10 drones

- Drone
  - **Serial number**
    - 100 chars max
    - Assume unique
  - **Model**
    - Lightweight
    - Middleweight
    - Cruiserweight
    - Heavyweight
  - **Weight Limit**
    - 500 grams max
    - Assume limit differs with the model
  - **Battery Capacity**
    - Periodic task should check battery level and create a audit log
  - **State**
    - IDLE
    - LOADING
      - If battery level is >= 25%
        - Assume min battery level to be loading can change with delivering distance
    - LOADED
    - DELIVERING
      - If (loading weight + loaded weight) <= weight limit
    - DELIVERED
    - RETURNING

- Medication
  - **Name**
    - Letters, Numbers, -, _
  - **Weight**
  - **Code**
    - Upper case letters, Numbers, _
    - Assume unique
  - **Image**

- Drone Bucket
  - **Drone**
  - **Delivery** **(only if the delivery is implemented, this will be in the drone bucket db table)**
  - **Total Loaded weight**
  - **Item List**

- Delivery (This is the way to preserve delivery details and get **previously delivered item lists**.)
  - **Drone Bucket**
  - **Delivered Date / Time**
  - **Started Longitude**
  - **Started Latitude**
  - **Delivered Longitude**
  - **Delivered Latitude**

- **Starting Battery Level**
- **Ending Battery Level**

- Dispatch Controller
  - Register a drone
  - Loading a drone with medication
  - Checking loaded medication for a given drone
  - Checking available drones for loading (idle drones)
  - Check battery level for a given drone

- Drone Repository functions
  - Get drone by serial number
  - Save drone
  - Get drone by state

- Drone Bucket Repository functions
  - Get drone bucket by drone
  - Add item to drone bucket
  - Get drone bucket item list by drone

- Medication Repository functions
  - Get medication by code

- Development Tools
  - **Spring Boot version 3.0.0**
    - Spring Web
    - Spring Data JPA
    - Spring Boot DevTools
    - H2 Database
  - **Java version 17 (mostly using up-to Java 11 features)**
  - **Domain Driven Architecture**
    - Adapting concepts from the clean architecture
      - Achieving Scalability through interface driven implementation.
      - Achieving Maintainability through model class separation for each context of the architecture (domain context, repository context, web context, etc.).
      - Achieving Reusability by keeping Spring framework, database and other dependencies decoupled from the domain logic as much as possible.
      - Achieving Encapsulation and Domain Purity by encapsulating domain logic and attributes together with only necessary dependencies.

- Development Decision
  - **Not keeping drone's battery level, state in the database** since these are **constantly changing** and keeping them in the database will not be useful.
    - If the application stopped running due to some issue, meaning of these values will be expired anyway. It is really application's job to keep the battery level and state and take necessary actions accordingly.
  - **Keeping drone's active status to filter from active and deactivated drones.**
    - Only active drones are being used to deliver items.

- **Uses Spring framework as an outer layer to keep application domain logic decoupled from the framework.**
- **The folder structure that I'm using is different from the usual 3 layered architecture structure. Trying to separate domain and other resources to gain more clarity over what the project is about through the folder structure. Also, I believe this structure will make it easy when scaling.**
- **Ignoring user authentication and authorization to resources as that is not necessary for the assessment scope.**
- **Introduces an interface to be implemented by all item entities that are capable of loading into drone bucket. This way, if we decide to deliver something other than medication, drone bucket will still support the item as long as it implements the interface DroneBucketItem.**
-