Documentation

**Topic**: Project “Warehouse System”

**Made by**: Kaloyan Dimitrov(back-end)

and Slavyan Hristov(front-end)

Summary:

1. *General*

* *Application information*
* *Tools and technologies*
* *Information about starting the application*
* *Project time and work distribution*

1. *System roles and business requirements*

* *System roles*
* *Business Requirements*

1. *Structure and Architecture*

* *Architecture, ER model, UML diagrams*

1. *System Implementation(back-end*)
2. *Tests for some cases*

General

Application information

The application represents a warehouse system which purpose is to ease the process of maintaining warehouses and information about them digitally. The users of this system are: administrator, warehouse owner and warehouse agent.

Tools and technologies

Java, Maven, Hibernate + JPA, JavaFX, MySQL, Log4J2, JUnit Jupiter

Information about starting the application

In order to start the application JavaFX SDK should be installed and added to the path of the project. Also, VM options should be added to the project.

The application uses MySQL driver for database connection. Thus, MySQL server should be ON.

Project time and work distribution

The application was built in teamwork with my co-student and friend during our university semester. The whole project, including documentation and presentation was completed at end of the semester. We worked together on planning, architecting and designing the application. My personal work was implementing the back-end. His personal work was implementing the front-end. My work included working on everything else beside JavaFX.

Roles and business requirements

Roles

1. Administrator

* Login
* Logout
* Create owners and agents
* View, update, delete profiles

1. Warehouse owner

* Login
* Logout
* Create, update, view, delete personal warehoues
* Assign warehouse agents for renting out the warehouse
* View, update personal profile

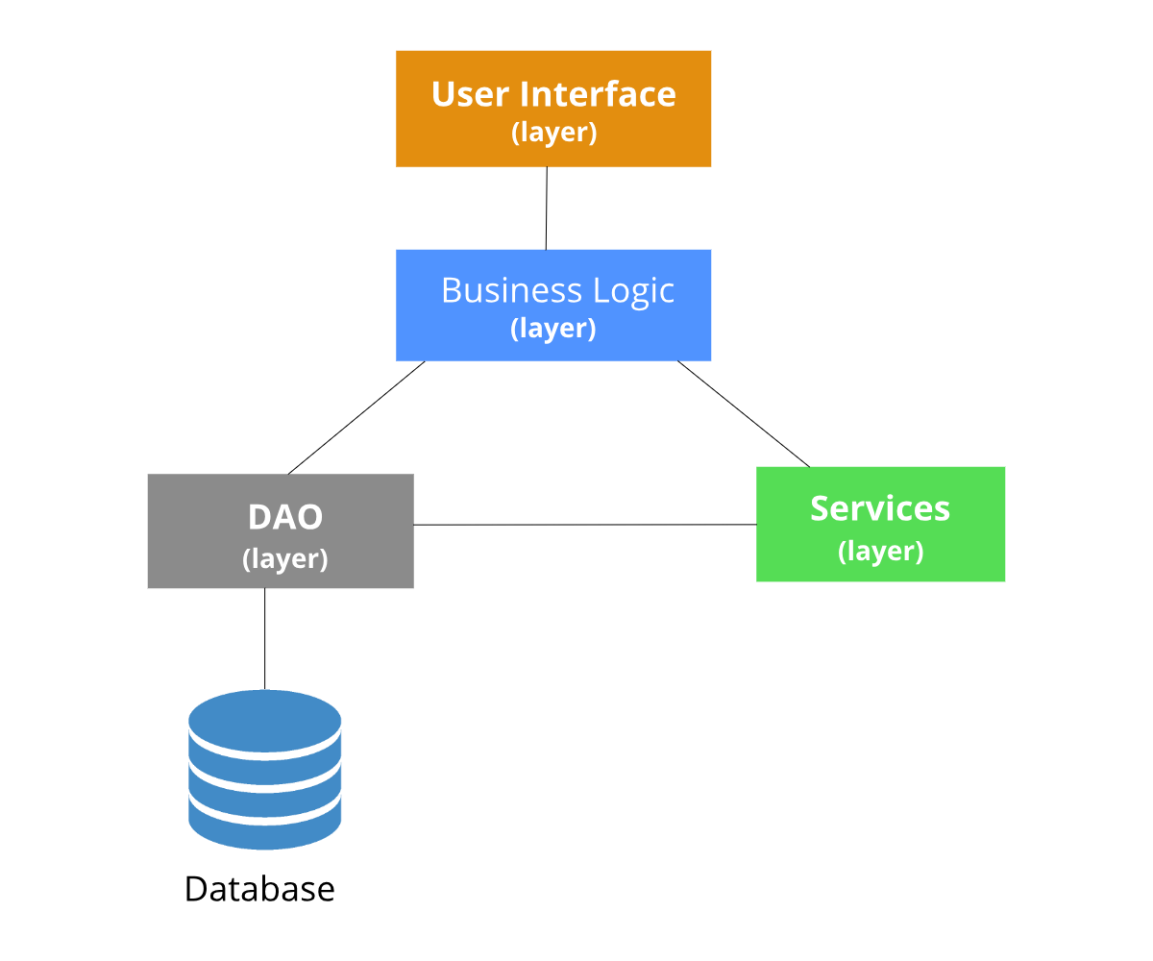
1. Warehouse agent

* Login
* Logout
* Renting out warehouses
* View warehouses
* View, update personal profile

