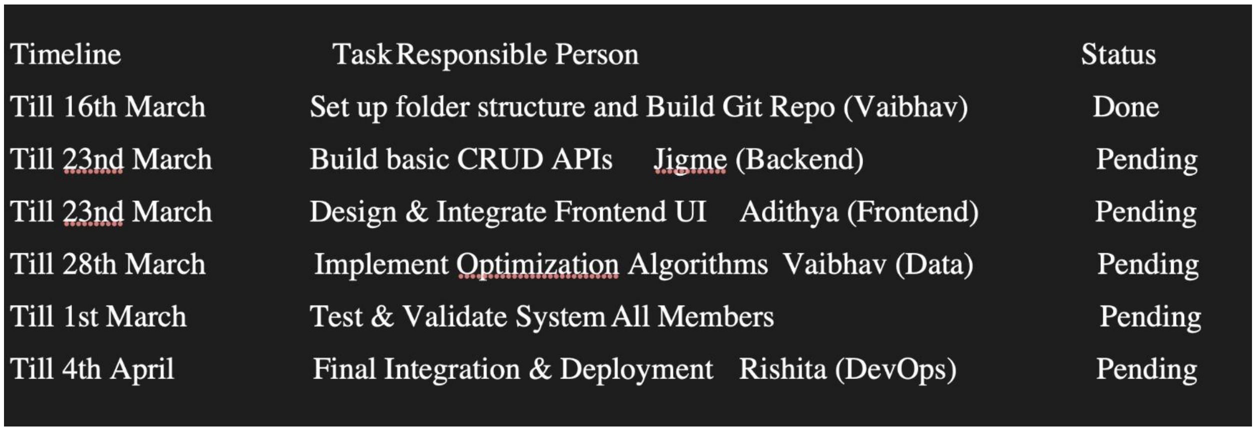

Timeline	Task	Responsible Person	Status
Till 16th March	Set up folder structure and Build Git Repo	(Vaibhav)	Done
Till 23nd March	Build basic CRUD APIs	Jigme (Backend)	Pending
Till 23nd March	Design & Integrate Frontend UI	Adithya (Frontend)	Pending
Till 28th March	Implement Optimization Algorithms	Vaibhav (Data)	Pending
Till 1st March	Test & Validate System	All Members	Pending
Till 4th April	Final Integration & Deployment	Rishita (DevOps)	Pending

□ Step 1: Frontend – User Interaction Layer (Adithya's Responsibility)

The **Frontend Developer** is responsible for designing the **User Interface (UI)** that astronauts will use to interact with the system. This interface should be:

✅**Simple and Intuitive** – Easy for astronauts to quickly access and manage cargo.

✅**Interactive** – Users can perform actions like **searching, adding, retrieving**, and **tracking waste**.

✅**Real-time** – Reflects **current storage status** and updates after each action.

□ **Frontend Workflow**:

1. **Design UI Components**:
    - Dashboard: Overview of total cargo (Available, Retrieved, Expired).
    - Search Bar: Locate items by name, type, or priority.
    - Add Item Form: Input form to add new cargo (name, category, expiry, priority).
    - Retrieve Button: Initiates item retrieval.
    - Waste Tracker: Lists expired or depleted items.
2. **Connect to Backend via APIs**:
    - **GET /api/items** – Fetch all available cargo.
    - **POST /api/items** – Add a new item to storage.
    - **PUT /api/items/retrieve –** Retrieve an item from storage.
    - **DELETE /api/items/waste** – Mark an item as waste.

□ Step 2: Backend – Logic & Data Handling (Jigme's Responsibility)

The **Backend Developer** handles the **core logic** that processes astronaut requests, communicates with the database, and integrates the optimization algorithms.

□ **Backend Workflow**:

1. **Define Database Schema**:

Items Table:

```
id (Primary Key)
name (String)
category (String)
location (String)
priority (Integer)
expiry_date (Date)
status (Available, Retrieved, Waste)
```

2. **Create Core API Endpoints**:
    - **GET /api/items** – Return all stored items (with filtering options).
    - **POST /api/items** – Add new items to the database.
    - **PUT /api/items/retrieve** – Retrieve items and update status.
    - **DELETE /api/items/waste** – Mark expired items for disposal.
3. **Implement Business Logic**:
    - Validate new item data.
    - Ensure duplicate items are not stored.
    - Efficient retrieval using optimized algorithms from the **Data Engineer**.

□ Step 3: Data & Simulation – Optimization & Efficiency (Vaibhav's Responsibility)

The **Data & Simulation Engineer** ensures that **item placement and retrieval** are optimized for speed and efficiency, especially under **computational constraints**.

□ **Optimization Workflow**:

1. **Design Efficient Storage Algorithms**:
    - **Greedy Algorithm** for fast retrieval based on priority & location.
    - **Knapsack Approach** to optimize space utilization when storage is tight.
2. **Implement Time Simulation**:
    - Simulate future cargo usage by advancing time.
    - Identify expiring items and trigger waste management.
3. **Optimize Database Queries**:
    - Use **Indexing** for faster searches on critical fields (e.g., `priority`, `expiry_date`).
    - Implement **Caching** for frequent queries (e.g., available items list).

□ Step 4: DevOps & Deployment – System Integration (Rishita's Responsibility)

The **DevOps Engineer** handles the **automation** and **deployment** process, ensuring that all parts (Frontend, Backend, Data Processing) are **integrated and operational**.

□ **Deployment Workflow**:

1. **Dockerize Each Module**:
    - Create separate **Dockerfiles** for:
        - **Frontend** (React.js + Axios)
        - **Backend** (FastAPI/Node.js)
        - **Data Layer** (Python Algorithms)
    - Ensure **cross-container communication** using Docker Compose.
2. **Set Up CI/CD Pipeline**:
    - Use **GitHub Actions** for automated testing and deployment.
    - Trigger builds on new **feature branches** and deploy to the **staging environment**.
3. **Monitoring & Logs**:
    - Implement logging (e.g., using FastAPI's logger).
    - Track errors and user actions (e.g., retrievals, additions).

Final System Integration Workflow

1. **Astronaut interacts with the Frontend** (Adithya’s Module).
2. **Frontend sends requests** to the Backend via REST APIs (Jigme’s Module).
3. **Backend communicates** with the database and the Optimization Layer (Vaibhav’s Module).
4. **Optimized logic** improves storage & retrieval, sending results back to the Backend.
5. **Responses are displayed** on the dashboard in real time.
6. **DevOps** (Rishita’s Module) ensures everything runs smoothly with Docker.