async function login() → Authenticates admin via Supabase, stores session tokens in cookies, and redirects to dashboard
  + async function checkLogin() → Validates access token, redirects between login and dashboard accordingly
  + async function logout() → Clears cookies, updates admin’s is\_loggedIn flag in DB, and redirects to login page
  + async function getValidAccessToken() → Refreshes token if expired using refresh\_token
  + async function hashPassword(password) → Hashes password using SHA-256 before storing in cookies
  + async function verifyPassword(inputPassword) → Verifies login attempt against stored password hash
* API Calls (Supabase):
  + supabaseClient.auth.signInWithPassword({ email, password }) → Authenticates user
  + supabaseClient.auth.getUser(token) → Retrieves authenticated user details
  + supabaseClient.auth.setSession({ access\_token, refresh\_token }) → Stores/refreshes user session

1. Admin Management Module

* Functions:
  + async function setLoggedIn(email, value) → Updates is\_loggedIn and last\_login in the admins table
  + async function checkAdminLoggedIn() → Ensures an active session; logs out if is\_loggedIn is false
  + async function isAdmin(email) → Verifies if the user belongs to the admins table
  + async function getUserRole() → Retrieves admin role from DB for role-based access control
* API Calls (Supabase):
  + supabaseClient.from("admins").select("is\_loggedIn").eq("email", email) → Check login state
  + supabaseClient.from("admins").update({ is\_loggedIn: true/false }).eq("email", email) → Update admin status
  + supabaseClient.from("admins").select("role").eq("email", email) → Fetch user role

1. Parcel Tracking Module

* Functions:
  + async function trackParcel() → Fetches parcel details by public tracking ID, verifies CAPTCHA, and displays results on map
  + function showMessage(msg, type) → Displays styled status/error messages dynamically
* API Calls (Supabase):
  + supabaseClient.from("parcels").select("\*").eq("public\_id", id).maybeSingle() → Retrieves parcel record

1. Customer Management Module

* Functions:
  + async function getCustomerById(customerId) → Retrieves customer details linked to a parcel
* API Calls (Supabase):
  + supabaseClient.from("customers").select("\*").eq("id", customerId) → Fetch customer data

1. Database Module

* Tables:
  + admins(admin\_id, email, password\_hash, is\_loggedIn, last\_login, role)
  + parcels(parcel\_id, public\_id, parcel\_name, status, customer\_id, sender\_name, sender\_address, sender\_contact, receiver\_name, receiver\_address, receiver\_contact, latitude, longitude, updated\_at)
  + customers(customer\_id, name, address, contact\_number, email)

1. User Interface Module

* Functions / Behaviors:
  + Displays login form, tracking form, and admin dashboard
  + Uses reCAPTCHA to prevent spam in tracking
  + Uses Leaflet.js for map display and parcel location updates

## Dependency management and interaction between components

Authentication Module → Admin Management Module

* Login must succeed before admin management functions can be accessed

Authentication Module → Parcel Tracking Module

* Customers can only view their parcels if authenticated

Parcel Tracking Module → Database Module

* Parcel data is retrieved from the parcels table

Admin Management Module → Database Module

* Updates admin status in the admins table

Customer Management Module → Database Module

* Fetches customer information and links it with parcel data

User Interface Module → All Modules

* UI serves as the access point, invoking functions from Authentication, Tracking, and Admin modules

# DATA FLOW DIAGRAMS

This chapter will illustrate Data Flow Diagrams and identifications of data sources, processing logic, and destinations.A diagram of a diagram

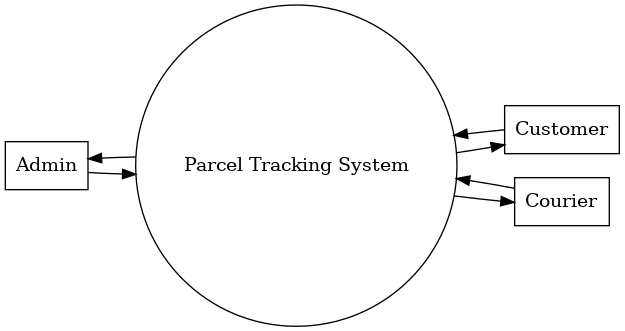
AI-generated content may be incorrect.

*Figure 16: Data Flow Diagram of Level 2*

A diagram of a system

AI-generated content may be incorrect.