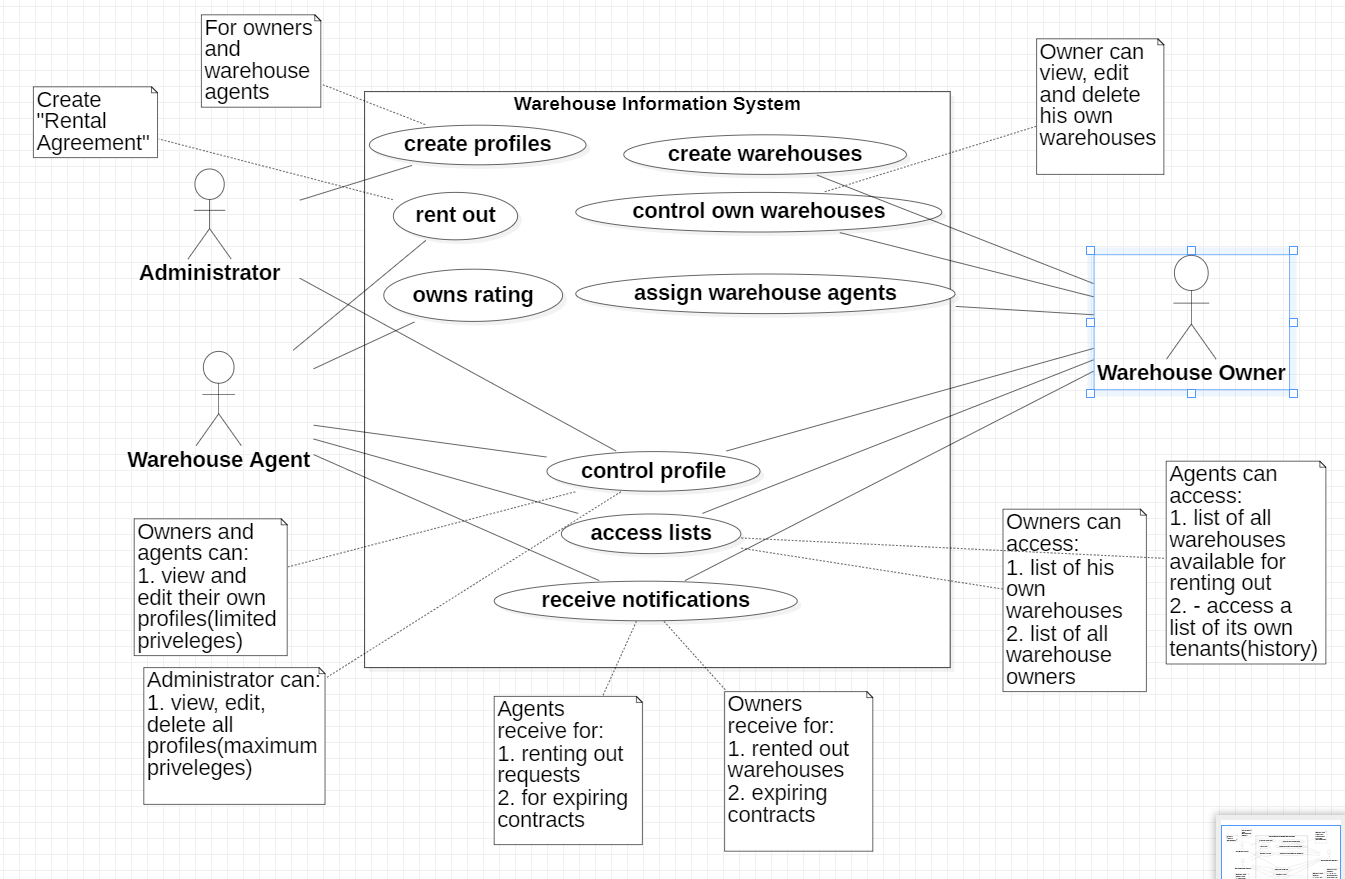
Functional requirements

1. Validation
2. Authentication
3. Data manipulation
4. Logout funtionality
5. Password Encryption
6. View, Create, update, delete functionality

