### 

### 1. Project Structure

logistics/

│

├── main.py # Entry point of the application, integrates Engine

├── models/

│ ├── package.py # Package class

│ ├── route.py # Route class

│ ├── truck.py # Truck class

│ ├── user.py #User class

│ └── \_\_init\_\_.py

│

├── data/

│ ├── application\_data.py # Manages in-memory data state (like ApplicationData)

│ └── \_\_init\_\_.py

│

├── services/

│ ├── command\_factory.py # Handles creation of command objects (like CommandFactory)

│ ├── commands/ # Directory for various command classes

│ │ ├── create\_package.py # Command to create a package

│ │ ├── create\_route.py # Command to create a route

│ │ ├── assign\_truck.py # Command to assign a truck to a route

│ │ └── \_\_init\_\_.py

│ └── \_\_init\_\_.py

├── errors/

│ ├── errors.py

├── ui/

│ ├── engine.py # Engine class that runs the command loop (like Engine)

│ └── \_\_init\_\_.py

│

├── utils/

│ ├── constants.py # Constants used across the application

│ ├── validators.py # Input validation functions

│ └── \_\_init\_\_.py

│

├── tests/

│ ├── test\_commands.py # Tests for command functionalities

│ └── \_\_init\_\_.py

│

└── README.md

### 2. Core Components

#### data/application\_data.py

* Manages all in-memory data like routes, packages, and trucks.
* Provides methods to search, create, and manage data entries similar to the provided example.

#### services/command\_factory.py

* Initializes with an instance of ApplicationData.
* Responsible for parsing user input and creating appropriate command objects.
* Maps command strings to command classes, handling invalid command scenarios.

#### services/commands/\*.py

* Each file defines a specific command class, such as creating a package, assigning a truck, or creating a route.
* Commands interact with ApplicationData to execute business logic.

#### ui/engine.py

* Contains the Engine class, which runs the main command loop, receiving inputs and executing commands through the CommandFactory.
* Handles output and captures exceptions to provide feedback to the user.

### 3. Detailed File Contents

`main.py`

- Main script to run the application.

- Setup and initialization of the app.

- Main loop for handling user input and showing results.

`models.py`

- Classes for `Package`, `Route`, and `Truck`.

- Each class should include relevant attributes and methods for its operations.

`tests/`

- Unit tests for each component of the application.

- Mock data and scenarios to test functionalities thoroughly.

## 4. Data Scheme Design

|  |  |  |
| --- | --- | --- |
| Package | Route | Truck |
| `id`: Unique identifier | id: Unique identifier | id: Unique identifier |
| `start\_location`: Starting city | locations: Ordered list of cities | name: Model of the truck |
| `end\_location`: Destination city | departure\_time: Starting time from the first location | capacity: Maximum load capacity in kilograms |
| `weight`: Weight of the package in kilograms | arrival\_times: Expected arrival times at subsequent locations | range: Maximum range in kilometers |
| `customer\_info`: Contact details of the customer |  |  |

#### 

#### 4.1. Main.py

**from data.application\_data import ApplicationData**

**from services.command\_factory import CommandFactory**

**from ui.engine import Engine**

**def main():**

**app\_data = ApplicationData()**

**command\_factory = CommandFactory(app\_data)**

**engine = Engine(command\_factory)**

**engine.start()**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

# 5. Must

#### 5.1. Creating a Delivery Package

* **Properties**:
  + id (int or string): Unique identifier for the package. **Setter not needed** as ID should be immutable once assigned.
  + start\_location (string): Starting city of the package. **Setter not needed** as the start location is typically fixed post-creation.
  + end\_location (string): Destination city of the package. **Setter not needed** for similar reasons as start\_location.
  + weight (float): Weight of the package in kilograms. **Setter optional**, depending on whether weight can change after creation.
  + customer\_info (dict or custom class): Information about the customer such as name, contact details. **Setter optional** if updates to customer information are allowed.

#### 5.2. Creating a Delivery Route

* **Properties**:
  + id (int or string): Unique identifier for the route. **Setter not needed** as ID should be immutable once assigned.
  + locations (list of tuples): List containing tuples of locations with times, where the first location includes the departure time, and subsequent locations include the expected arrival times. **Setter not needed**; should manage locations through dedicated methods to add or update stops.

#### 5.3. Searching for a Route

* **Method**: Based on the package’s start and end locations, filter available routes.
* **Implementation**: A search function in the route management system that iterates through routes and matches start and end locations.

#### 5.4. Updating a Delivery Route

* **Assignment of a Free Truck**:
  + **Method**: A method to assign a truck that matches the required capacity and is not currently assigned to a route.
  + is\_available – if there is no arrange or are no packages or capacity left for the delivery
  + mark\_as\_unavailable
  + mark\_as\_available
  + update\_capacity – after summing the weight of every package in list of packages to update if there is capacity left (how much is it) or it is full
* **Assignment of a Delivery Package**:
  + **Method**: A method to add a package to a route, ensuring the total weight does not exceed the truck's capacity.

# 6. User

### 6.1. General User Attributes

Both types of users share common attributes:

* **User ID**: A unique identifier for each user.
* **Name**: The user's full name, which helps in identifying and addressing the user within the system.

### 6.2. Client

Clients are typically customers who are using the logistics company to ship packages. Their interaction with the system is primarily concerned with the packages they are sending or receiving.

#### 6.2.1. Attributes

* Inherits from the general User attributes.
* **Contact Information**: Email and phone number to receive notifications and updates about their packages.

