



RiderPro App Specification

You are to generate a **full-stack app** called **RiderPro** that manages real-time shipment operations for delivery and pickup riders.

🔑 Key Requirements

1. Shipment Management (Core Functionality)

- App receives real-time payloads with shipment details:
 - `type: "delivery" or "pickup"`.
 - `order details`: name, mobile, address, cost, delivery time, route/group name, assigned employee ID, etc.
- Riders can:
 - View assigned shipments.
 - Update status: `"Delivered"`, `"Picked Up"`, `"Returned"`, `"Cancelled"`.
 - Capture **signature + photo** on completion (delivery or pickup).
 - This capture serves as the **acknowledgement of delivery/pickup**.
 - It is saved against that shipment in the database.
 - It can also be **synced back to the external source** along with the status update.

2. Filtering & Batch Updates

- Riders can filter shipments by:
 - Time.
 - Route/group name.
 - Type (delivery/pickup).
 - Status.
- Allow **batch updates** of order statuses + individual updates.

3. Database Management

- Use **SQLite** for simplicity with two databases:
 - **Live DB**: resets every day via Node schedule service.
 - **Replica DB**: retains last 3 days of data (older auto-cleaned by Node schedule).
- Both `sqlite.db` (live) and `replica_sqlite.db` should be stored inside the **db folder** for better structure.
- Optimize queries for fast filtering and batch updates.

4. Dashboard (Homepage)

- Dashboard displays real-time metrics:

- Total shipments.
- Shipments grouped by:
 - Status.
 - Type.
 - Route/group name.

- Must auto-refresh using **Live DB**.

5. External Data Sync

- Any action (status update, acknowledgement, etc.) must be **synced to an external API**.
- Sync includes:
 - Shipment status.
 - Signature and photo acknowledgement (if applicable).
- Implement **retry + error handling** for failed syncs.

6. API Endpoints

Create robust REST API endpoints for:

Inbound Payloads (Receiving Data)

- **POST /shipments**
 - Receives a new shipment payload (delivery/pickup).
 - Stores it in both **Live DB** and **Replica DB**.

Outbound Payloads (External Sync)

- Automatic sync service sends updates to external API:
 - Status changes.
 - Acknowledgement data (signature + photo).
 - Retries on failure.

Shipment Management

- **GET /shipments** → List shipments (supports query params: status, type, time, route).
- **PATCH /shipments/:id** → Update a single shipment's status.
- **PATCH /shipments/batch** → Update multiple shipments' statuses in one request.
- **POST /shipments/:id/acknowledgement** → Upload signature + photo for acknowledgement (links to that shipment).

Dashboard

- **GET /dashboard**
 - Returns counts grouped by status, type, route.
 - Uses **Live DB** for real-time insights.

7. UI/UX

- Mobile-first design (for riders on smartphones).
- Simple shipment list with filters + batch actions.
- Shipment detail page with:
 - Shipment shown in cards style
 - Status update buttons.
 - Capture **signature + photo upload** for acknowledgement.
- Dashboard page for managers with real-time insights.

8. Global Error Boundary

- Add logging + monitoring for API, DB, and sync failures.
 - Display fallback error messages gracefully in UI.
-

📁 Project Structure

None

```
RiderPro/
  └── client/
    ├── src/          # Frontend source (React + Tailwind + Vite)
    |   ├── components/  # Reusable UI components
    |   |   ├── ShipmentList.tsx
    |   |   ├── ShipmentDetail.tsx
    |   |   ├── Dashboard.tsx
    |   |   └── Filters.tsx
    |   ├── pages/      # App pages
    |   |   ├── Home.tsx
    |   |   ├── Shipments.tsx
    |   |   └── DashboardPage.tsx
    |   ├── api/        # API service layer (frontend)
    |   |   └── shipments.ts
    |   └── App.tsx
    └── index.html
  └── server/
    └── src/
      ├── api/          # API routes
      |   ├── shipments.ts
      |   ├── dashboard.ts
      |   └── sync.ts
      ├── hook/         # Scheduled jobs (daily reset, cleanup)
      |   ├── resetLiveDb.ts
      |   └── cleanupReplicaDb.ts
      ├── db/           # DB connection + queries + sqlite files
      |   ├── sqlite.db
      |   ├── replica_sqlite.db
      |   ├── connection.ts
      |   └── queries.ts
      └── utils/        # Helpers
          ├── errorHandler.ts
          ├── externalSync.ts
          └── logger.ts
  └── public/
  └── node_modules/
  └── .env
  └── .gitignore
  └── LICENSE
  └── README.md
  └── tailwind.config.js
  └── vite.config.ts
  └── package.json
```

└ package-lock.json

Payload Schemas

Shipment Payload (Inbound POST /shipments)

JSON

```
{  
  "id": "uuid",  
  "type": "delivery", // or "pickup"  
  "customerName": "John Doe",  
  "customerMobile": "+91-9876543210",  
  "address": "123 Street, City",  
  "cost": 500,  
  "deliveryTime": "2025-08-28T15:00:00Z",  
  "routeName": "Route A",  
  "employeeId": "EMP123",  
  "status": "Assigned" // default status  
}
```

Status Update Payload (PATCH /shipments/:id)

JSON

```
{  
  "status": "Delivered" // or "Cancelled", "Returned", "Picked Up"  
}
```

Batch Status Update Payload (PATCH /shipments/batch)

JSON

```
{  
  "updates": [  
    { "id": "uuid1", "status": "Delivered" },  
    { "id": "uuid2", "status": "Cancelled" }  
  ]  
}
```

Acknowledgement Payload (`POST /shipments/:id/acknowledgement`)

Multipart Form Data:

- `signature`: Image file (PNG/JPEG).
- `photo`: Image file (PNG/JPEG).

Saved internally as:

```
JSON
{
  "shipmentId": "uuid",
  "signatureUrl": "/uploads/signatures/uuid.png",
  "photoUrl": "/uploads/photos/uuid.png",
  "capturedAt": "2025-08-28T15:10:00Z"
}
```

Outbound Sync Payload (to External API)

```
JSON
{
  "shipmentId": "uuid",
  "status": "Delivered",
  "syncedAt": "2025-08-28T15:11:00Z",
  "acknowledgement": {
    "signatureUrl": "https://cdn.riderpro.com/signatures/uuid.png",
    "photoUrl": "https://cdn.riderpro.com/photos/uuid.png",
    "capturedAt": "2025-08-28T15:10:00Z"
  }
}
```

Tech Stack

- **Frontend:** React + TailwindCSS + Vite
 - **Backend:** Node.js + Express + SQLite
 - **Task Scheduling:** `node-cron` (for reset + cleanup jobs)
 - **External Sync:** Axios with retry logic
 - **File Uploads:** Multer (for photos/signatures)
-

Deliverables

- Modular **full-stack codebase** with the above structure.
- Working shipment lifecycle (assign → deliver/pickup/return → acknowledgement → sync).
- Delivery/pickup acknowledgement with **signature + photo capture**, saved against shipment and synced externally.
- Real-time dashboard.
- Filters and batch update logic.
- External API sync with error handling.
- Scheduled DB jobs (reset + cleanup).