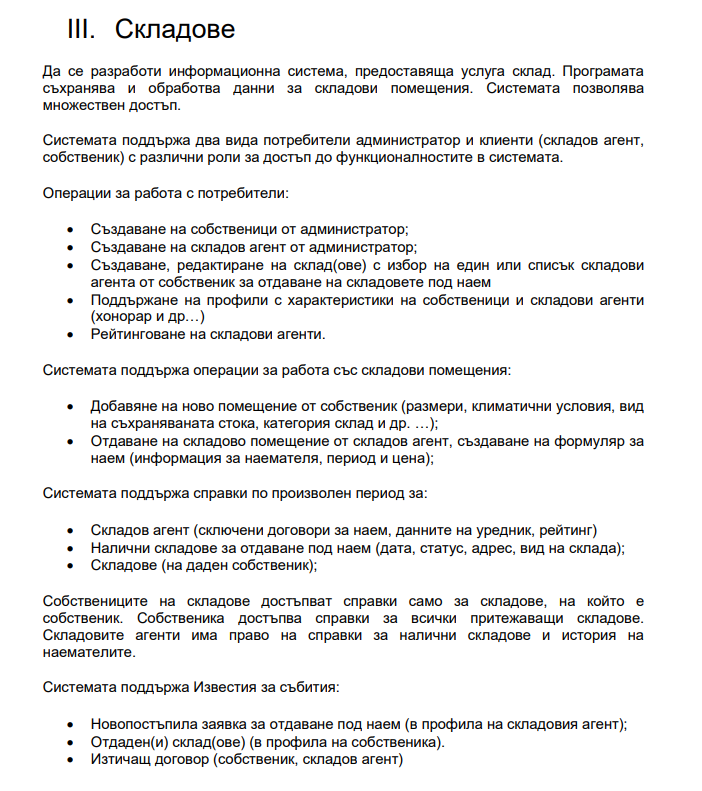
Курсов проект ООП2

Тема: Информационна система, предоставяща услуга склад.

Условие на проекта:

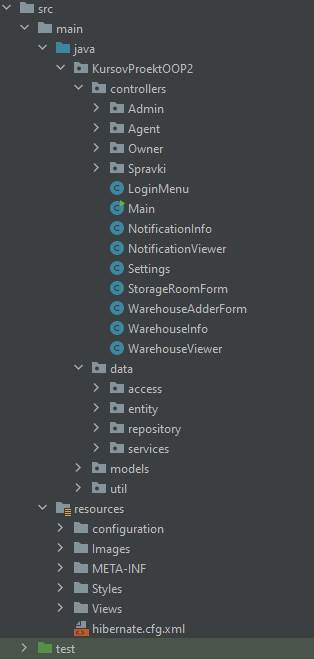


I. Анализ на проблема

Ii. Функционални изисквания

1. Логин меню за всички потребители  
2. Администратор:  
 2.1. Има достъп до всичките функционалности на приложението.  
 2.2. Има меню за регистрация на нови потребители.  
 2.3. Има допълнително меню за преглеждане и поддържане на профилите на другите потребители.  
3. Всеобщи функционалности  
 3.1. Всеки потребител при първо влизане в системата получава известия за възможност за смяна на паролата.  
 3.2. Всеки потребител има опция за смяна на паролата.  
 3.3. Потребители нямат право да си сменят ролите.  
 3.4. Всеки потребител може да прави справка за складов агент (сключени договори за наем, рейтинг)  
 3.5. Всеки потребител може да прави справка за складовете на даден собственик (тяхно местоположение)  
4. Клиент с роля "Собственик на склад"  
 4.1. Може да отдава складови помещения от даден склад като създава списък със складови агенти, които могат да отдават помещенията под наем.  
 4.2. Може да добавя нови помещения в склад като задава техните характеристики.  
 4.3. Вижда детайлна информация за собствените си складове и стаите в тях.  
 4.4. Може да дава рейтинг на складови агенти.  
5. Клиент с роля "Складов агент"  
 5.1. Може да отдава складови помещения под наем, ако е добавен като агент за дадения склад (Създава и попълва формуляр с информацията на наемателя)  
 5.2. Може да преглежда история на наема на дадено помещение  
 5.3. Може да прави справка по произволен период за налични складове за отдаване под наем, в които е добавен той.  
6. Системата изпраща известие към даден собственик за отдадена стая от склад след финализиране на даден формуляр.  
7. Системата изпраща известие към даден складов агент при новопостъпила заявка за отдаване под наем.  
8. Системата при всяко влизане проверява за изтичане на даден договор и изпраща известие към профилите на складовия агент отговарящ за дадения склад и неговия собственик.

Iii. Структура на проекта



Проектът е разделен на четири главни пакета.

Controllers съдържа контролерите на всички view-та, разделен е 4 подпакета, които съдържат специфични за ролята контролери + справките.

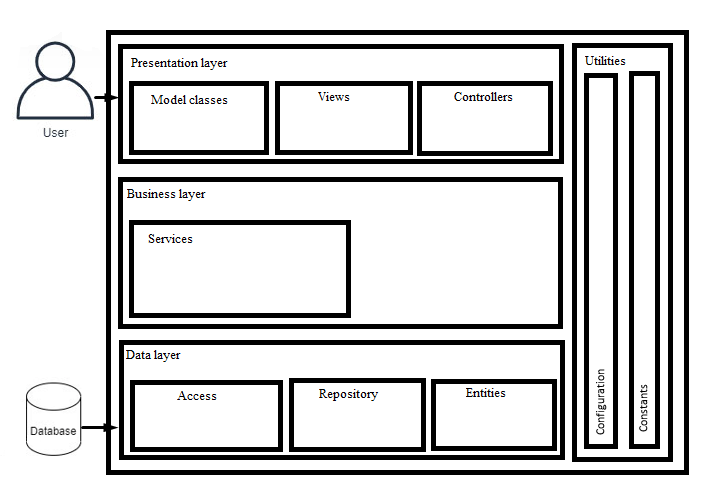
Data съдържа също 4 подпакета, в които се съхранява класът за връзка с базата от данни. Entity - та, техните репозиторита и сървиси.

Models съдържа класове, чрез които се представя информацията в DataTable-ите на справките. Останалите модели се съхраняват в пакета с контролери.

Util съдържа класове с общо предназначение.

Отделно имаме две папки, една предназначена за ресурси, в нея се съхраняват css, fxml, png, jpeg файлове и една папка за тестове.

Iiii. Дефиниция на модулите на системата



Три слойна архитектура.