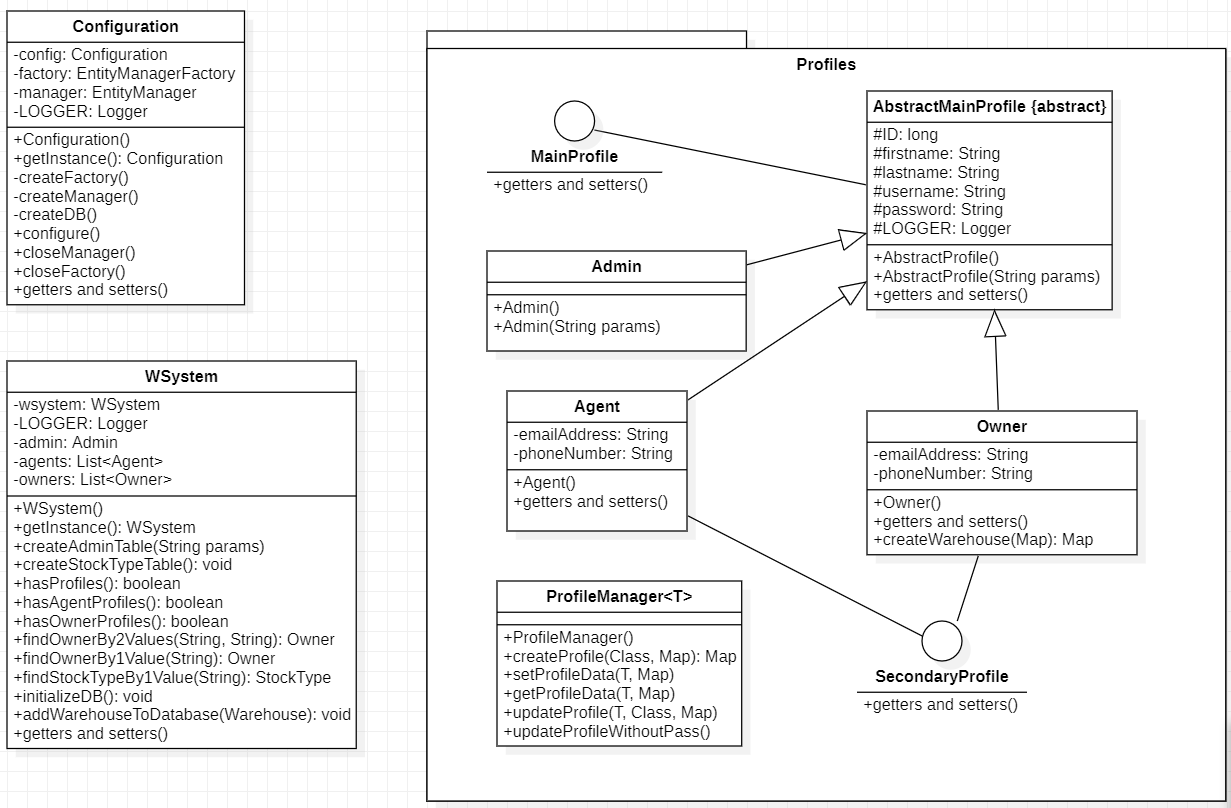
Structure and Architecture