*Figure 17: Data Flow Diagram of Level 1*



*Figure 18: Data Flow Diagram of Level 0*

1. Data Sources

* Admins (via Supabase Auth): Provide login credentials, parcel creation requests, and update instructions
* Couriers: Provide parcel delivery status updates (Ex.: in-transit, delivered)
* Customers: Provide parcel tracking requests using parcel ID or tracking number
* External Accounts (J&T Express Website): Customers and couriers are assumed to already have accounts created outside the system

1. Processing Logic

* Authentication Module (Supabase Auth): Validates admin credentials and manages sessions
* Parcel Management Module: Handles parcel creation (by admins), status updates (by couriers/admins), and branch transfers
* Tracking Module: Retrieves parcel information and updates for customers
* Database Triggers/Functions: Optionally auto-sync parcel\_updates.status → parcels.status when a new update is recorded

1. Destinations

* Parcel Database (parcels): Stores core parcel records tied to branch IDs
* Parcel Update Database (parcel\_updates): Stores detailed event history (status, timestamps, branch transfers)
* Customer Interface (Tracking Page): Displays parcel status and history
* Admin Dashboard: Displays branch-specific parcels, customer data, and courier assignments
* Courier Dashboard: Displays assigned parcels and allows updating statuses

# SECURITY DESIGN

This Chapter will include overview of security requirements and consideration, authentication and authorization, and data encryption and and protection measures

## Overview of Security Requirements and Considerations

The system must safeguard sensitive information about parcels, customers, and branch operations by enforcing strict access controls. Security requirements include:

* Ensuring only authorized administrators can log in
* Preventing unauthorized access to data across different branches
* Enforcing single active sessions to reduce account sharing or hijacking risks
* Protecting communication channels and stored credentials from interception or exposure

## Authentication and Authorization Mechanisms

* Authentication
  + Handled via Supabase Auth with email and password
  + Admin credentials are securely verified against the Supabase authentication system
  + Only administrators authenticate; couriers and customers are not part of Supabase Auth
* Authorization
  + Each admin is tied to a specific branch\_id
  + Admins can only view and manage parcels belonging to their assigned branch
  + Higher-level roles (Ex.: system owners) may have global access across all branches
  + Session control is enforced through an is\_loggedIn flag, ensuring only one active session per admin account

## Data Encryption and Protection Measures

* Password Hashing
  + Passwords are never stored in plain text. They are hashed using SHA-256 before being stored in cookies or validated
* Transport Layer Security
  + All client-server communication occurs over HTTPS/TLS to prevent eavesdropping and man-in-the-middle attacks
* Token Security
  + Supabase issues secure access (sb-access-token) and refresh tokens (sb-refresh-token)
  + Tokens are stored in cookies with security attributes (Secure, SameSite=Strict) to protect against cross-site attacks
* Session Management
  + Expired or invalid tokens trigger automatic logout and redirection to the login page
  + Active sessions are refreshed securely using Supabase’s built-in session handling

# PERFORMANCE DESIGN

This chapter will explain Performance Requirements and Objectives, Strategies for optimization, and Performance testing plan

## Performance Requirements and Objectives

The system is required to maintain high responsiveness and reliability to ensure efficient parcel tracking and management. The main objectives are:

* Low Latency: Parcel lookups, updates, and admin dashboard loads should respond within 2–3 seconds under normal load
* High Availability: The system should operate with at least 99.9% uptime during branch operating hours
* Scalability: The platform must handle increasing parcel records, updates, and admin accounts as the number of branches and customers grows
* Efficient Data Handling: The system should optimize database queries and prevent performance degradation as parcel histories become larger

## Strategies for Optimizing System Performance

* Database Optimization
  + Use indexing on critical fields such as parcel\_id, branch\_id, and status
  + Normalize tables (1NF–3NF) while selectively denormalizing where quick lookups are frequently required
  + Apply constraints and ENUMs to enforce consistency and reduce query complexity
* Caching
  + Implement client-side caching for frequently accessed data like parcel status or branch info
  + Use Supabase edge caching to minimize repeated API calls
* Load Balancing
  + Serve web assets (HTML, CSS, JS) using a Content Delivery Network (CDN)
  + Allow Supabase backend auto-scaling to distribute database queries efficiently
* Code and Query Efficiency
  + Apply pagination and “lazy loading” for large parcel datasets in the dashboard and for message history on chat box
  + Use batch operations for multiple updates rather than numerous single writes
  + Minimize heavy client-side rendering and optimize frontend scripts

## Performance Testing Plan

* Load Testing: Simulate multiple admins accessing parcel records at the same time to measure system response times
* Stress Testing: Test the system with extreme parcel update requests or bulk data imports to evaluate stability under peak loads
* Scalability Testing: Gradually increase database size and concurrent users to ensure the system adapts without degradation
* Monitoring & Logging: Utilize Supabase logs and analytics dashboards to monitor query performance and detect bottlenecks in real time

# ERROR HANDLING AND LOGGING

This explains how we can do about debugging the system with validation

## Error handling mechanisms and strategies

Centralized error handling:

Wrap all Supabase queries (.from().select(), .update(), .auth) in try/catch blocks or use .then(({data, error}) => ...).  
A screen shot of a computer

AI-generated content may be incorrect.