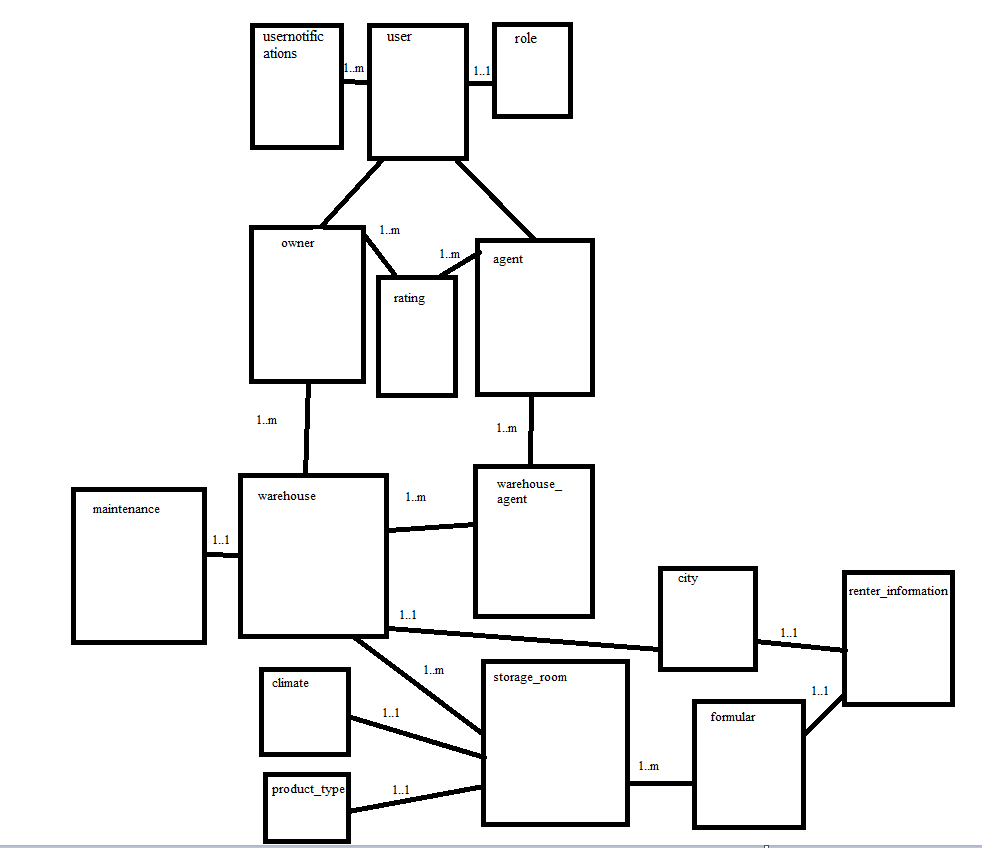
Презентационен слой, чрез който потребителят работи с приложението.

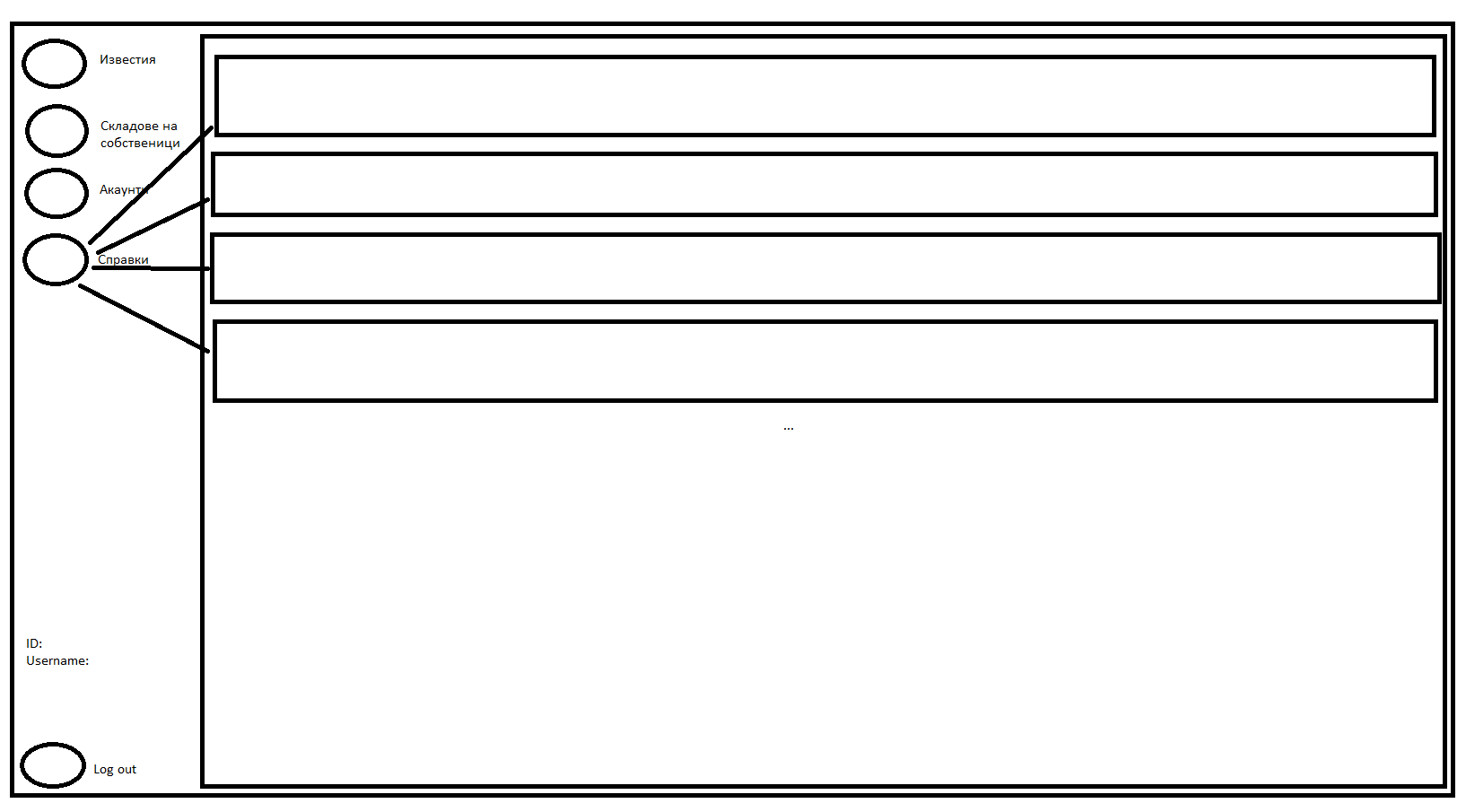
Даннов слой, чрез който базата взема и записва информацията.

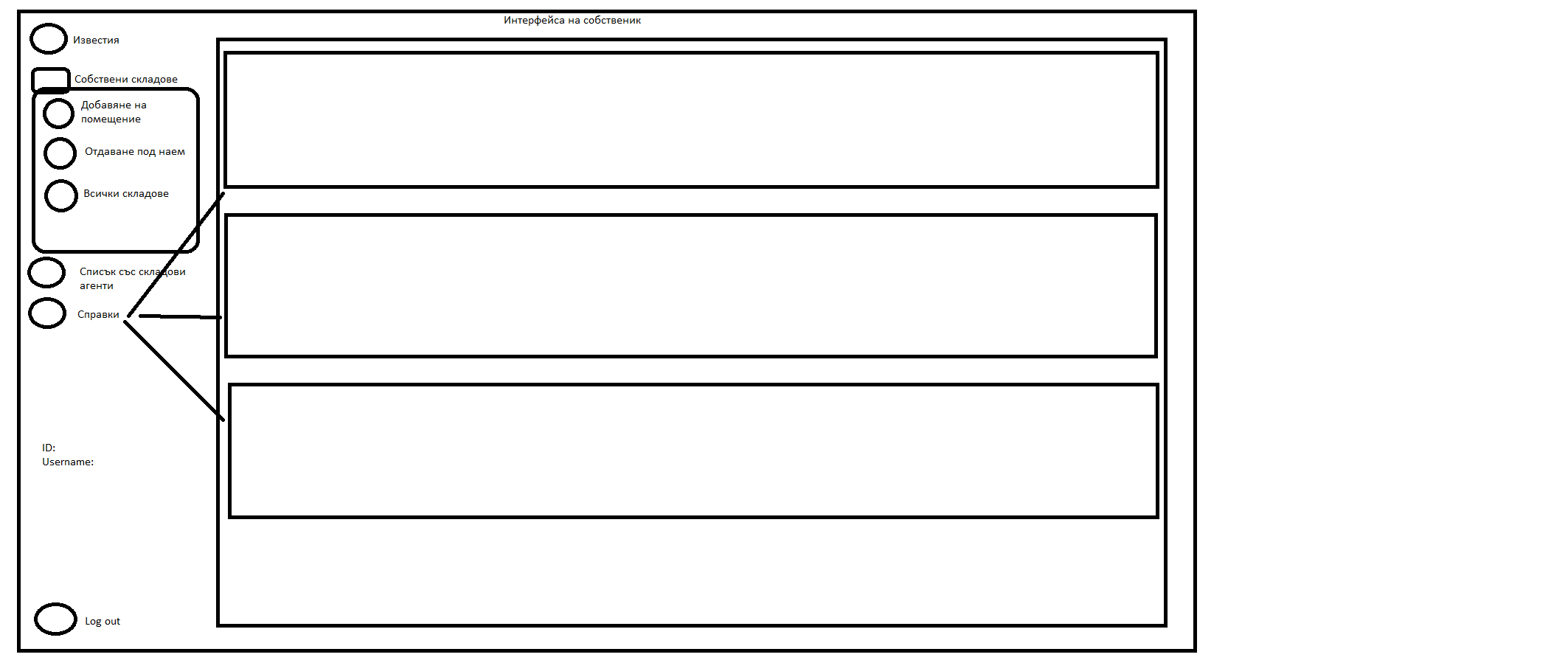
Бизнес слой, който свърза по-горните два слоя.