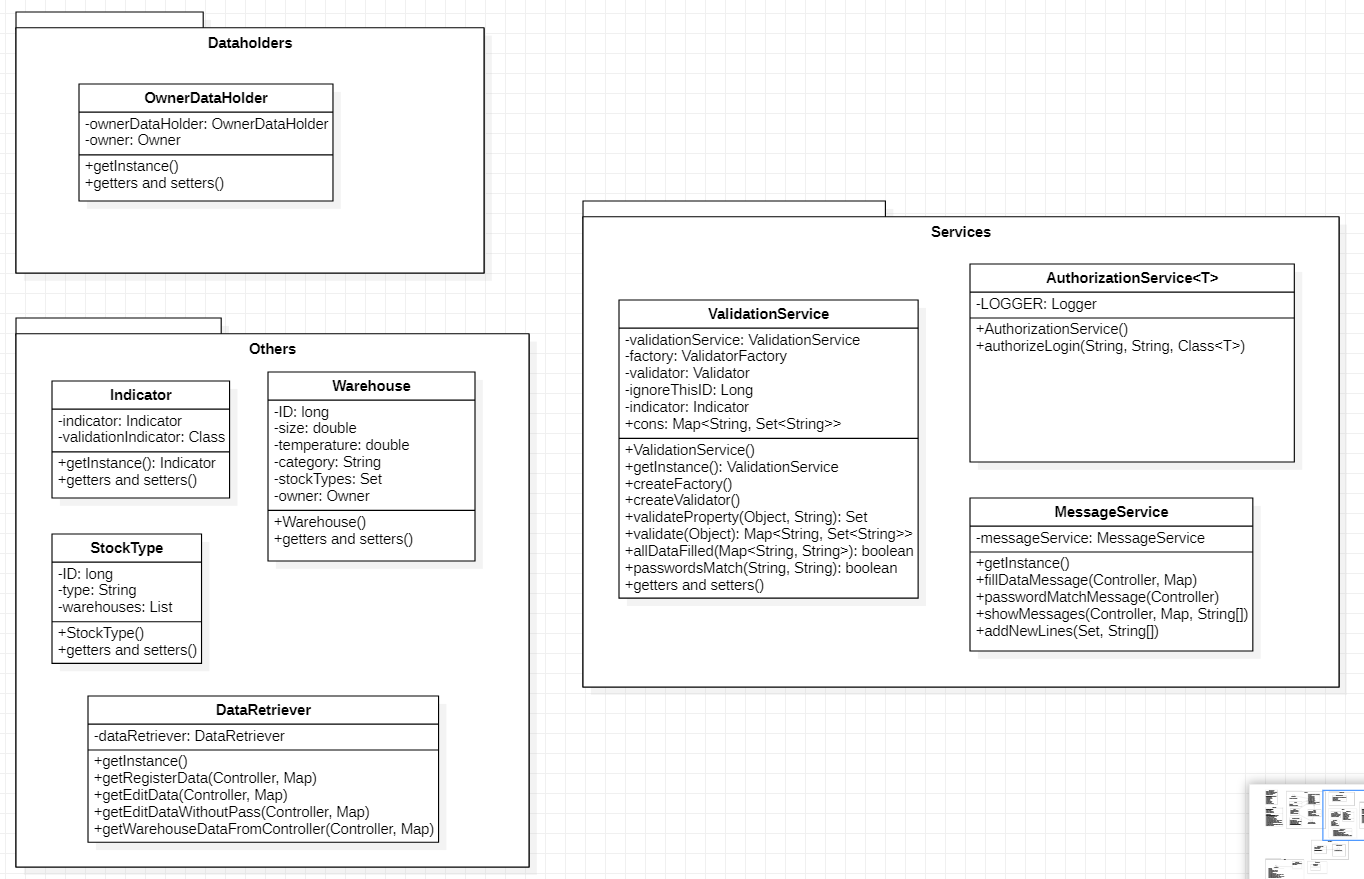


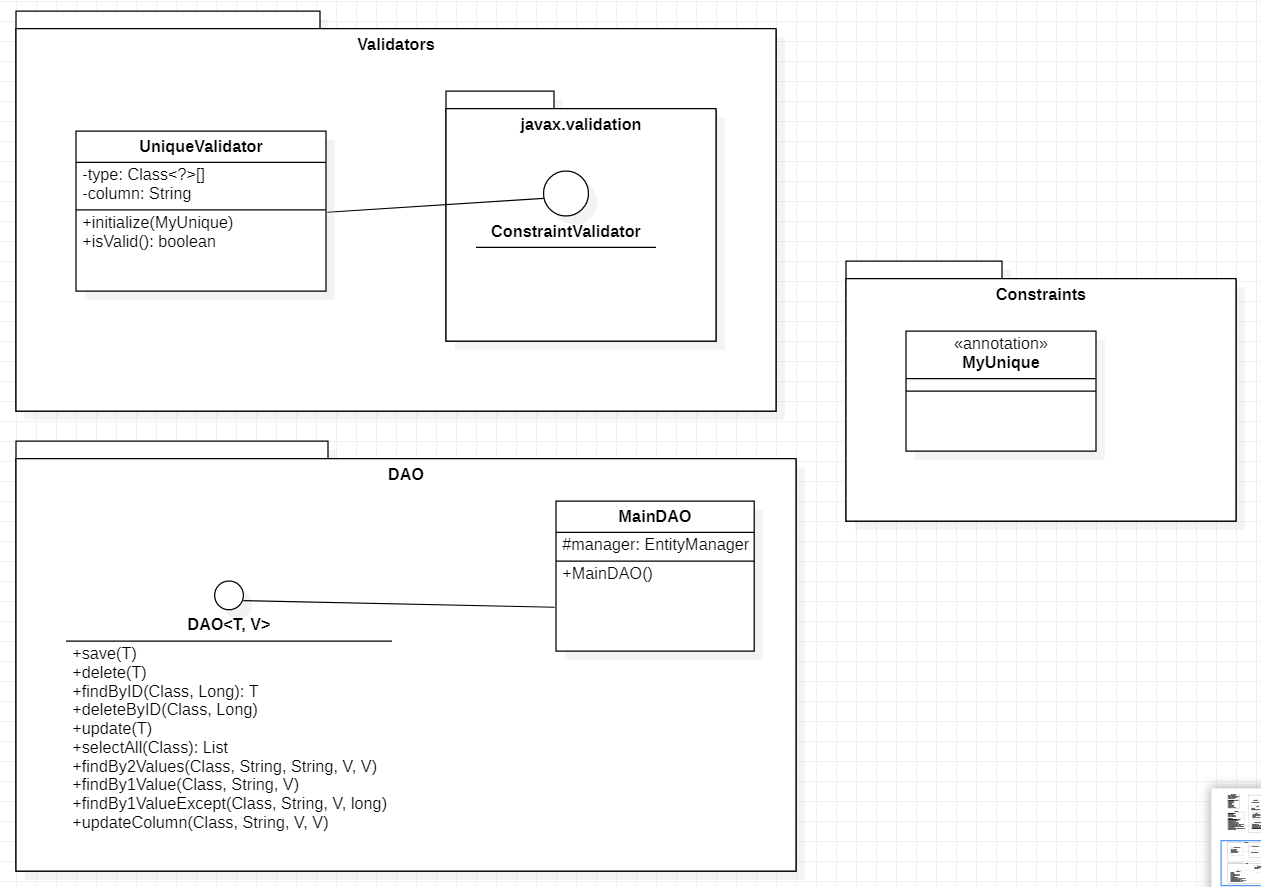
Use Case Diagram