*Figure 19: Query of Internet Checking Connection on Web-based Application System*

Graceful fallback:

* If parcel fetch fails → show a clear error to the user (❌ Parcel not found)
* If token validation fails → force logout to avoid unauthorized access
* If admin login check fails → terminate session to prevent ghost logins

Categorize errors:

* User input errors → Missing tracking ID, invalid email/password
* Authentication errors → Invalid/expired token
* Database errors → Network issues, query failures
* System errors → JavaScript runtime issues, unavailable APIs

## Logging Requirements and Specifications

console.log(): Used to print general information or debugging messages.

console.warn(): Used to highlight a warning (not an error, but something that might cause issues).

console.error(): Used to log critical errors that break functionality or need immediate attention.

## Error Codes and Messages

* **Error Codes** (internal mapping):
  + E001: Invalid Tracking ID.
  + E002: Parcel not found.
  + E003: Authentication failed (wrong credentials).
  + E004: Session expired (invalid token).
  + E005: Database update failed.
  + E006: Unauthorized action (non-admin trying to access dashboard).

## User-Friendly Messages

* + **"⚠ Please enter a Tracking ID."** → (User Input)
  + **"❌ Parcel not found. Please check your Tracking ID."** → (E002)
  + **"❌ Login failed. Invalid email or password."** → (E003)
  + **"⚠ You have been logged out. Please log in again."** → (E004)
  + **"❌ Error fetching parcel. Please check your Internet Connection."** → (E005)

**THIRD-PARTY INTEGRATIONS**

List of Third-Party Services and APIs Integrated into the System

1. Supabase – Provides backend services including authentication, real-time database, and APIs for parcel, courier, and customer management.
2. Google reCAPTCHA – Protects the login page from spam and automated bot access.
3. Netlify – Used as the deployment and hosting platform for the Parcel Tracking System.
4. Leaflet.js – Enables interactive map visualizations for branch locations and parcel routing.
5. Chart.js – Used to generate analytical charts and visual reports of parcel tracking data.
6. Tailwind CSS – Utility-first CSS framework used for styling the web interface efficiently.

Description of Integration Points and Data Exchange Formats

* Supabase – Integrated via JavaScript SDK; communicates through RESTful APIs and WebSockets. Data is exchanged in JSON format for authentication and database queries.
* Google reCAPTCHA – Integrated into the login form via client-side script; verification tokens are sent in JSON format to the backend for validation.
* Netlify – Integrated as the hosting platform, serving frontend assets (HTML, CSS, JS). Supports continuous deployment with Git but won’t be using it. Only drag and drop folder.
* Leaflet.js – Integrated into the admin dashboard; consumes GeoJSON and coordinate data to display parcel routes and branch maps.
* Chart.js – Integrated into the analytics dashboard; receives data in JSON format from Supabase queries to render charts.
* Tailwind CSS – Integrated at the frontend level with HTML templates, no special data exchange required (utility classes applied directly in markup).

# DEPLOYMENT PLAN

This chapter will explain the process of deploying the Parcel Tracking System

into a production environment.

## Overview of the Deployment Process

The Parcel Tracking System will be deployed using Netlify for hosting. The frontend application (HTML, CSS, JavaScript, Tailwind, and third-party libraries like Leaflet.js and Chart.js) is built locally, packaged into a deployment folder (including index.html, CSS, JavaScript, and assets), and then uploaded to Netlify through the drag-and-drop method.

The backend services are managed by Supabase, which provides database, authentication, and APIs. Since Supabase is cloud-hosted, deployment involves setting up tables, authentication policies, and API configurations directly in the Supabase dashboard**.**

## Hardware and Software Requirements for Deployment

* Hardware Requirements
  + Mobile Phone: Used for barcode scanning as an alternative and website testing
  + Barcode Scanner: Used by admins/couriers for scanning parcels during check-in and delivery.
  + Standard Desktop/Laptop: Minimum 8GB RAM, dual-core processor, and stable internet connection for admins using the system.
  + Branch Devices: Computers in each branch office capable of running a modern web browser.
* Software Requirements
  + Frontend Hosting: Netlify (manual drag-and-drop uploads).
  + Backend Hosting: Supabase cloud services for authentication, database, and API handling.
  + Development Tools: Any IDE (Ex.: VS Code) for editing, and Node.js if needed for testing builds.
  + Browser Requirements: Chrome, Edge, or Firefox (latest versions).