II. Проектиране на системата

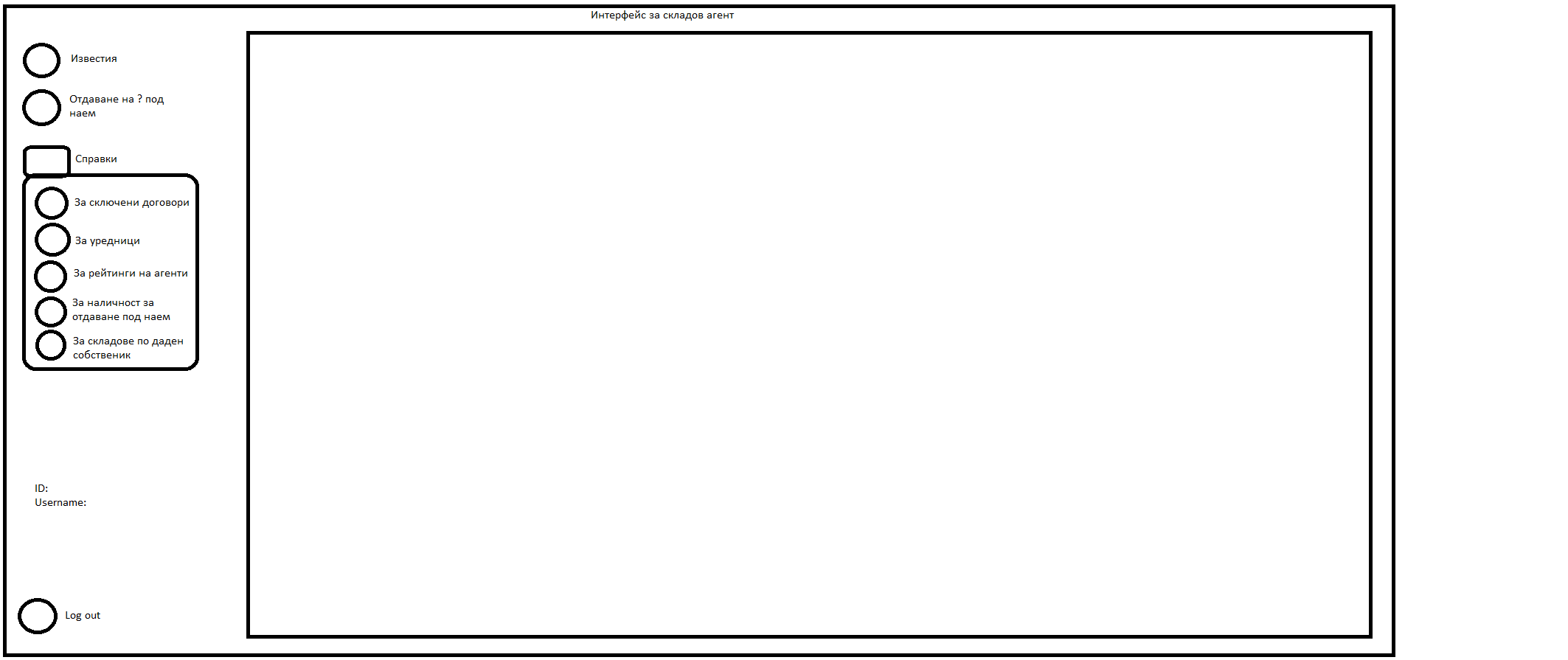
IIi. Проектиране на отделните модули

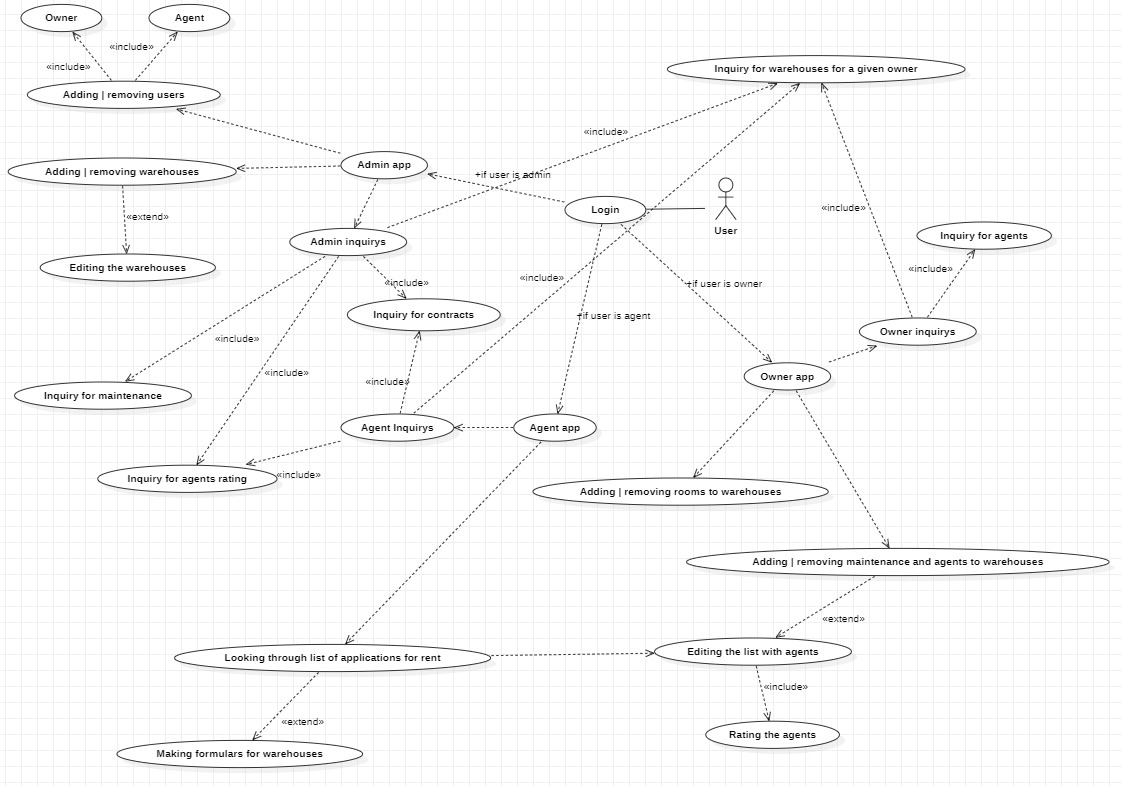
База данни:

