BITS

from collecting... to connecting...

CargoConnect Project Documentation

Overview

CargoConnect is a platform designed to connect users with transport providers for freight services. The application allows users to register, book freight services, track deliveries, and make payments. Drivers can accept bookings and update delivery statuses. Admins can manage users, view statistics, and review feedback.

Table of Contents

Setup and Installation

Prerequisites

Before you can set up and run CargoConnect, ensure that the following software is installed on your system:
- Python (version 3.8 or higher)
- pip (Python Package Installer)
- SQLite (for local database management)
- Git (for version control)

Installation Steps

1. Clone the Repository:
  ```bash
  git clone https://github.com/your-repo/CargoConnect.git
  cd CargoConnect
  ```
2. Create a Virtual Environment:
  ```bash
  python -m venv venv
  source venv/bin/activate  # On Windows use `venv\Scripts\activate`
  ```
3. Install Dependencies:
  ```bash
  pip install -r requirements.txt
  ```

4. Set Up Environment Variables:
   Create a `.env` file in the root directory of your project and add the following variables:
   ```env
   SESSION_SECRET=your_session_secret
   ```
5. Run the Application:
   ```bash
   python app.py
   ```
   The application will be available at http://localhost:5000.

Dependencies

CargoConnect relies on the following key libraries:
- Flask: A micro web framework for building the application.
- Flask-SQLAlchemy: Extension for Flask that adds support for SQLAlchemy, an ORM for handling the database.
- Flask-Login: Provides session management for user authentication.
- Flask-Migrate: Provides database migration support.
- ChapaService: Service used for handling payment processing.

Code Structure

Overview of Important Files

1. app.py: This is the main entry point for the application. It initializes the Flask app, sets up the database configuration, and loads essential extensions like SQLAlchemy, Migrate, and LoginManager.
2. views.py: Contains all the route handlers for different user actions, such as registration, login, booking creation, payment processing, and driver/administrator functionalities.
3. models.py (Not provided here but assumed): Defines the database models for User, Driver, Booking, and Payment.
4. services/chapa_service.py (Not provided here but assumed): Contains the logic for integrating with the Chapa payment service.

Routes Overview

User Routes:
- POST /register: Handles user registration.
- POST /login: Authenticates a user and starts a session.
- GET /profile: Displays user profile information.

Booking Routes:
- POST /create_booking: Allows users to create new bookings.
- GET /booking_list: Displays a list of bookings for the logged-in user.
- POST /booking/<int:booking_id>/pay: Processes payment for a booking.

Driver Routes:
- GET /driver/dashboard: Displays a driver’s dashboard with active deliveries and available bookings.
- POST /driver/accept-booking/<int:booking_id>: Allows drivers to accept a pending booking.
- POST /driver/update-status/<int:booking_id>: Allows drivers to update the status of a booking.

Admin Routes:
- GET /admin/dashboard: Displays the admin dashboard with statistics like the number of users, bookings, and drivers.

User Guide

Booking a Freight Service

1. Sign Up and Login: Register for an account by providing your details. After registration, log in with your credentials.
2. Creating a Booking: Once logged in, navigate to the 'Create Booking' page. Provide pickup and delivery locations, cargo type, weight, and any additional details. Confirm the booking, and it will be added to your booking list.
3. Tracking Your Booking: You can view the status of your bookings on the 'Booking List' page. You will also have the option to track the real-time location of your driver.
4. Payment Processing: Once a booking is confirmed, you can proceed to the payment page to complete the transaction. Payment is processed using the Chapa service, and you will be redirected to the payment gateway.

Driver Features

1. Driver Dashboard: Drivers can view available bookings and active deliveries from their dashboard. They can toggle their availability and accept new bookings.
2. Booking Acceptance: Drivers can accept bookings that are marked as 'pending.' Upon accepting a booking, the driver’s status changes to 'in progress.'
3. Updating Booking Status: Drivers can update the status of a booking as it progresses, including statuses like 'picked up,' 'in progress,' and 'completed.'

Admin Features

1. Admin Dashboard: Admins can view statistics like the total number of users, drivers, and bookings. Admins can also see recent bookings and monitor the overall system activity.

How to Run the Application

To run CargoConnect on your local machine:
1. Clone the repository:

```bash
   git clone https://github.com/your-repo/CargoConnect.git
   cd CargoConnect
   ```
2. Create a virtual environment and activate it:
   ```bash
   python -m venv venv
   source venv/bin/activate  # On Windows use `venv\Scripts\activate`
   ```
3. Install the required dependencies:
   ```bash
   pip install -r requirements.txt
   ```
4. Set up your environment variables by creating a `.env` file:
   ```env
   SESSION_SECRET=your_session_secret
   ```
5. Run the Flask app:
   ```bash
   python app.py
   ```

Visit `http://localhost:5000` in your browser to access the application.

Additional Information

Requirements

Functional Requirements:

- User Management: Users can register, log in, and manage their profiles.

- Cargo Booking: Customers can request cargo transport services.

- Payment Processing: Secure transactions using third-party gateways.

- Tracking & Updates: Users can monitor cargo status.

- Admin Management: Admins can oversee user activity and system settings.

Non-Functional Requirements:

- Scalability: The system should handle a growing number of users and transactions.

- Security: Implement authentication and data encryption.

- Performance: Optimize response times for booking and tracking.

- Availability: Ensure the system is available with minimal downtime.