## Configuration Management and Version Control Procedures

* File Management: Deployment folders are prepared locally and uploaded to Netlify via the drag-and-drop interface.
* Version Control: Project files are managed and updated locally. New versions are deployed by replacing the existing folder on Netlify with the updated one.
* Environment Management: Supabase environment keys (API URL and secret keys) are configured in Netlify’s environment variable settings to protect sensitive information.
* Change Management: Updates are tested locally before creating a new deployment folder for upload. Each new upload represents a new version of the system.
* Rollback Strategy: Netlify keeps deployment history, allowing the system to roll back to a previous stable version if issues occur.

# MAINTENANCE AND SUPPORT

This chapter will explain the guidelines and procedures for maintaining and supporting the Parcel Tracking System to ensure its reliability, security, and continued efficiency.

Proper maintenance ensures that the system remains up to date with the latest technologies and can handle new business requirements, while support ensures that users have assistance when encountering problems.

## Guidelines for System Maintenance and Support

* Regular Monitoring: The system should be monitored for uptime, performance, and errors using tools such as Netlify analytics and Supabase logs.
* Preventive Maintenance: Conduct scheduled database optimization (e.g., index rebuilding, cleanup of outdated parcel records) to ensure system performance.

## Procedures for Handling Software Updates, Patches, and Bug Fixes

* Software Updates:
  + Frontend updates (HTML, CSS, JavaScript, Tailwind, Chart.js, Leaflet.js) are developed and tested locally, then re-deployed via Netlify using the drag-and-drop method.
  + Backend updates (Supabase database schema, authentication policies, triggers) are performed through the Supabase dashboard, with careful versioning and testing before deployment.
* Security Patches:
  + Apply patches immediately for vulnerabilities in third-party libraries (e.g., Chart.js, Tailwind CSS, Leaflet.js).
  + Regularly rotate Supabase API keys and review access policies.
* Bug Fixes:
  + Minor bugs are logged, fixed locally, tested in a staging environment (if available), and then re-deployed.
  + Major bugs are prioritized and escalated to senior developers for immediate resolution.

## Escalation Process for Resolving Issues

* Level 1 – Initial Support: Admins and couriers encountering issues report them through the support channel. First-level support staff attempt to resolve using known solutions or documentation.
* Level 2 – Technical Support: If unresolved, issues are escalated to the development team for in-depth troubleshooting (Ex.: debugging frontend code, checking Supabase logs).
* Level 3 – Critical Escalation: Critical failures (Ex.: system downtime, data corruption, parcel tracking errors) are escalated to system administrators and project owners for immediate action. This may involve rolling back to a previous Netlify deployment or restoring a Supabase database backup.
* Issue Tracking: All issues are logged, assigned, and tracked until closure to ensure accountability and transparency.

# REVISION HISTORY

|  |  |  |
| --- | --- | --- |
| Version | Date | Description of Changes |
| 1.0 | August 26, 2025 | Initial Draft of this document |

# APPENDIX

## Tools & Framework Documentation

Chart.js. (2025). Chart.js documentation. https://www.chartjs.org/

Google. (2025). reCAPTCHA documentation. https://developers.google.com/recaptcha

Leaflet.js. (2025). Leaflet: An open-source JavaScript library for mobile-friendly interactive maps. https://leafletjs.com/

Supabase. (2025). The open-source Firebase alternative. https://supabase.com/

Tailwind CSS. (2025). Tailwind CSS documentation. https://tailwindcss.com/

## Other References

GeeksforGeeks. (n.d.). What is data encryption? GeeksforGeeks. Retrieved August 26, 2025, from https://www.geeksforgeeks.org/computer-networks/what-is-data-encryption/

GeeksforGeeks. (n.d.). *Introduction to database normalization*. GeeksforGeeks. Retrieved August 26, 2025, from

https://www.geeksforgeeks.org/dbms/introduction-of-database-normalization/GeeksforGeeks  
(Data Encryption)

OWASP Foundation. (n.d.). Session management cheatsheet. OWASP. Retrieved August 26, 2025,

https://owasp.org/www-project-cheat-sheets/cheatsheets/Session\_Management\_Cheat\_Sheet.html

(Single Active Session)

OWASP Foundation. (n.d.). Password storage cheatsheet. OWASP. Retrieved August 26, 2025,

https://owasp.org/www-project-cheat-sheets/cheatsheets/Password\_Storage\_Cheat\_Sheet.html (Hash Password)