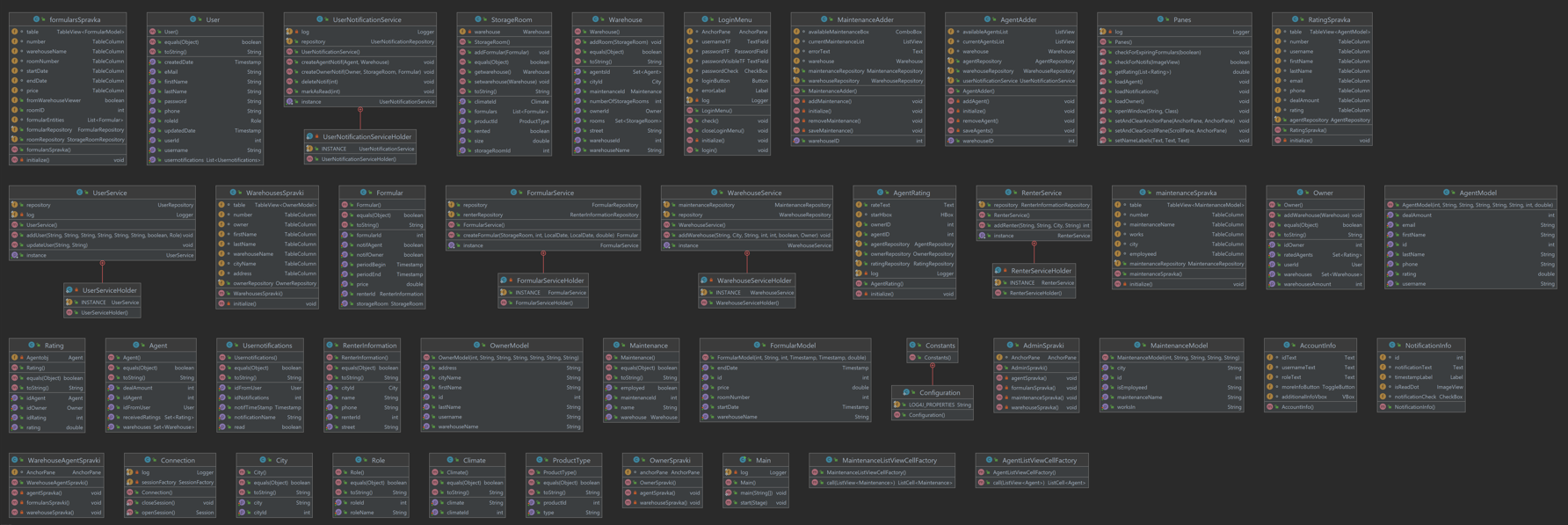
Интерфейс на админ:

Интерфейс на собственик:

Интерфейс на складов агент:



IIii. UML – Use case, Class Diagram, Sequence Diagram

Class Diagram

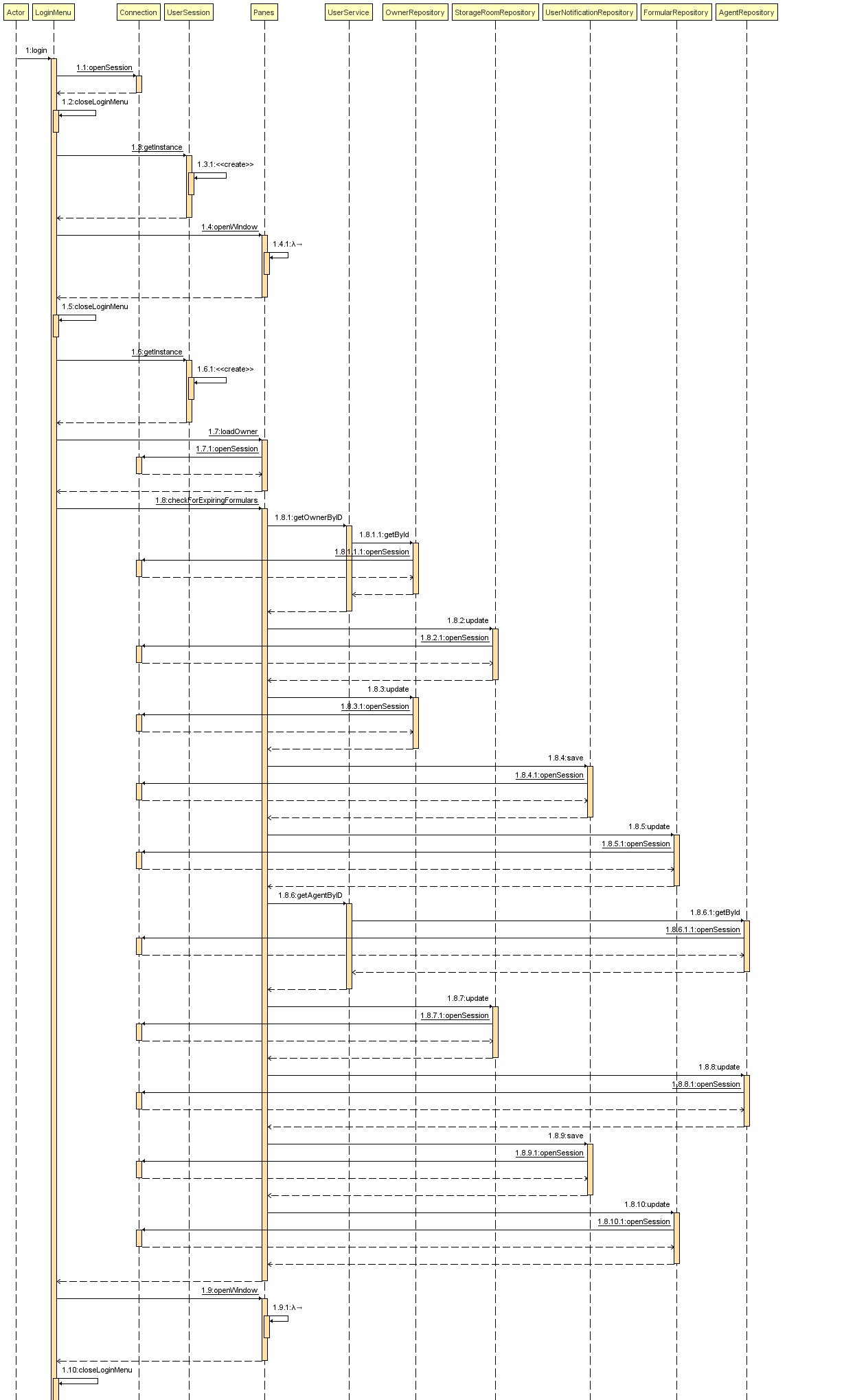
Use case диаграмата показва общите функционалности на програмата. Всеки потребител първо се логва, според неговата роля се отваря съответния view. Повечето справки се споделят между ролите.

