**Design**

**System Design**

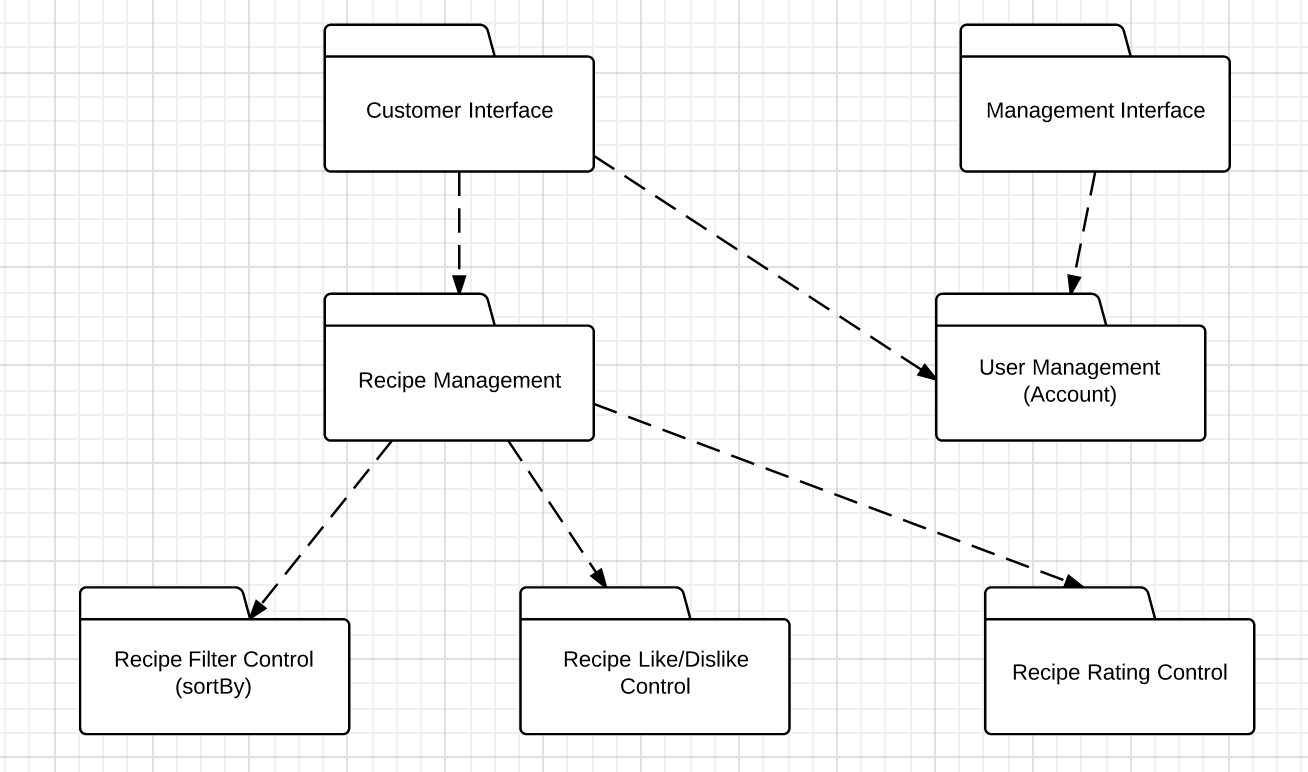
**Subsystem Design**

* Our system will be composed of seven (7) major subsystems being:

1. **Customer Interface**
2. **Management Interface (Administrator and Corporate)**
3. **User Management**
4. **Recipe Management**
5. **Recipe Filter Control (sortBy)**
6. **Recipe Like/Dislike Control**
7. **Recipe Rating Control**

* The approach to designing the seven major subsystems was to break down the subsystems to the lowest level where one subsystem could be implemented by one developer.

