Cargo Connect: Improving Airport Freight Flow

High Level Architecture vs 5.0

SE 6387 Advanced Software Engineering Project
Prof. R.Z. Wenkstern – UT Dallas

Date: 05/09/2025

Group 1
FNU ANUJA
BRADLEY EVAN STOVER
CHIJIOKE ELISHA-WIGWE
SMIT SONESHBHAI PATEL

Revision History

Version	Date	Decription	Authors
1.0	03/25/2025	Completed initial draft	FA, SP, BS, CE
2.0	04/01/2025	Completed second draft	FA, SP, BS, CE
3.0	04/17/2025	Completed third draft	FA, SP, BS, CE
4.0	04/24/2025	Completed fourth draft	FA, SP, BS, CE
5.0	05/09/2025	Completed final document	FA, SP, BS, CE

Contents

1. Introduction

The Cargo Connect Project seeks to make the transportation of freight from various origins to various destinations easier, with particular focus on scheduling trucks to, from airports and at airports. The system provides end-to-end logistics solutions, such as retrieve information, truck allocation, traffic coordination, and real-time status.

Key Goals:

1. Efficiency: Automate scheduling processes to minimize manual coordination and turnaround time.
2. Scalability: Accommodate growing volumes of trucks and routes without compromising performance.
3. Reliability: Provide continuous operation, even with high loads or local system failure.
4. Integration: Be integrated with external services (e.g., Airport System, DALI) to gain access to real-time data and schedule in a more informed way.

At an architectural level, it adopts a layer-based approach with Presentation, Application, Domain, Data Access, and Infrastructure layers isolating concerns. This maximizes code readability and makes it simpler to extend or change the system.

2. Architectural Factors

2.1. Functional requirements

Manage Scheduling	The system shall schedule trucks to and from the airport, at the airport, including dock assignment and tracking truck status.
Retrieving Information	The system shall pull in necessary information such as truck information, airport information (parking conditions, flight schedule), and road traffic information.
Truck data management	The system shall store and display truck data for reporting, analysis, and real-time decision-making (e.g., new truck addition, status update).
External integration	The system shall integrate with Mobile App (drivers/staff), Time service (for scheduling triggers), DALI System (traffic management and optimized routes), and Airport System (flight arrival estimated data)

2.2. Non-Functional requirements

Performance	The system shall be able to process large numbers of scheduling requests and able to react promptly to real-time events (traffic, truck status).
Scalability	The system shall be able to support a growing number of trucks, routes, airports, and external data feeds without affecting performance.

Availability	The system must be operational with minimal downtime and tolerate partial failures.
Security	The system shall protect sensitive data, verify communication with external API's and implement authentication and authorization
Usability	The system shall provide user friendly interface for all users.

2.3. Constraints

1. Dependence on the existing airport system's API and data accuracy.
2. Potential network latency affecting real-time updates.
3. Limited availability of parking and docking slots during peak hours.
4. Freight truck schedules must align with flight schedules, requiring strict adherence to narrow operational windows, particularly during peak commute hours and delivery deadlines.

3. Architectural Decisions

The architectural decisions for Cargo Connect have been chosen to leverage a lightweight, flexible backend with Python Flask, robust data storage with PostgreSQL, modern Android development using Kotlin and Jetpack Compose, and simplified deployment on Heroku. With JWT for stateless security, JSON for efficient communication. This tech stack offers a scalable, maintainable, and secure freight management solution.

1. **Backend Framework: Python Flask**

   We have planned to use Python Flask as the primary backend framework. Flask is lightweight, flexible, and perfect for building RESTful APIs. It favors quick development and boasts a mature extension ecosystem. It facilitates easier development of a JSON-based API for mobile and web client communication. It also facilitates easier integration with JWT-based authentication and PostgreSQL.

2. **Database: PostgreSQL**

   We are using PostgreSQL as the relational database for storing fundamental freight management data (schedules, truck records, user profiles, etc.). This offers excellent ACID compliance, good support for advanced queries, and scalability. It integrates well with Python with libraries like SQLAlchemy. This indeed ensures data consistency and transactional integrity for critical scheduling operations. It supports high-performance querying and reporting on freight data.