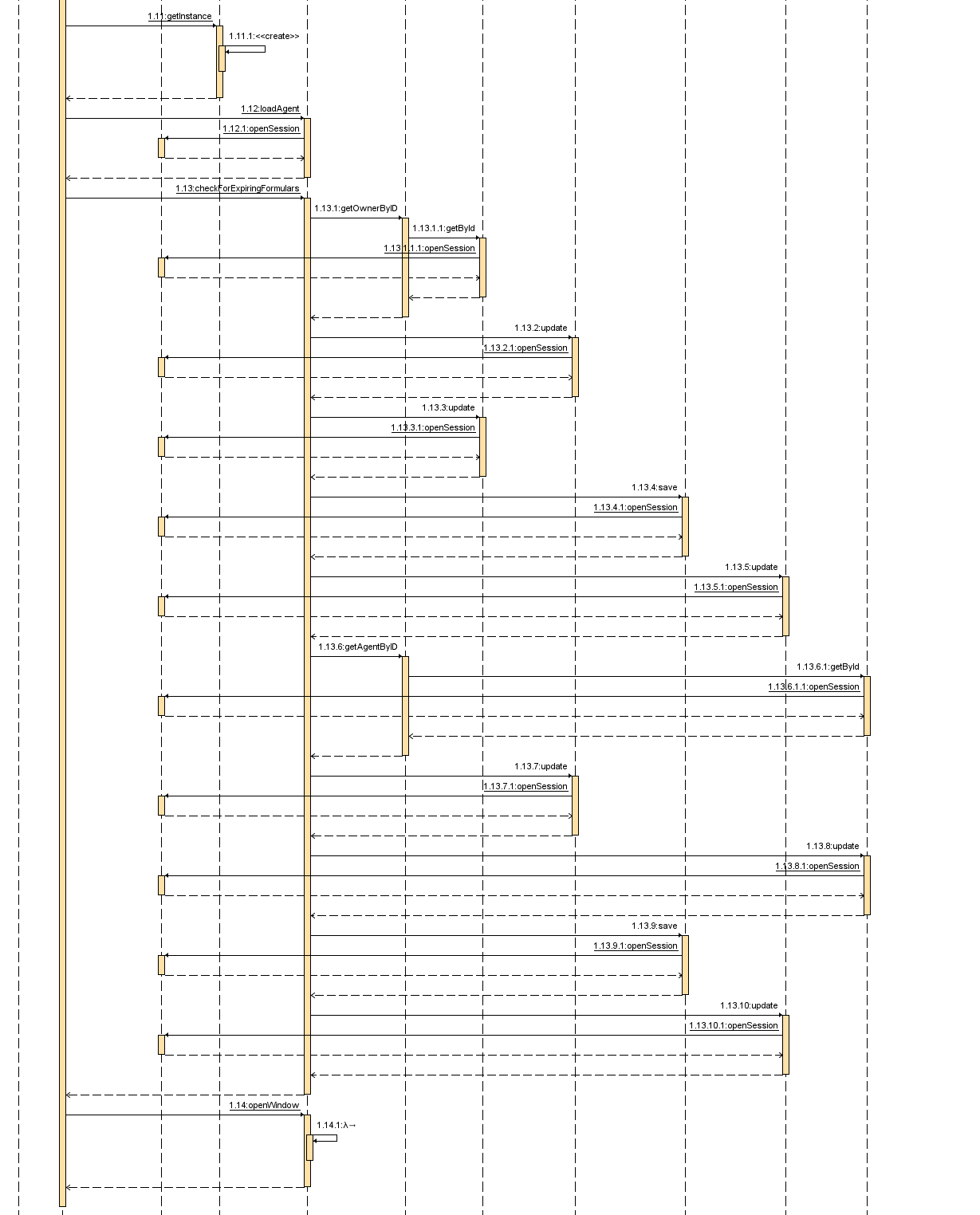
Админът има възможността да добавя и изтрива потребители и складове.

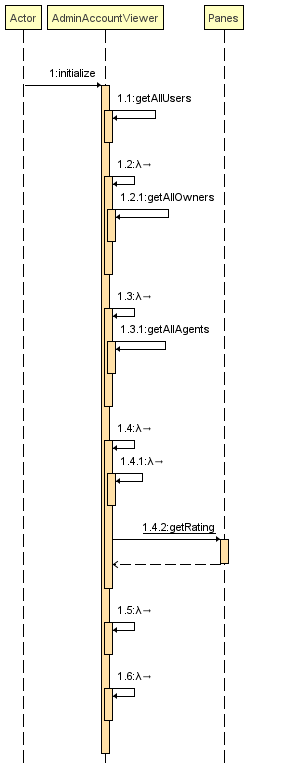
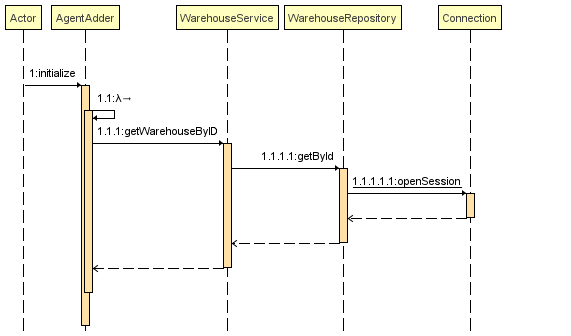
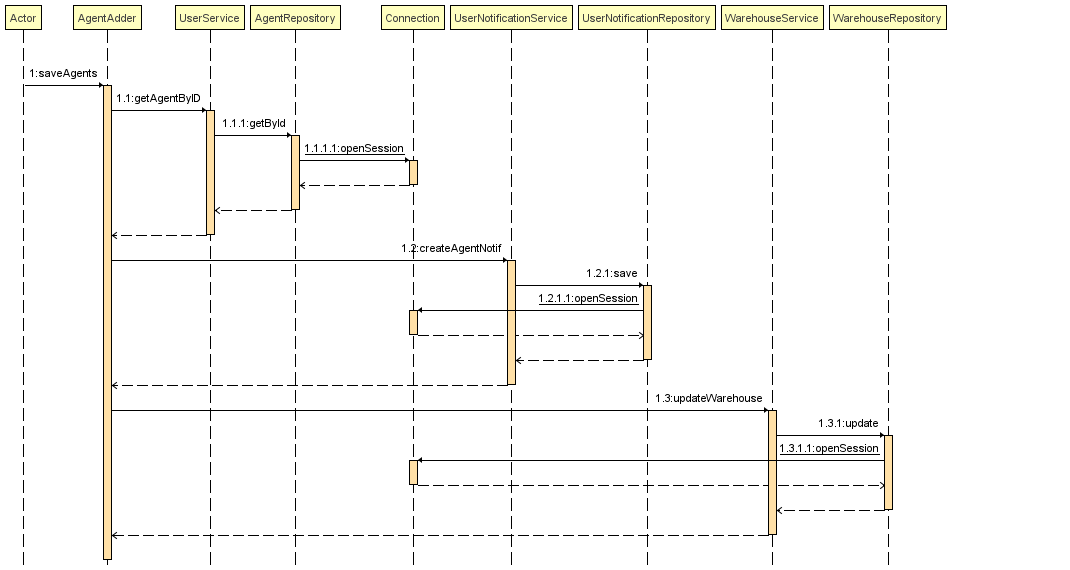
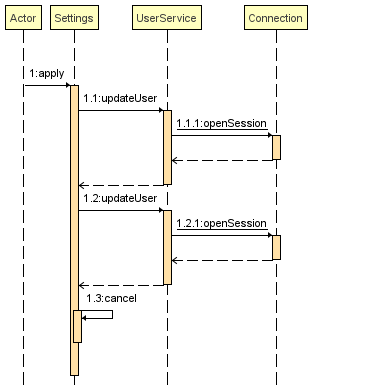
Собственикът може да добавя и премахва стаи в собствените си складове, може да променя подръжката, да редактира списъка с агенти.

