**Courier Service System Project Report**

**GROUP MEMBERS:**

1. **MUJTABA HASHMI (2312126)**
2. **M.HASSAN SIDDIQ (2312123)**

**INTRODUCTION:**

This report provides a comprehensive overview of the **Courier Service System** project. Designed to manage various aspects of a courier business, this system facilitates efficient handling of customers, couriers, packages, and shipments from creation to delivery. It leverages a modern three-tier architecture, ensuring scalability and ease of maintenance.

**PROJECT OVERVIEW**

The Courier Service System is built to streamline operations for a courier company. It provides a robust backend API for data management and an intuitive frontend interface for user interaction and data visualization. The system is divided into two main components:

1. **Backend (Node.js with Express.js & PostgreSQL/Neon DB):** Handles data storage, retrieval, and business logic. It exposes RESTful API endpoints for all core entities.
2. **Frontend (HTML, CSS, JavaScript):** Provides a user-friendly web interface to interact with the backend, allowing for data entry, viewing, and insightful reporting.

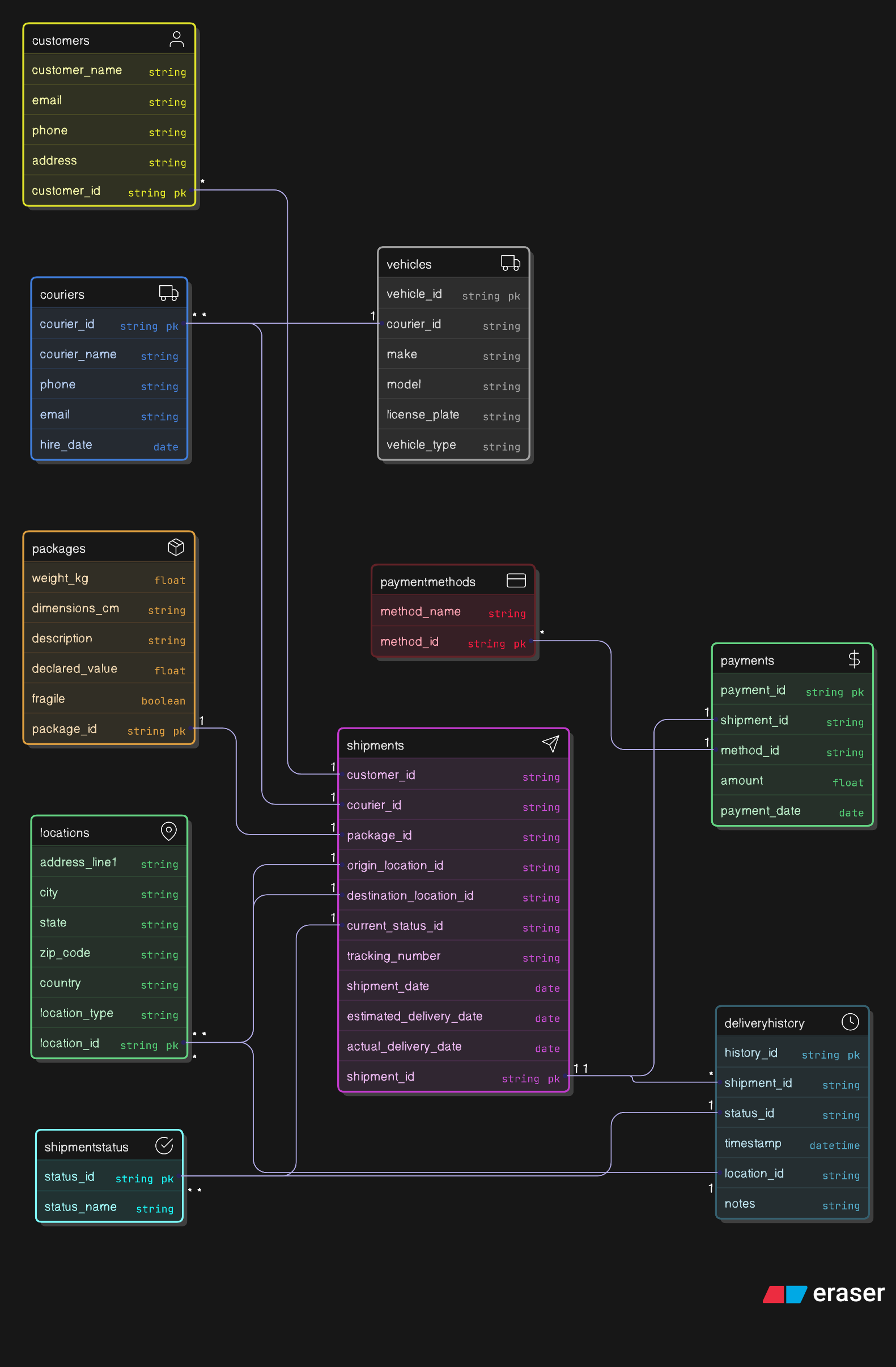
The project adheres to a **black and white theme** for a clean and professional aesthetic.