Database (SQLAlchemy)

CargoConnect uses Flask-SQLAlchemy for database management. The key tables include:

- User (id, name, email, password, role)

- Cargo (id, user_id, description, weight, status)

- Booking (id, user_id, cargo_id, pickup, destination, status)

- Payment (id, booking_id, amount, status, transaction_id)

Example Model (User):

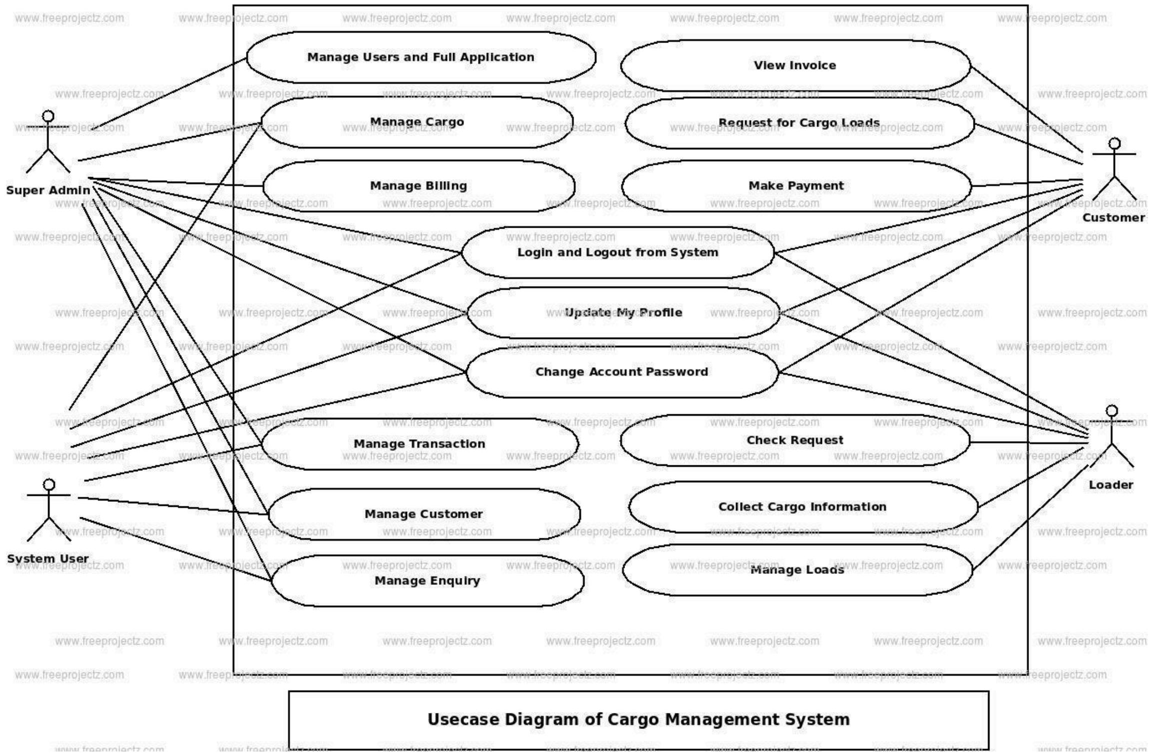
```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    role = db.Column(db.String(50), nullable=False)
```

Use Case Diagram

The following diagram represents the Cargo Management System and its interactions:

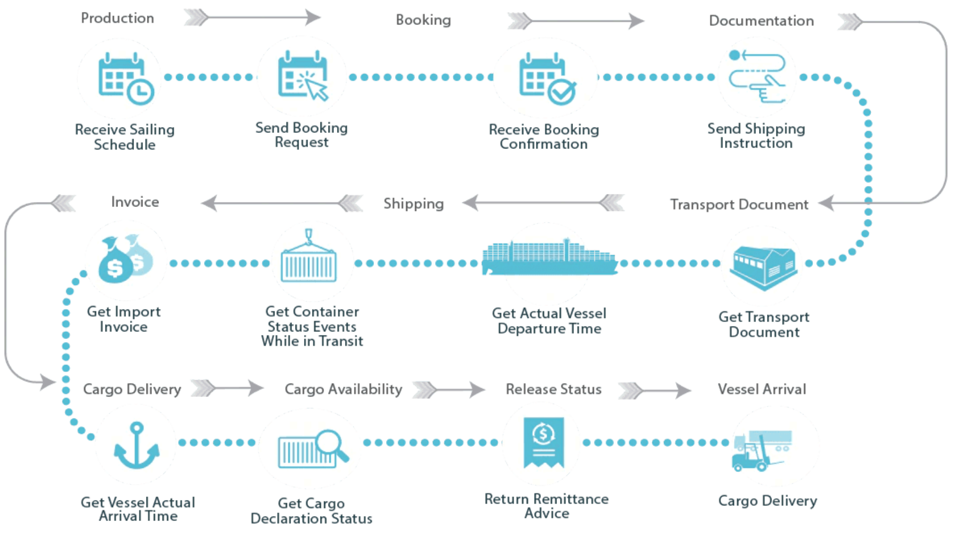


Explanation:

- Super Admin: Manages users, transactions, and billing.
- System User: Handles customer interactions and cargo management.
- Customer: Requests cargo loads, makes payments, and views invoices.
- Loader: Checks requests, manages cargo loads, and collects cargo information.

Cargo Workflow Diagram

The following diagram illustrates the workflow of cargo shipment:



Explanation:

1. Production: Receive the sailing schedule and send a booking request.
2. Booking: Receive booking confirmation.
3. Documentation: Send shipping instructions.
4. Shipping: Monitor container status and vessel departure.
5. Invoice & Transport: Process payments and generate transport documents.
6. Cargo Arrival & Delivery: Receive cargo declaration status, arrival time, and final delivery.