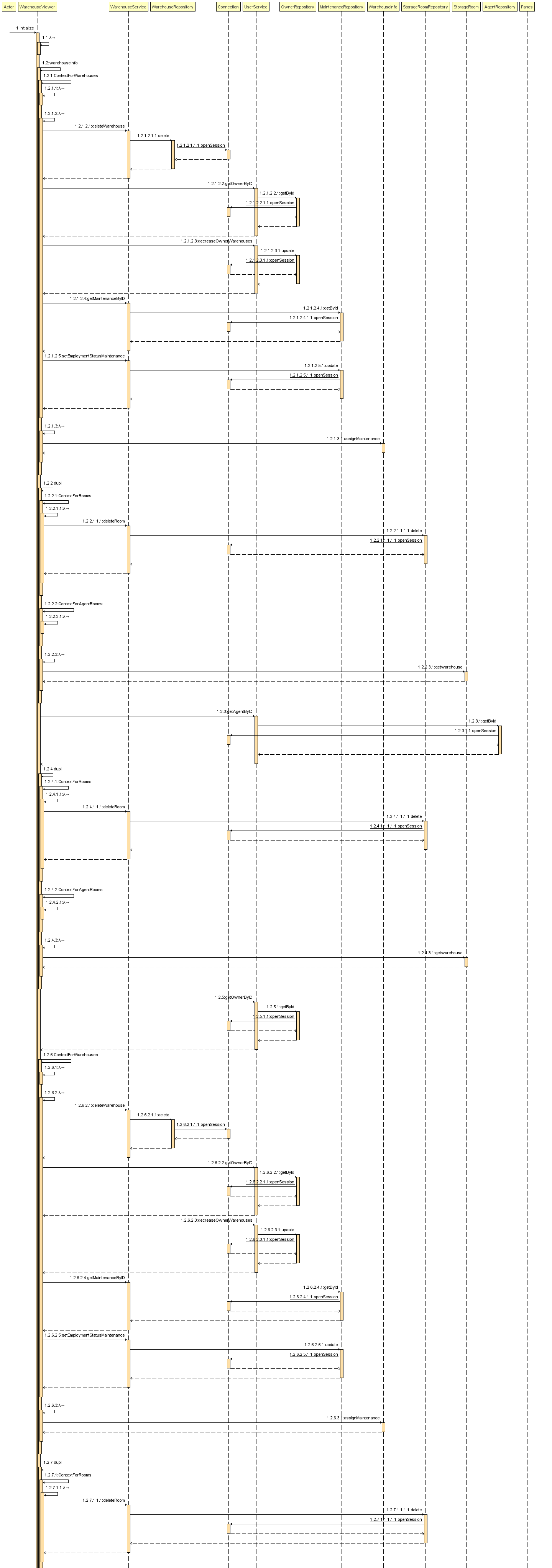
Агентът, може да отдава стаи под наем, като попълва формуляр с информацията на наемателя, може да преглежда историята на наемателите.