**ENTITY RELATIONSHIP DIAGRAM (ERD) - CONCEPTUAL DESCRIPTION**

Since a direct ERD image cannot be generated, below is a detailed textual description of the database schema and relationships. The database consists of 10 interconnected tables designed to store all essential courier operations data.

**Entities and Their Relationships:**

1. **Customers**
   * customer\_id (Primary Key)
   * Attributes: customer\_name, email, phone, address
   * Relationship: One-to-Many with Shipments (A customer can initiate multiple shipments).
2. **Couriers**
   * courier\_id (Primary Key)
   * Attributes: courier\_name, phone, email, hire\_date
   * Relationship: One-to-Many with Shipments (A courier can be assigned to multiple shipments).
   * Relationship: One-to-Many with Vehicles (A courier can be assigned multiple vehicles).
3. **Packages**
   * package\_id (Primary Key)
   * Attributes: weight\_kg, dimensions\_cm, description, declared\_value, fragile (boolean)
   * Relationship: One-to-One with Shipments (Each shipment involves exactly one package).
4. **Locations**
   * location\_id (Primary Key)
   * Attributes: address\_line1, city, state, zip\_code, country, location\_type (e.g., 'warehouse', 'customer\_address')
   * Relationship: One-to-Many with Shipments (Both origin\_location\_id and destination\_location\_id in Shipments reference this table).
   * Relationship: One-to-Many with DeliveryHistory (Location where status update occurred).
5. **ShipmentStatus** (Lookup Table)
   * status\_id (Primary Key)
   * Attributes: status\_name (e.g., 'Pending', 'In Transit', 'Delivered')
   * Relationship: One-to-Many with Shipments (Each shipment has one current status).
   * Relationship: One-to-Many with DeliveryHistory (Each history entry has a status).
6. **Shipments**
   * shipment\_id (Primary Key)
   * Attributes: customer\_id (FK), courier\_id (FK), package\_id (FK), origin\_location\_id (FK), destination\_location\_id (FK), current\_status\_id (FK), tracking\_number, shipment\_date, estimated\_delivery\_date, actual\_delivery\_date (nullable)
   * Relationship: One-to-One with Payments (Each shipment can have one payment).
   * Relationship: One-to-Many with DeliveryHistory (A shipment can have multiple history entries).
7. **PaymentMethods** (Lookup Table)
   * method\_id (Primary Key)
   * Attributes: method\_name (e.g., 'Cash on Delivery (COD)', 'Online Payment')
   * Relationship: One-to-Many with Payments (Multiple payments can use the same method).
8. **Payments**
   * payment\_id (Primary Key)
   * Attributes: shipment\_id (FK, Unique), method\_id (FK), amount, payment\_date
   * Relationship: One-to-One with Shipments.
9. **Vehicles**
   * vehicle\_id (Primary Key)
   * Attributes: courier\_id (FK, nullable), make, model, license\_plate, vehicle\_type (e.g., 'Motorcycle', 'Car', 'Truck')
   * Relationship: Many-to-One with Couriers.
10. **DeliveryHistory**
    * history\_id (Primary Key)
    * Attributes: shipment\_id (FK), status\_id (FK), timestamp, location\_id (FK, nullable), notes
    * Relationship: Many-to-One with Shipments.



**Explanation of Components**

**Database (PostgreSQL on Neon.tech)**

The data layer uses **PostgreSQL**, hosted on **Neon.tech**, for robust and scalable data storage.

* **Table Design:** The 10 tables (Customers, Couriers, Packages, Locations, ShipmentStatus, Shipments, PaymentMethods, Payments, Vehicles, DeliveryHistory) are designed with appropriate primary and foreign keys to ensure data integrity and define relationships as described in the ERD section.
* **Data Insertion:** Each table is pre-populated with 20 sample records, tailored with Pakistani context (names, addresses, vehicle types), to facilitate immediate testing and demonstration of the system.

**Backend (Node.js with Express.js)**

The backend is built with **Node.js** and the **Express.js** framework, providing a RESTful API.

* **Database Connection:** It connects to the PostgreSQL database using the pg library.
* **API Endpoints:**
  + **GET (Read) methods:** For each of the 10 tables, there's an endpoint (e.g., /api/customers, /api/shipments) to fetch all records.
  + **POST (Create) methods:** For each table, an endpoint (e.g., /api/customers, /api/shipments) allows adding new records. These methods handle data validation and error reporting (e.g., for foreign key violations, duplicate entries, or missing data).
  + **Report API:** A dedicated endpoint (/api/report/shipment\_summary) is available to fetch aggregated data for the dashboard report. This query joins multiple tables to provide a comprehensive summary of shipments by their status, including total shipments, average package weight, and total revenue.

**Frontend (HTML, CSS, JavaScript)**

The frontend is a web-based application built with **HTML**, styled using **Tailwind CSS** (for a black and white theme), and powered by **JavaScript**.

* **Navigation:** A navigation bar at the top allows users to easily switch between different entity management pages and the dashboard.
* **Table Views:** Each entity (Customers, Couriers, etc.) has its own HTML page displaying existing data in a structured