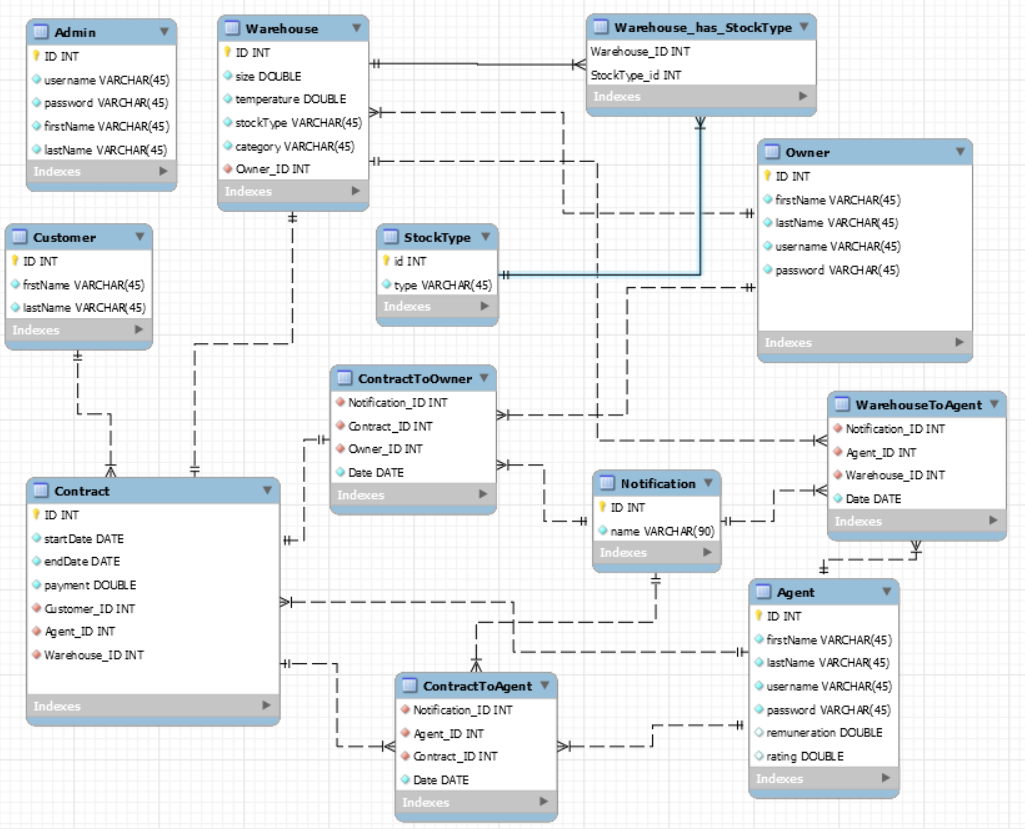
Class Diagram

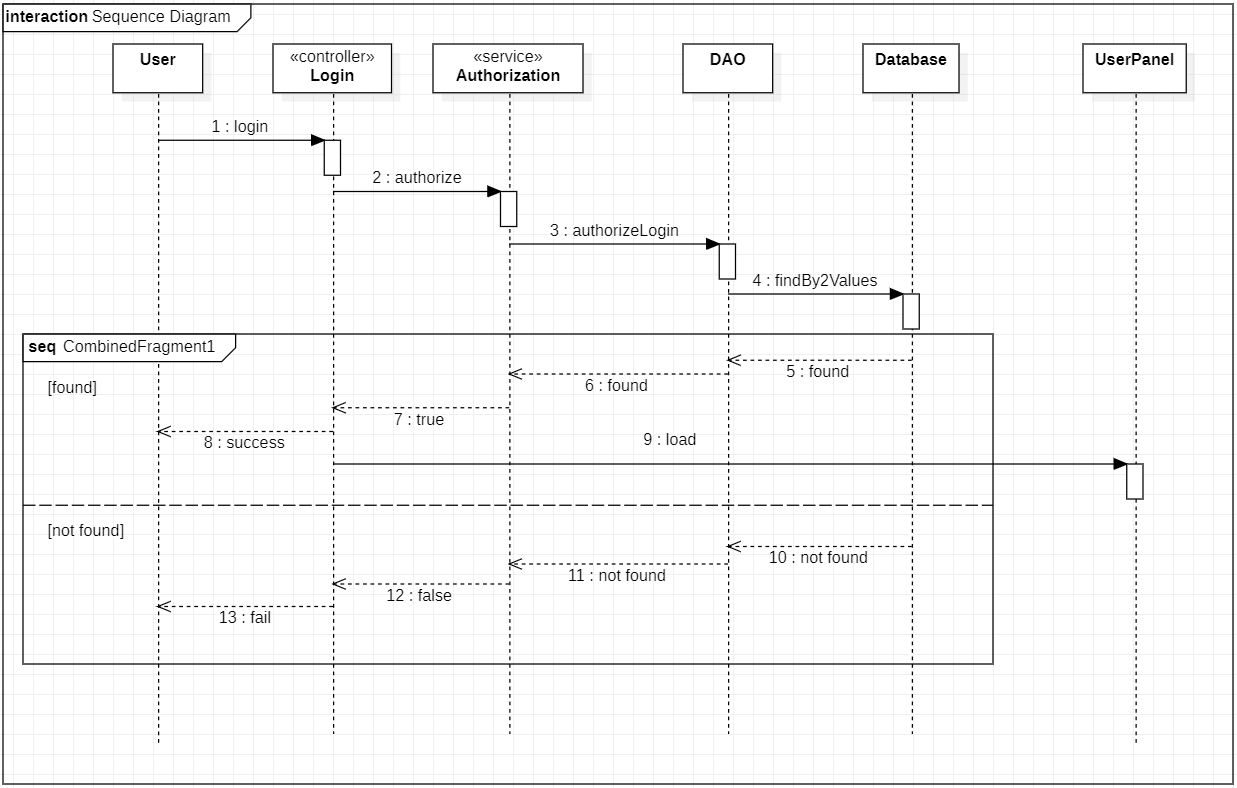