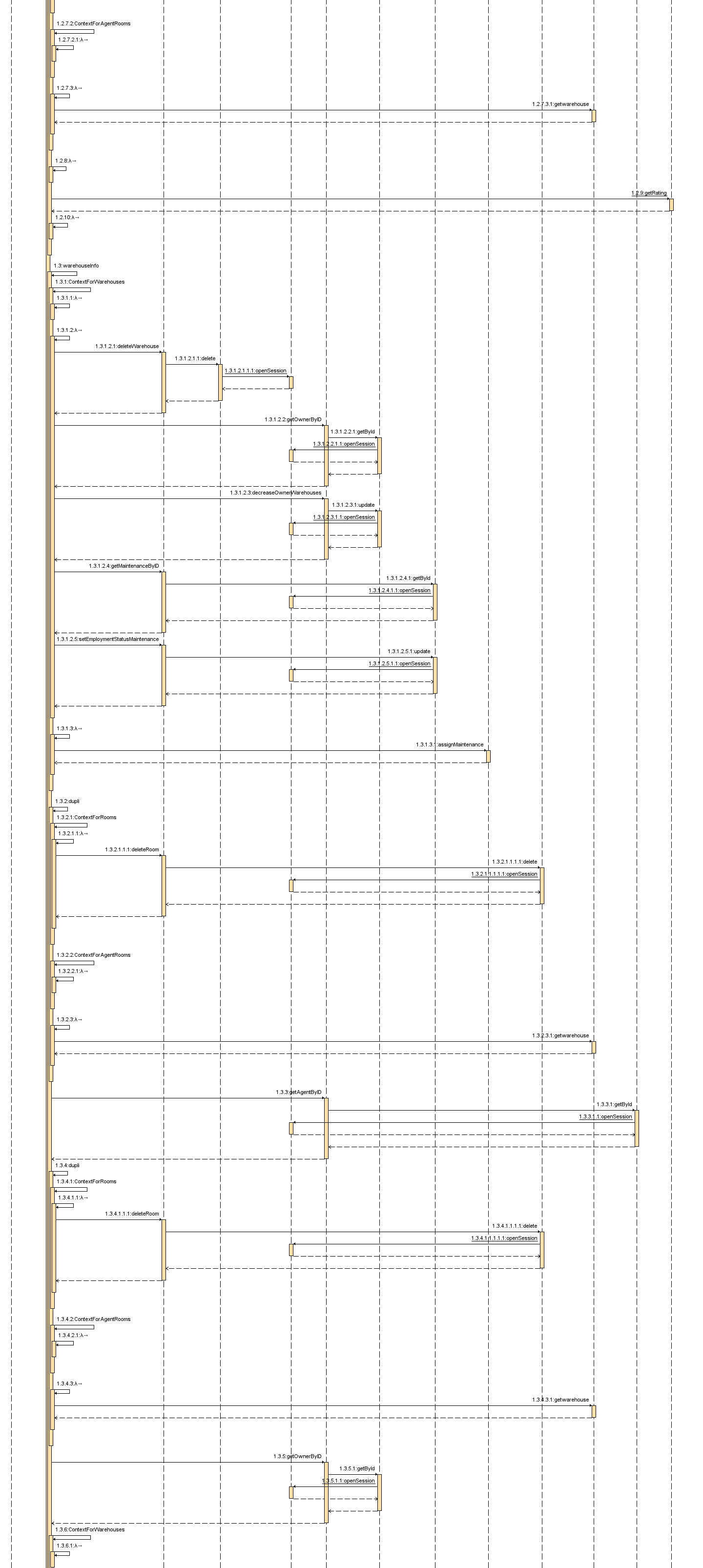
Sequence diagrams

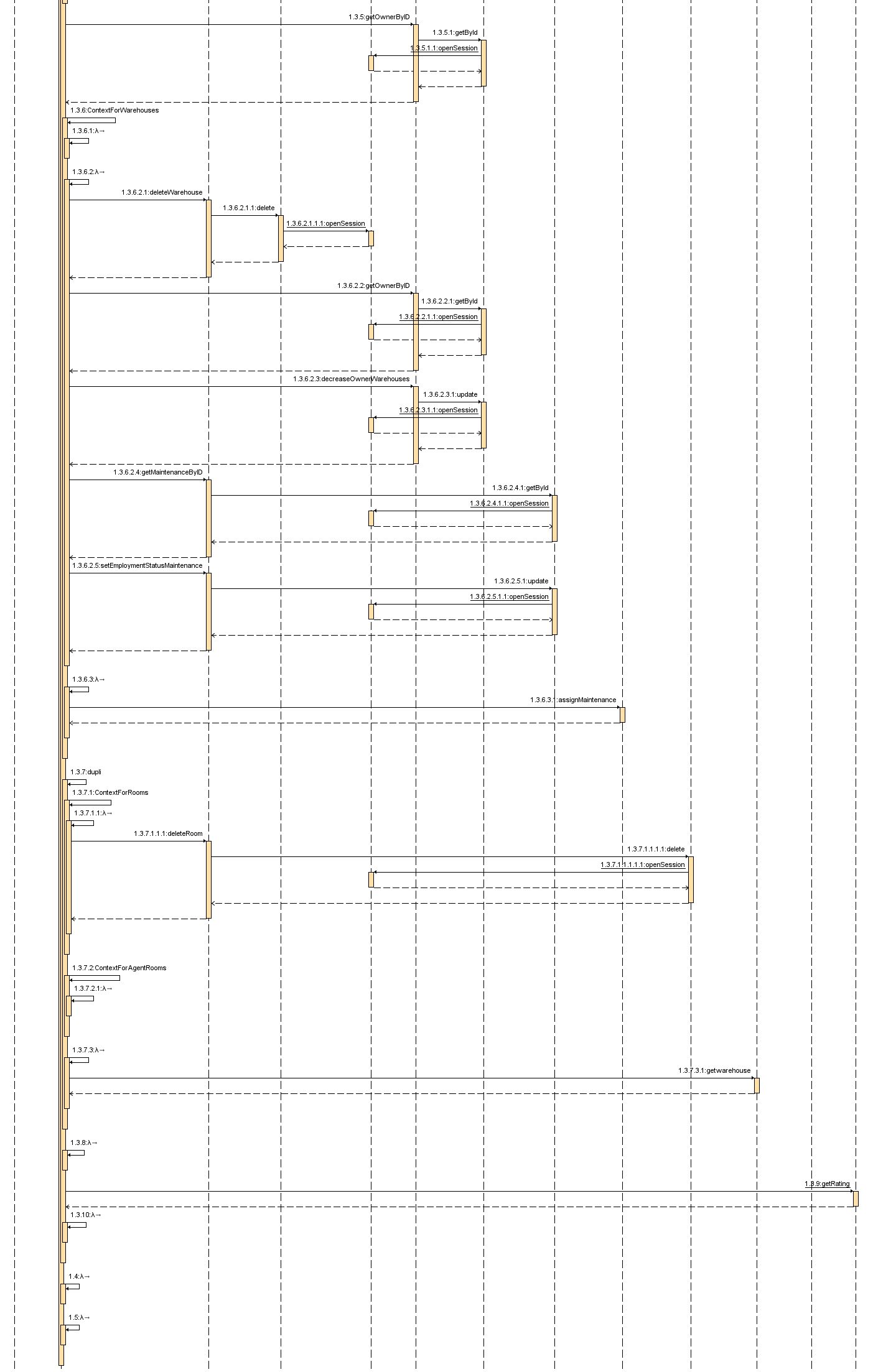




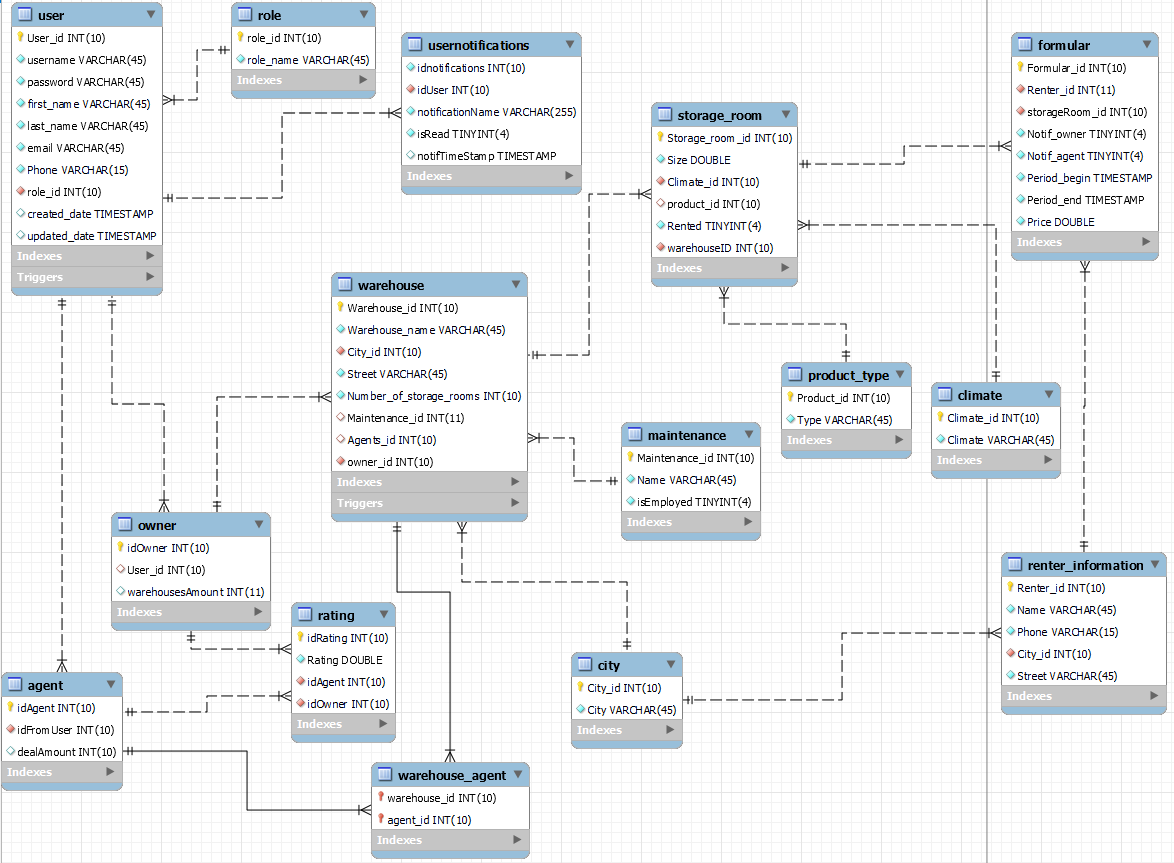








IIiii. ER Diagram



III. Реализация на системата

IIIi. Реализация на базата от данни

В горната ER диаграма са описани таблиците и техните връзки. Главно преобладава

целочисленият тип данна за означаване на primary и foreign keys. Използва се също за следене на броя складове и сделки съответно на собствениците и агентите. В таблицата rating се съхраняват оценките дадени от складовите собственици на агенти. Те са от тип double. Другите два типа данни са VARCHAR ,TINYINT и TIMESTAMP. VARCHAR се изпозлва за съхранението на имена на потребители, наематели, складове на собственици. TINYINT за отблезяване на статут на прочитане на нотификация, дали стая е дадена под наем и дали собственик на склад и агент са осведомени за изтичащ формуляр. TIMESTAMP за съхранение на различни времеви марки (Създаване на профил, последен ъпдейт, начален и краен период на формуляр, време на създаване на нотификация).

CREATE TABLE `warehouse`.`rating` (

`idRating` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`Rating` DOUBLE UNSIGNED NOT NULL,

`idAgent` INT(10) UNSIGNED NOT NULL,

`idOwner` INT(10) UNSIGNED NOT NULL,

PRIMARY KEY (`idRating`));

ALTER TABLE rating ADD UNIQUE (`idAgent`,`idOwner`);

ALTER TABLE `warehouse`.`rating`

ADD CONSTRAINT `fkAgentRating`

FOREIGN KEY (`idAgent`)

REFERENCES `warehouse`.`agent` (`idAgent`)

ON DELETE CASCADE

ON UPDATE NO ACTION,

ADD CONSTRAINT `fkOwnerRating`

FOREIGN KEY (`idOwner`)

REFERENCES `warehouse`.`owner` (`idOwner`)

ON DELETE CASCADE

ON UPDATE NO ACTION;

CREATE TABLE warehouse.warehouse\_agent (

warehouse\_id INT(10) UNSIGNED NOT NULL,

agent\_id INT(10) UNSIGNED NOT NULL,

PRIMARY KEY (warehouse\_id, agent\_id),

INDEX fk\_agent\_id\_idx (agent\_id ASC) ,

CONSTRAINT fk\_warehouse\_id

FOREIGN KEY (warehouse\_id)

REFERENCES warehouse.warehouse (Warehouse\_id)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT fk\_agent\_id

FOREIGN KEY (agent\_id)

REFERENCES warehouse.agent (idAgent)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

IIIii. Реализация на слоя за работа с базата данни

За връзка с базата данни се ползва класът “Connection”, който създава SessionFactory, чрез подаден configuration файл “hibernate.cfg.xml”.