3. **Mobile App: Android (Kotlin + Jetpack Compose)**

   We are using Kotlin as the primary language and Jetpack Compose for UI to develop the Android app. Kotlin is new, expressive, and fully supported for Android development. Jetpack Compose declares UI development easier and results in well-maintained code. It provides a modern, responsive, and stunning user interface for

truckers and staff. This also enables faster UI cycles and a reactive UI that stays synchronized with backend changes.

4. **Hosting: Heroku**

   This helps hosting the backend on Heroku. Heroku simplifies deployment and scaling via a managed platform. It provides add-ons (e.g., for PostgreSQL, logging, monitoring) that simplify operations. It also decreases infrastructure management overhead and allows for quick deployments and allows for automatic scaling and integration with CI/CD pipelines without hassle.

5. **Authentication: JWT**

   We are using JSON Web Tokens (JWT) for stateless authentication and authorization. JWTs provide secure, token-based authentication that is best suited to RESTful APIs. Clients send a JWT in the Authorization header (Bearer <token>) with each request. Thus, making security management easier and reduces overhead on backend resources.

6. **Data Format: JSON**

   We have JSON as the default data format between the backend and mobile clients. JSON is lightweight, human-readable, and natively implemented in Python and Kotlin. It is the default for REST APIs, and easier to integrate across platforms. It ensures smooth communication between the Flask API and the Android app and makes debugging and serialization/deserialization of data easier on both sides.

8. Architectural Considerations Overall

Modular Design: Using Flask blueprints or other patterns to split the API into modules (e.g., scheduling, truck management, notifications).

Layered Architecture:

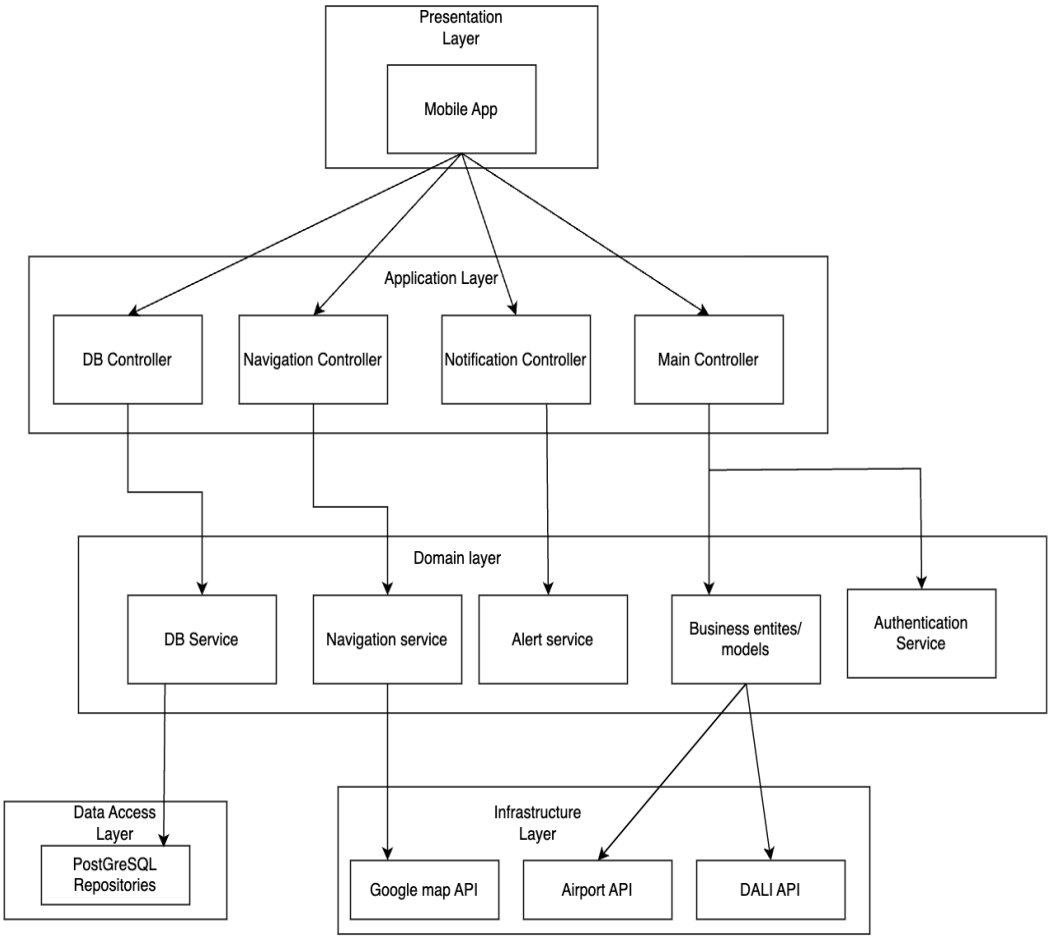
Presentation: UI/UX handled by the Android app (through Jetpack Compose).
Application/Backend API: Python Flask routes receive incoming requests.
Domain/Business Logic: Buried inside Flask services or external modules.
Data Access: PostgreSQL accessed through ORM (e.g., SQLAlchemy) or raw SQL queries.
Infrastructure: Third-party integrations (third-party APIs for traffic/airport data).
CI/CD: Utilize Heroku's pipelines or connect to services like GitHub Actions to perform automatic testing, builds, and deploys.
Security**:** Deploy HTTPS on all endpoints, secure JWT secret handling, and utilize token expiration and refresh to further protect it.