#### 6.2.2. Capabilities

* **Track Package**: Clients can check the status and location of their packages. This includes details such as current location, expected arrival times, and any delays.
* **Update Contact Info**: Clients might need to update their contact details like phone numbers or email addresses.
* **View Package History**: Clients can view past shipments to track frequent shipments or analyze previous transactions.
* Last time logged – saving information

### 6.3. Manager

Managers are employees of the logistics company who manage the operational aspects of package delivery. This includes managing routes, vehicles, and packages.

#### 6.3.1 Attributes

* Inherits from the general User attributes.
* **Access Level**: An attribute that indicates the level of access within the system, such as basic, supervisor, or administrator, allowing for different permissions and capabilities.

#### 6.3.2 Capabilities

* **Create and Manage Packages**: Managers can enter new packages into the system, update package details, and delete packages when necessary.
* **Create and Update Routes**: Managers are responsible for creating routes for package delivery, updating existing routes, and optimizing routes based on various factors such as distance, package urgency, and vehicle availability.
* **Assign Vehicles to Routes**: Managers select and assign trucks to specific routes based on the route requirements and the vehicle's capacity and range.
* **Vehicle Management**: Managers can add new vehicles to the fleet, update vehicle information, and manage vehicle maintenance schedules.
* **User Management**: For managers with higher access levels, they can add or remove system users, assign roles, and manage user permissions.
* **Reporting and Analytics**: Managers can generate reports on delivery efficiency, vehicle usage, cost analysis, and more to help in strategic decision-making.

# 7. Should

#### 7.1. Utilizing Vehicle and Distance Data

* **Vehicles**: Store vehicle data as a list of Truck objects in ApplicationData. Include properties like id, name, capacity, and max\_range.

Something for creation:  
def create\_trucks():

app\_data = ApplicationData()

# Create Scania trucks

for i in range(1001, 1011): # 1001 to 1010 inclusive

app\_data.add\_truck(Truck(i, "Scania", 42000, 8000))

# Create Man trucks

for i in range(1011, 1026): # 1011 to 1025 inclusive

app\_data.add\_truck(Truck(i, "Man", 37000, 10000))

# Create Actros trucks

for i in range(1026, 1041): # 1026 to 1040 inclusive

app\_data.add\_truck(Truck(i, "Actros", 26000, 13000))

return app\_data

app\_data = create\_trucks()

app\_data.list\_trucks() # This will print out all the trucks with their details

**7.2. Distances**:

* Use a dictionary to store distances between cities. For instance, {'SYD': {'MEL': 877, 'ADL': 1376, ...}, ...}.
* This allows for easy lookup of distances when calculating route logistics or constraints.

distances = {

'SYD': {'MEL': 877, 'ADL': 1376, 'ASP': 2762, 'BRI': 909, 'DAR': 3935, 'PER': 4016},

'MEL': {'ADL': 725, 'ASP': 2255, 'BRI': 1765, 'DAR': 3509},

# More cities and distances here

} - from here we will know the km and where the truck will cross when doing long route

def get\_distance(start, end):

return distances[start][end] # Assumes that distances are symmetric

### 8. Functional Overview

The application will feature several key functionalities each supported by specific methods and classes:

#### 1. **Creating a Delivery Package**

* **Functionality**: Employees input package details including a unique ID, start location, end location, weight, and customer contact info.
* **Process**: The system creates a Package object and stores it in ApplicationData. Each package is uniquely identified to prevent data duplication and ensure traceability.

#### 2. **Creating a Delivery Route**

* **Functionality**: Routes are created with a unique ID and a list of locations, including a designated starting point with departure time and subsequent locations with expected arrival times.

--------------------------->1--------------first truck--------------->2------another truck-------->(3)

Reset – return the truck in first point

Truck info – starting point – different place – different scope - small routes 1 -> 2, 2 ->3, 3-> 4 - everyday and big routes 1-->5 – wait 1 day because they are in 2 days once

For every new delivery we will check first the long route if there is no long route we make schema from small routes

Continue – next delivery from the last point where the truck was (in the example case (3) )

* **Process**: The system constructs a Route object encapsulating these details. Routes are also stored in ApplicationData and can be accessed or modified through their unique IDs.

#### 3. **Searching for a Route**

* **Functionality**: Based on a package's start and end locations, the system identifies possible delivery routes.
* **Process**: A method within ApplicationData filters through the stored routes to find matches that connect the specified start and end locations.

#### 4. **Updating a Delivery Route**

* **Assigning a Free Truck**: The system assigns an available truck that matches the route requirements (capacity and range).
* **Assigning a Delivery Package**: The system adds a package to a specified route if the cumulative weight of the route is within the truck's capacity.
* **Process**: Route objects will have methods to assign trucks and packages, ensuring that logistical constraints are met.

#### 5. **Viewing Information about Routes, Packages, and Trucks**

* **Functionality**: Display comprehensive details about current routes, their assigned packages, and the trucks servicing them.
* **Process**: Queries against the ApplicationData can retrieve and display this information formatted for easy viewing.

# 7. Should

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | class Package | class Truck | class User | class ApplicationData | class Manager |
| package |  |  | track\_package | create\_package | create\_package |
|  |  | view\_package\_history | find\_package | delete\_package |
|  |  |  | assign\_package | update\_package |
| truck |  |  |  | add\_truck | add\_truck |
|  |  |  | assign\_truck | assign\_truck |
|  |  |  |  | update\_truck |
| route |  |  |  | create\_route | create\_route |
|  |  |  | find\_route | update\_route |
| user |  |  | upadate\_user\_info |  | add\_user |
|  |  |  |  | remove\_user |
|  |  |  |  | assign\_role |