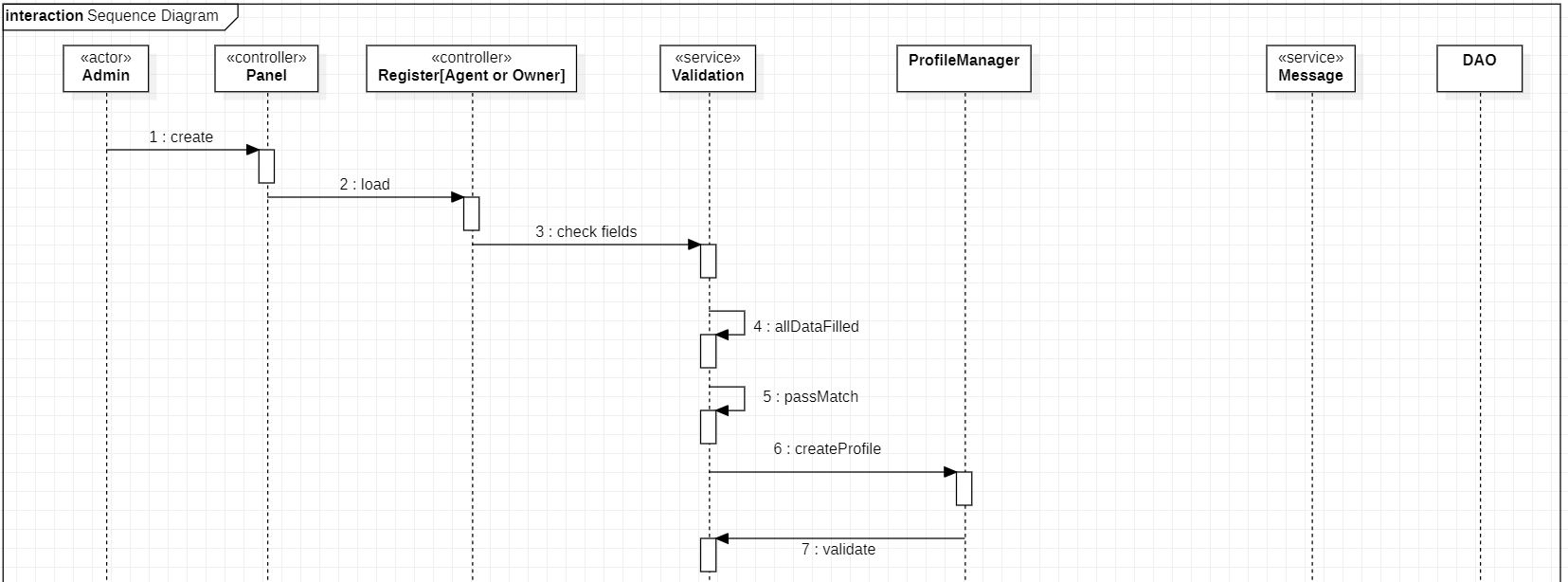
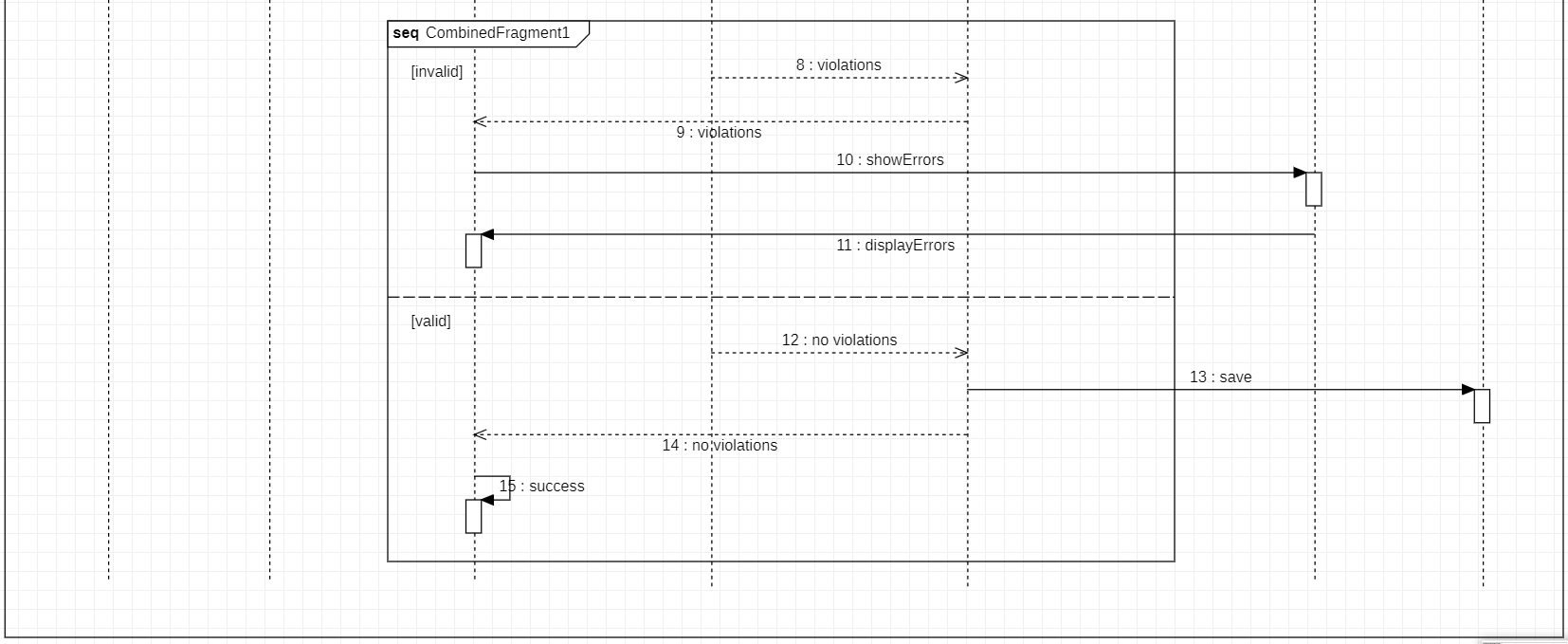


