

Automatic Conveyor Belt Router

Author: Mamidipelli Krishna Hasini

1. System Overview

The **Automatic Conveyor Belt Router** is an intelligent logistics system designed to automate the sorting of parcels. It uses Computer Vision (OCR) to read labels from physical packages and queries a high-performance database to determine the routing direction (Straight, Left, Right) in real-time.

1.1 Architecture

The system follows a Client-Server architecture:

- **Frontend:** React (Vite) - Handles user interaction, camera input, and dashboard visualization.
 - **Backend:** FastAPI (Python) - Handles business logic, OCR processing, and database interactions.
 - **Database:** PostgreSQL (Supabase) - Stores user credentials, city mappings, and routing rules.
 - **AI Engine:** Tesseract OCR & OpenCV - Extracts text from images.
-

Employee portal to enter data

The screenshot shows a web browser window with a blue header bar containing the AutoRouter logo and navigation links for Home, About, Admin, and Employee. Below the header is a light gray content area. In the center, there is a white rectangular form titled "Employee Portal". It contains two input fields: the first is labeled "admin" and the second is labeled ".....". Below these fields is a blue "Login" button. Underneath the button, a small error message reads "Invalid Credentials. Please check username/password."

2. Admin portal to add data

The screenshot shows a web browser window with a blue header bar containing the AutoRouter logo and navigation links for Home, About, Admin, and Employee. Below the header is a light gray content area. In the center, there is a white rectangular form titled "Admin Portal". It contains two input fields: the first is labeled "admin" and the second is labeled ".....". Below these fields is a blue "Login" button. Underneath the button, a small error message reads "Invalid Credentials. Please check username/password."

3. Scanner portal to input the courier image

The screenshot shows a web application titled "AutoRouter" with a blue header bar containing the logo and navigation links for Home, About, Admin, and Employee.

The main content area is titled "Smart Conveyor Routing" and includes a sub-instruction: "Upload a parcel image to automatically determine the conveyor direction."

A central callout box is labeled "1. Scan Parcel". It contains a file input field with the placeholder "Choose File No file chosen" and a blue "Extract Data" button below it.

4. Extracted information shown under this

The screenshot shows the same "AutoRouter" interface, but the "Extracted Info" section is now populated with data.

The "1. Scan Parcel" section shows the file "Screenshot 2025-11-26 224915.png" has been uploaded.

The "2. Extracted Info" section displays the following details:

- Source:** CHIPARAKOTA (ID: 4286)
- Destination:** KATHANAGARH (ID: 2421)
- Type:** Normal

A blue "Find Route" button is located at the bottom of this section.

5 find route will give the route decision by our system

The screenshot shows a three-step process for finding a route:

- 1. Scan Parcel**: A file named "Screenshot 2025-11-26 224915.png" is uploaded, and a blue "Extract Data" button is visible.
- 2. Extracted Info**: Displays source (CHIPARAKOTA ID: 4286), destination (KATHANAGARH ID: 2421), and type (Normal). A blue "Find Route" button is present.
- 3. Routing Decision**: Shows a left turn icon with the text "LEFT" and "Route Code: 1".

6.Employee Portal has option to add new cities to the database and new routes to the system

The screenshot shows the employee portal interface for adding a new routing rule:

- Data Points** sidebar includes "Cities Database", "My Profile", and "Refresh System".
- Add New Routing Rule** form:
 - Source City and Destination City input fields.
 - Parcel Type dropdown set to "Normal (0)".
 - Route dropdown set to "Straight".
- Add Entry** button.
- Recent Data Entries** table header with columns: Source, Dest, Type, Route.

2. Database Schema

The system uses a relational PostgreSQL database. Below are the table definitions.

2.1 admin-logins & employee-logins

Stores authentication credentials for different roles.

2.2 cities

Master list of valid source/destination locations.

2.3 datapoints (Routing Rules)

The core logic table. Defines how a parcel should move based on source, destination, and type.

3. API Reference

3.1 Authentication

Login

- **Endpoint:** POST /login/{role}
- **Params:** role (string) -> "admin" or "employee"

Body:

JSON

```
{  
  "username": "admin",  
  "password": "admin123"  
}
```

- **Response:** Returns User ID and Role.

3.2 Intelligent Routing (The Core Logic)

Extract Text from Image

- **Endpoint:** POST /extract-from-image
- **Body:** multipart/form-data (File upload)
- **Process:**
 1. Image is received and pre-processed using OpenCV (Grayscale, Thresholding).
 2. Tesseract OCR extracts raw text.
 3. Regex patterns identify "Source", "Destination", and "Parcel Type".

Response:

```
JSON
{
  "source_city": "MUMBAI",
  "source_id": 101,
  "dest_city": "DELHI",
  "dest_id": 204,
  "type": 0
}
```

•

Find Route

- **Endpoint:** POST /find-route

Body:

```
JSON
{
  "source_city_id": 101,
  "destination_city_id": 204,
  "parcel_type": 0
}
```

•

- **Logic:** Checks the in-memory ROUTE_CACHE ($O(1)$ lookup) for a match.

Response:

```
JSON
{
  "found": true,
  "route_code": 2,
  "direction": "Right"
}
```

•

3.3 Management Endpoints (Admin Only)

Method	Endpoint	Description
GET	/users/{role}	Get list of admins or employees.
POST	/add-user/{role}	Create a new user account.
DELETE	/remove-user/{role}/{id}	Delete a user account.
GET	/cities	List all registered cities.
POST	/add-city	Register a new city.
POST	/add-datapoint	Add a new routing rule (e.g., Mumbai->Delhi = Left).
POST	/refresh-data	Forces backend to reload DB rules into RAM cache.

[Export to Sheets](#)

4. Algorithms & Logic

4.1 Image Pre-processing Pipeline

To ensure high accuracy in reading labels, images undergo the following OpenCV transformations before OCR:

1. **Upscaling:** `cv2.resize(fx=3, fy=3)` - Increases image resolution.
2. **Grayscale:** `cv2.cvtColor(BGR2GRAY)` - Removes color noise.
3. **Otsu's Thresholding:** `cv2.threshold` - Converts to strict Black & White to separate text from background.

4.2 Caching Mechanism (Performance Optimization)

Database queries are slow. To ensure the conveyor belt runs smoothly:

1. **On Startup:** The server fetches **all** rows from the `datapoints` table.
2. **Memory Map:** It builds a Python Dictionary (Hash Map):
 - o **Key:** (`source_id`, `dest_id`, `type`)
 - o **Value:** `route_direction`
3. **Lookup:** When scanning a parcel, the system checks this Hash Map.
 - o **Time Complexity:** O(1) (Instant).
 - o **Benefit:** No database lag during operation.

5. Deployment Configuration

5.1 Environment Variables (.env)

DATABASE_URL

```
postgresql://postgres.clgjswrlcwzmdxhhegtk:adhithya  
365@aws-1-ap-south-1.pooler.supabase.com:6543/postg  
res
```

,DEBUG

5.2 Docker Container

Because the backend requires **Tesseract OCR** (a C++ binary), it cannot run on standard Python hosting. It runs in a Docker container:

- **Base Image:** python:3.10-slim
 - **System Dependencies:** tesseract-ocr, libgl1 (for OpenCV).
 - **Port:** 8000
-

5. Deployment Under Process

Server: <https://automaticconveyorbeltroutersystemdesign-l3l8.onrender.com/docs>

Client :<https://automaticconveyorbeltroutersystemdesign.onrender.com>