



The notification system implements the **Observer design pattern**, it holds a data structure containing all the notifications that are needed to be displayed in the UI, once the UI initiates a `checkNotifications` request (every 500 msec for example), the `UserService` returns all the notifications back to the UI, simulating a real time response of the server.

Each **Owner cannot appoint itself** (or any other subscriber that is already an owner/manager of this store), this condition is enforced and explained in the relevant appointment use-cases

The `UserController` will allow **user authentication**, but if the authentication process is complex (consists of tokens, etc) it will split up and pass the responsibility to a smaller **Authenticator** entity

Each registered user (subscriber) has a **unique username**

System Administrator (admin) is a kind of subscriber with managing permissions, **there has to be at least 1 at all times**

Each **Owner/Manager** object represents an ownership/management of a **single store** by a single user, therefore a **single user can be an owner in some stores and a manager in other stores** with no problem

Each **shopping basket** contains the requested products **from a single store**

Represents the **order history** of the user, each order is created from the current shopping cart, all of its information is **copied** from the baskets once the order system successfully processed it.  
If the order failed it could still be saved in the order history as failed.

Checks the **availability** using the **external services** and the **store package**. Supports **concurrent runtime** meaning customers won't need to wait for each operation of other users before beginning its runtime