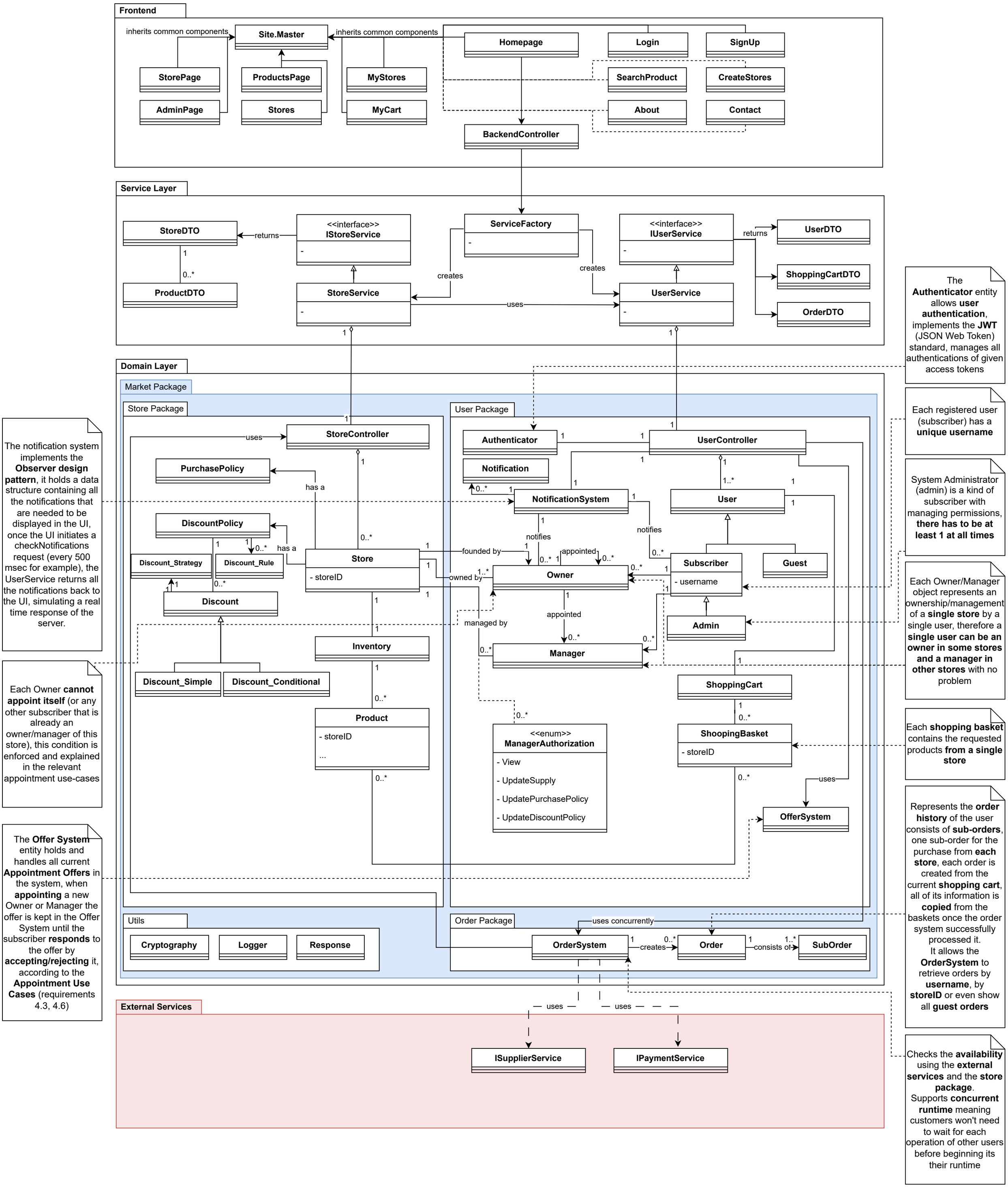
Frontend
inherits common components
Site.Master
inherits common components
Homepage
Login
SignUp
StorePage
ProductsPage
MyStores
SearchProduct
CreateStores
AdminPage
Stores
MyCart
About
Contact
BackendController
Service Layer
StoreDTO
returns
<<interface>>
IStoreService
ServiceFactory
<<interface>>
IUserService
returns
UserDTO
ShoppingCartDTO
OrderDTO
ProductDTO
StoreService
creates
uses
creates
UserService
Domain Layer
Market Package
Store Package
User Package
StoreController
uses
PurchasePolicy
has a
DiscountPolicy
has a
Discount_Strategy
Discount_Rule
Discount
Discount_Simple
Discount_Conditional
Store
- storeID
founded by
owned by
Inventory
Product
- storeID
...
Authenticator
Notification
UserController
NotificationSystem
notifies
notifies
User
Subscriber
- username
Guest
Owner
appointed
appointed
managed by
Admin
Manager
ShoppingCart
ShoopingBasket
- storeID
<<enum>>
ManagerAuthorization
- View
- UpdateSupply
- UpdatePurchasePolicy
- UpdateDiscountPolicy
OfferSystem
uses
Utils
Cryptography
Logger
Response
Order Package
uses concurrently
OrderSystem
creates
Order
consists of
SubOrder
External Services
uses
uses
ISupplierService
IPaymentService
The Authenticator entity allows user authentication, implements the JWT (JSON Web Token) standard, manages all authentications of given access tokens
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 consists of sub-orders, one sub-order for the purchase from each store, 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. It allows the OrderSystem to retrieve orders by username, by storeID or even show all guest orders
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 their runtime
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 Offer System entity holds and handles all current Appointment Offers in the system, when appointing a new Owner or Manager the offer is kept in the Offer System until the subscriber responds to the offer by accepting/rejecting it, according to the Appointment Use Cases (requirements 4.3, 4.6)