4. Architectural Views

4.1. Logical View

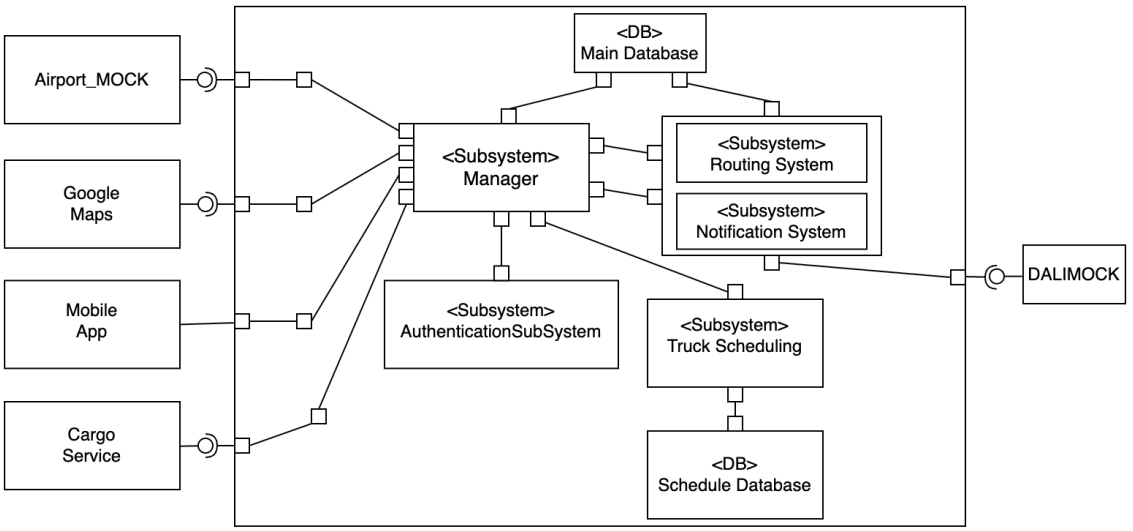
4.1.1. Layered View

Layered perspective is one of the architectures for organizing and visualizing a software system by breaking it into distinct layers or tiers where each layer is used for some specific purpose and typically only depends on the layers that follow it (or is completely independent at times). The idea is to partition the concerns so that each layer deals with some specific aspect of the application, so the system is easier to write, maintain, and extend.