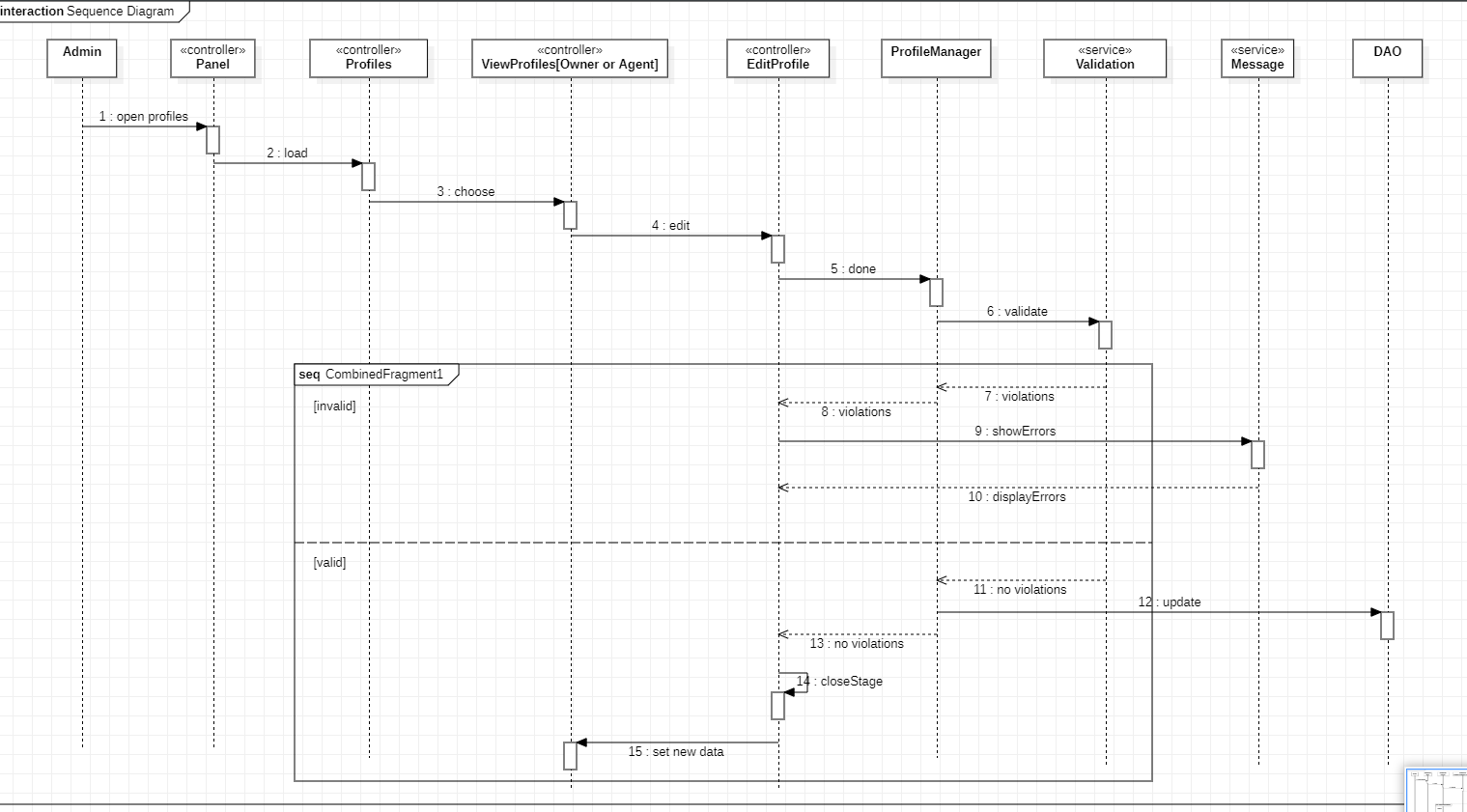


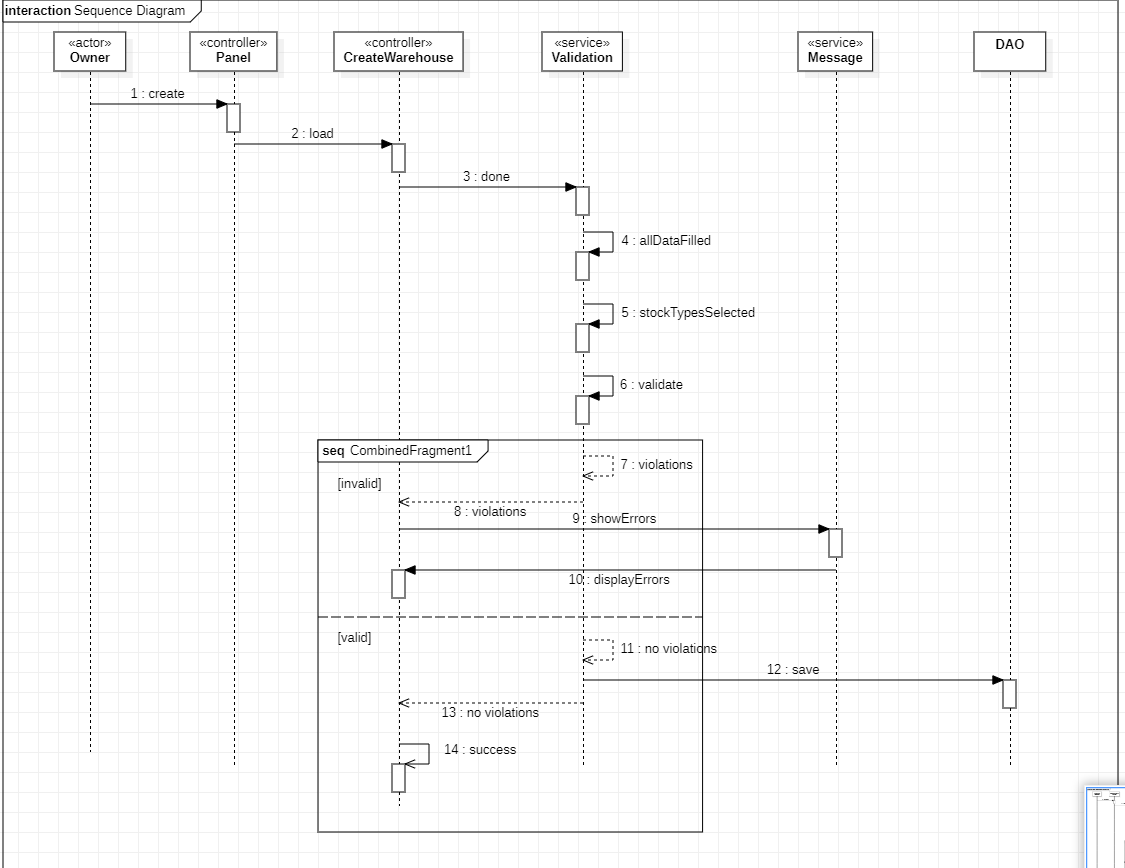
ER Model Diagram



Sequence Diagram for Login Scenario

Sequence Diagram for Register User Scenario

Sequence Diagram for Admin Update Profiles Scenario

Sequence Diagram for Create Warehouse Scenario

Some of the scenarious don’t have a sequence diagram.

System Implementation(back-end)

*Configuration*

The configuration type is XML. Since I am not using the Spring Framework, I had to work with hibernate.cfg for Hibernate configuration and persistance.xml for JPA configuration.

*Explanation of class components and their functionality*

1. *Configuration.class* – Called when the application starts. Creates the database, initializes the schema and creates the system
2. *DAO interface –* this is the interface encapsulating the required functionality of the DAO layer.
3. *class MainDAO<T, V> implements DAO<T, V> -* this is the abstract DAO class encapsulating the general funtionality implemenation and data of the DAO layer. I used “CriteriaQuery API” instead of JPQL and HQL in order to avoid typo mistakes and use Generics
4. *class AuthorizationService<T>* - This class is used for Authentication. In order to avoid duplication, it is from generic type
5. *Class ValidationService* – This class is used for validation during registration or creating and updating a profile. There are constraints which have to be considered.
6. *class CryptoService* – Used for encryption of the password with AES
7. *interface MainProfile –* Consists of getters and setters for some of the profile data.
8. *interface SecondaryProfile –* Consists of getters and setters for some of the profile data
9. *abstract class AbstractMainProfile implements MainProfile –* Consists of the general data and funtionality implementation of every profile