**High-Level Subsystem Design**

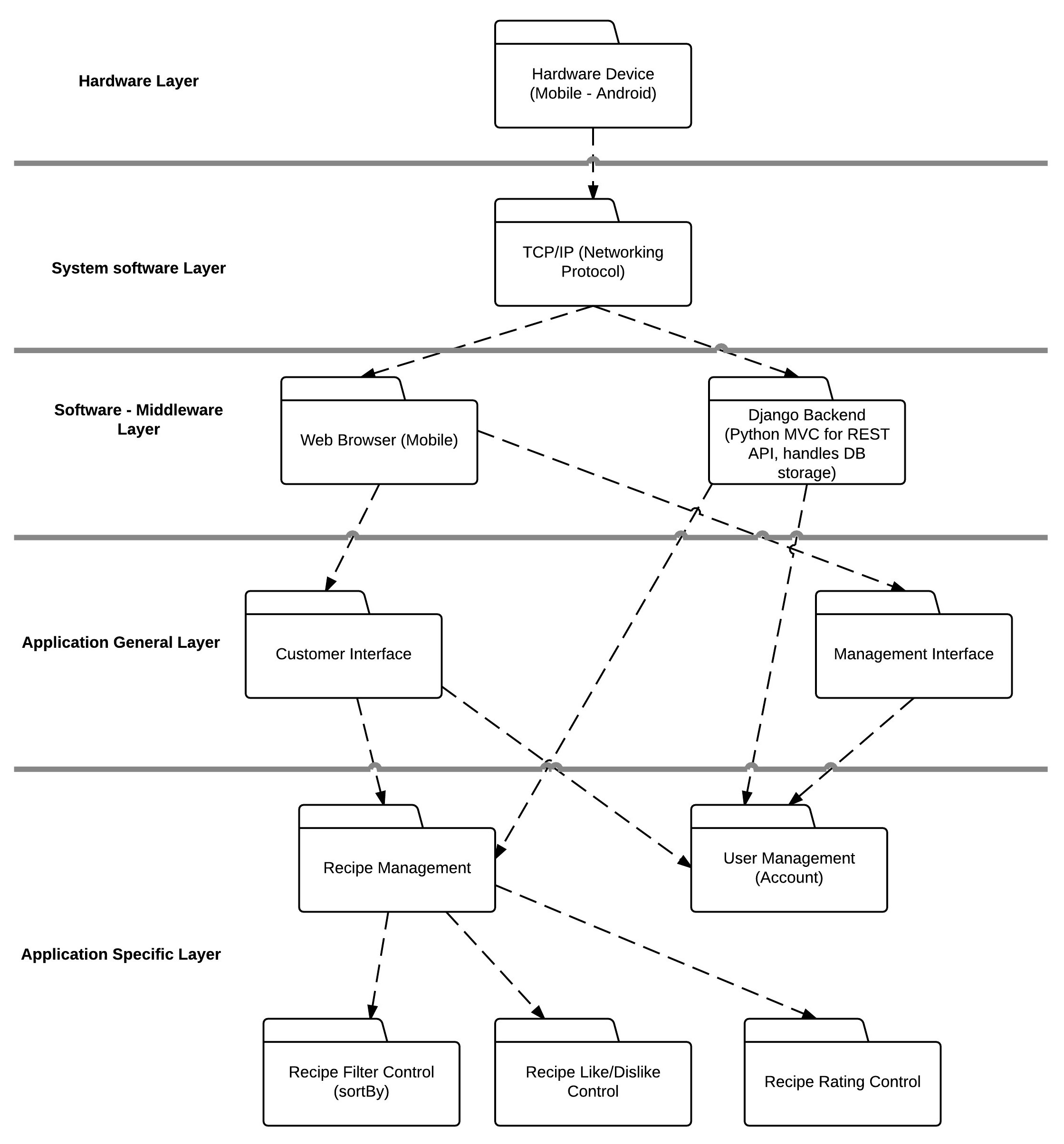


**Subsystem Responsibilities**

1. **Customer Interface**: This subsystem is responsible for the handling all UI design and user interface aspect of the application. Meaning that this interface will complete the link of human-machine interaction (machine being Android or some other mobile device) and will only serve the application’s users. This subsystem will have only features available to users.
2. **Management Interface**: This is the UI and user interface for management and corporate in a real world scenario. This interface provides extended management features such as banning abusive users and deleting recipes that are deemed offensive and abusive in terms of content.
3. **User Management**: This subsystem is responsible for all account related functionality such as editing profile and deactivating your account. This subsystem was also made to be responsible for working parallel with payment management subsystem should that need arise in the future where such a cooking app provides exclusive features who purchase an account subscription.
4. **Recipe Management**: This subsystem is responsible for handling all recipe related functionality such as creating recipes, editing an existing recipe, and deleting a recipe. Other functionality such as rating, like/dislike, and sorting are siphoned off to the subsystems below, as they are major features that require their own subsystems to keep this current one uncluttered.
5. **Recipe Filter Control**: This subsystem is responsible for returning filter/sort by data to the recipe management subsystem based on certain parameters that are predetermined. These parameters being like sort by upload date, popularity, rating, etc.
6. **Recipe Like/Dislike Control**: This subsystem is responsible for handling all the like and dislike related functionality.
7. **Recipe Rating Control**: This subsystem is responsible for handling the rating of a recipe (ex. Rate this recipe 5/5 stars) functionality.

**Dependencies**

* The dependencies are outlined by the **dotted lines** in the high-level subsystem design diagram. The most crucial dependencies in our system design are between **Recipe Management** and **Recipe Filter Control, Recipe Like/Dislike Control, Recipe Rating Control**. The **Recipe Management** subsystem depends on those three subsystems to ensure it has full functionality in the handling of recipes.

**Subsystem to Hardware Mapping**

The diagram depicted above shows the subsystems as layered and mapped to the corresponding **middleware**, **software**, and **hardware**. As per specific subsystem mapping to the hardware the Android hardware will be using TCP/IP networking protocol will be communicating directly with the web browser which will render the front end interface and it will be communicating with our backend (which handles the application specific subsystems) to perform functions such as adding recipes, user logins, and etc.

**Design Rationale**

We as a group agreed on a system design, which is a combination of both layering and partitioning.

We chose layering because we needed hierarchical structure from layering to define parent child relationship for certain classes to avoid class cluttering and over complication. We follow Object Oriented Programming (OOP) so we definitely needed the hierarchical structure. We have an open architecture meaning that some layers will access layers at deeper levels (ex. Django backend [Middleware layer] accesses *Recipe Management* and *User Management* [Application Specific Layer]).

As per partitioning, we chose this because we needed some subsystems to work directly and communicate with other subsystems in that *partition*. For ex. *Recipe Management* subsystem communicates with *User Management* in parallel to determine whether someone has the sufficient permissions to post a recipe or edit a recipe (we don’t want someone editing a recipe that doesn’t belong to them). Thus, the following reasons why we went with a combination of both layering and partitioning when choosing a system design.

**Object Design**