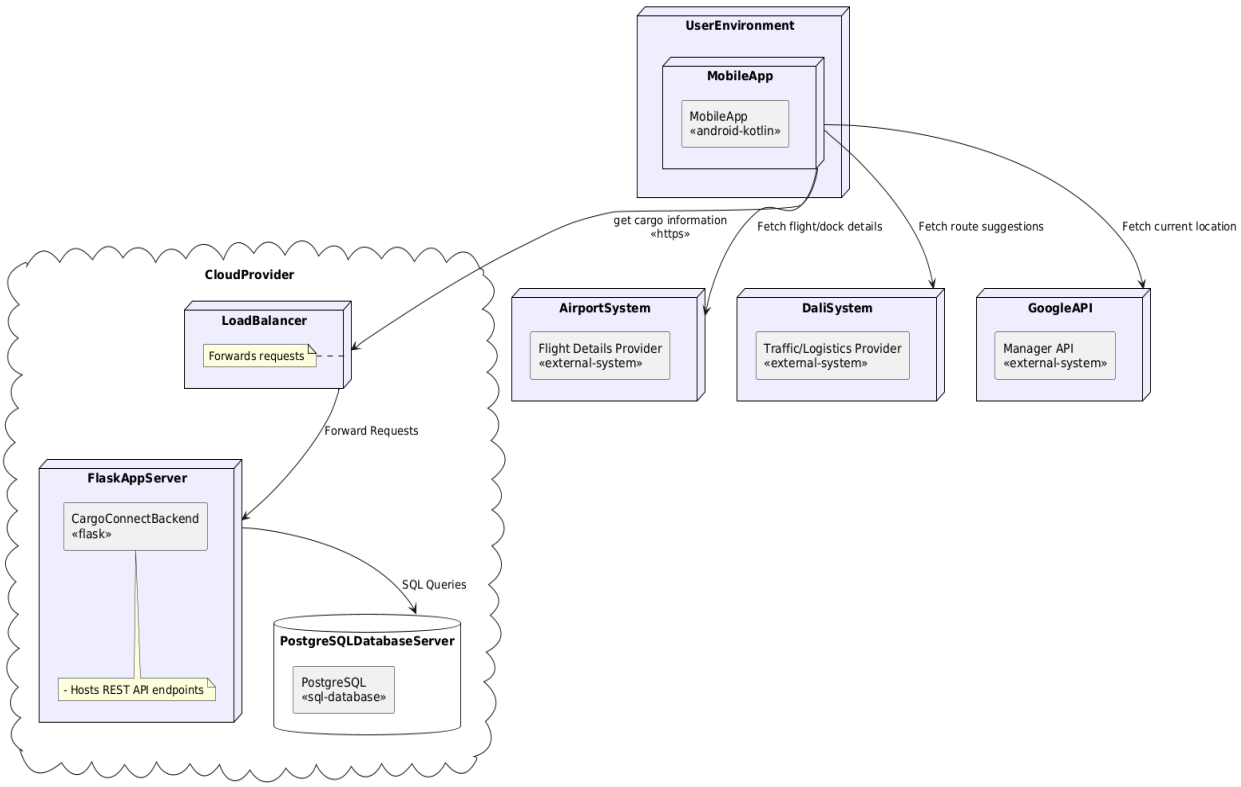
4.1.2. System and Subsystem Component Diagrams

Subsystems are a type of stereotyped component that represent independent, behavioral units in a system. Subsystems are used in class, component, and use-case diagrams to represent large-scale components in the system that you are modeling. We can model an entire system as a hierarchy of subsystems and also define the behavior that each subsystem represents by specifying interfaces to the subsystems and the operations that support the interfaces.



4.3. Deployment View

Deployment diagram is a type of UML diagram that visually represents the physical deployment software artifacts (e.g., programs or databases) onto hardware nodes (e.g., servers or devices), showing the runtime architecture of the system.


UserEnvironment
MobileApp
MobileApp
«android-kotlin»
get cargo information
«https»
Fetch flight/dock details
Fetch route suggestions
Fetch current location
CloudProvider
LoadBalancer
Forwards requests
Forward Requests
FlaskAppServer
CargoConnectBackend
«flask»
- Hosts REST API endpoints
SQL Queries
PostgreSQLDatabaseServer
PostgreSQL
«sql-database»
AirportSystem
Flight Details Provider
«external-system»
DaliSystem
Traffic/Logistics Provider
«external-system»
GoogleAPI
Manager API
«external-system»

Appendix A: Glossary

Term	Definition
Cargo Connect	A freight management solution aimed at improving airport freight flow by automating scheduling, truck allocation, and real-time tracking of freight operations.
API	A set of protocols and tools that allow different software applications to communicate with each other
Python Flask	A lightweight and flexible web framework used for building RESTful APIs. It facilitates rapid development and integration of various services like JWT authentication.
PostgreSQL	An open-source relational database system used to store and manage critical freight management data such as schedules, truck records, and user profiles
Deployment Diagram	A type of UML diagram that illustrates the physical distribution of software components (artifacts) across hardware nodes, showing how the system is deployed in a runtime environment.
Architectural Views	Different perspectives (such as logical, layered, and deployment views) used to analyze and represent the various aspects of the system's structure and behavior
CI/CD	A set of practices and tools designed to automate software integration, testing, and deployment. This ensures that changes are delivered quickly and reliably

Appendix B: References

[1] Requirement analysis

[2] Architectural Views: The State of Practice in Open-Source Software Projects