10. *class Owner extends AbstractMainProfile implements SecondaryProfile –* This class represents the owner of the warehouse

*@OneToMany(mappedBy = "owner", cascade = CascadeType.ALL)*

*private List<Warehouse> warehouses = new ArrayList<>();*

The owner has many warehouses. One-to-many relationship is used.

1. *class Agent extends AbstractMainProfile implements SecondaryProfile –* this class represents the warehouse agent
2. *class Admin extends AbstractMainProfile –* This class represents the administrator
3. *class ProfileManager<T> -* This class is used for operations with profiles. In order to avoid code duplicate, a generic one is used.
4. *@interface MyUnique –* defines a custom annotation for unique constraint used during the validation
5. *class UniqueValidator implements ConstraintValidator<MyUnique, String> -* This is the validator class which is needed for performing the unique constraint validation
6. *class Warehouse –* This class represents the warehouse

*@ManyToMany(cascade = CascadeType.PERSIST)*

*@JoinTable(name = "Warehouse\_StockType",*

*joinColumns = {@JoinColumn(name = "warehouse\_id")},*

*inverseJoinColumns = {@JoinColumn(name = "stockType\_id")})*

*private Set<StockType> stockTypes;*

One warehouse can have many stocktypes and one stocktype can exist in many warehouses, so the relationship is many-to-many.

1. *class StockType –* This is the class for every stock type
2. *class Wsystem –* This is the class for the system(singleton)
3. *class MessageService –* This is the class used for showing the errors during the validation process

Tests for some of the cases

*А)* Test for registration of user by admin: registerUserTest()

Б) Test for updating a profile by admin: adminUpdateProfilesTest()

В) Test for updating a warehouse by owner: updateWarehouseTest()