<hibernate-configuration>

<session-factory>

<!-- SQL Dialect -->  
 <property name="dialect">

org.hibernate.dialect.MariaDBDialect

</property>  
 <!-- Database Connection Settings -->  
 <property name="hibernate.connection.driver\_class">

org.mariadb.jdbc.Driver

</property>  
 <property name="connection.url">

jdbc:mariadb://localhost:3306/warehouse

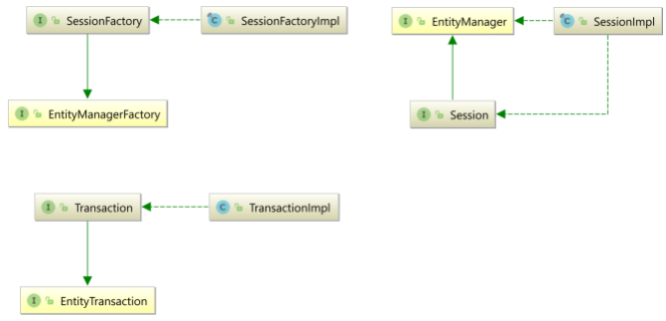
</property>  
 <property name="hibernate.connection.username">root</property>  
 <property name="hibernate.connection.password"></property>  
 <property name="show\_sql">true</property>  
  
 <!-- Specifying Session Context -->  
 <property name="hibernate.current\_session\_context\_class">  
 org.hibernate.context.internal.ThreadLocalSessionContext  
 </property>  
 <!-- Mapping With Model Class Containing Annotations -->  
 <mapping class="KursovProektOOP2.data.entity.Agent"/>  
 <mapping class="KursovProektOOP2.data.entity.Owner"/>  
 <mapping class="KursovProektOOP2.data.entity.City"/>  
 <mapping class="KursovProektOOP2.data.entity.Climate"/>  
 <mapping class="KursovProektOOP2.data.entity.Formular"/>  
 <mapping class="KursovProektOOP2.data.entity.Maintenance"/>  
 <mapping class="KursovProektOOP2.data.entity.ProductType"/>  
 <mapping class="KursovProektOOP2.data.entity.RenterInformation"/>  
 <mapping class="KursovProektOOP2.data.entity.Role"/>  
 <mapping class="KursovProektOOP2.data.entity.StorageRoom"/>  
 <mapping class="KursovProektOOP2.data.entity.User"/>  
 <mapping class="KursovProektOOP2.data.entity.Warehouse"/>  
 <mapping class="KursovProektOOP2.data.entity.Usernotifications"/>  
 <mapping class="KursovProektOOP2.data.entity.Rating"/>

</session-factory>

</hibernate-configuration>

package KursovProektOOP2.data.access;  
  
  
import org.apache.log4j.Logger;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
public class Connection {  
 private static final Logger *log* = Logger.*getLogger*(Connection.class);  
 private static SessionFactory *sessionFactory*;  
  
 static{  
 try{  
 *sessionFactory* = new Configuration().configure().buildSessionFactory();  
 }catch (Throwable ex){  
 *log*.error("Initialization of sessionFactory failed" + ex);  
 }  
 }  
  
 public static Session openSession(){  
 return *sessionFactory*.openSession();  
 }  
  
 public static void closeSession(){  
 *sessionFactory*.close();  
 }  
}

Двата главни метода за връзка с база данни са EntityManagerFactory от Java Persistence API и SessionFactory от Hibernate. И двата метода съдържат нужните метаданни за mapping на entity и позволяват създаването на Session (специфично за Hibernate) и EntityManager. SessionFactory наследява EntityManagerFactory и позволява използването на методите му. Използването на Hibernate specific features намалява портативността на кода.



UPDATE

Session session = Connection.*openSession*();  
Transaction transaction = session.beginTransaction();  
String UPDATE\_QUERY = "UPDATE Usernotifications SET isRead = true WHERE idNotifications = :idNotif";  
try{  
 session.createQuery(UPDATE\_QUERY).setParameter("idNotif", id).executeUpdate();  
}catch (Exception ex){  
 *log*.error("Notifications marking unsuccessful " + "\n" + ex.getMessage());  
}finally {  
 transaction.commit();  
}

DELETE

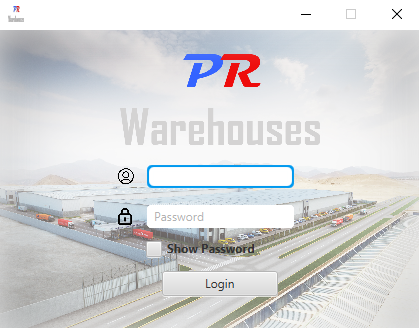
Session session = Connection.*openSession*();  
Transaction transaction = session.beginTransaction();  
String DELETE\_QUERY = "DELETE FROM Usernotifications WHERE idNotifications = :idNotif";  
try{  
 session.createQuery(DELETE\_QUERY).setParameter("idNotif", id).executeUpdate();  
}catch (Exception ex){  
 *log*.error("Notifications deletion unsuccessful " + "\n" + ex.getMessage());  
}finally {  
 transaction.commit();  
}

SELECT

Session session = Connection.*openSession*();  
Transaction transaction = session.beginTransaction();  
String NOTIFICATION\_QUERY = "SELECT u FROM Usernotifications u WHERE idFromUser.userId = :userID";  
try {  
 List<Usernotifications> notifications = session.createQuery(NOTIFICATION\_QUERY).setParameter("userID", UserSession.*getUserID*()).getResultList();  
 UserSession.*setNotifications*(notifications);  
} catch (Exception ex) {  
 *log*.error("Notifications retrieval unsuccessful " + "\n" + ex.getMessage());  
} finally {  
 transaction.commit();  
}

INSERT (генериран чрез hibernate)

Session session = Connection.*openSession*();  
Transaction transaction = session.beginTransaction();  
try {  
 session.save(obj);  
 *log*.info("User saved successfully");  
} catch (Exception ex) {  
 *log*.error("User save error" + ex.getMessage());  
} finally {  
 transaction.commit();  
 session.close();  
}

IIIiii. Реализация на бизнеслогика и графичен интерфейс

Логин менюто се състои от две текстови полета, където потребителят въвежда неговото потребителско име и парола, и бутон за логин, който проверява за съответсвие в базата данни, чрез SELECT заявка. Успешният логин записва текущият потребител в паметта и отваря съответният прозорец отговарящ за ролята му.

<TextField fx:id="usernameTF" layoutX="146.0" layoutY="134.0" promptText="Username" />  
<TextField fx:id="passwordVisibleTF" layoutX="146.0" layoutY="174.0" promptText="Password" />  
<PasswordField fx:id="passwordTF" layoutX="146.0" layoutY="174.0" promptText="Password" />

<Button fx:id="loginButton" layoutX="162.0" layoutY="241.0" mnemonicParsing="false" onAction="#login" prefHeight="25.0" prefWidth="116.0" text="Login" />

Бутона има атрибут onAction, който извиква функцията login.

Част от нея:

String LOGIN\_QUERY = "SELECT u FROM User u WHERE username = '" + usernameTF.getText() + "' AND password = '" + passwordTF.getText() + "'";

User result = (User) session.createQuery(LOGIN\_QUERY).getSingleResult();  
if(result.getRoleId().getRoleId() == 1){ // LOAD ADMIN VIEW  
 closeLoginMenu();  
 UserSession.*getInstance*(result);  
 Panes.*openWindow*("/Views/AdminViews/AdminGUI.fxml", AdminGUI.class);  
}  
if(result.getRoleId().getRoleId() == 2){ // LOAD OWNER VIEW  
 closeLoginMenu();  
 UserSession.*getInstance*(result);  
 Panes.*loadOwner*();  
 Panes.*checkForExpiringFormulars*(true);  
 Panes.*openWindow*("/Views/OwnerViews/OwnerGUI.fxml", OwnerGUI.class);  
}  
if(result.getRoleId().getRoleId() == 3){ // LOAD AGENT VIEW  
 closeLoginMenu();  
 UserSession.*getInstance*(result);  
 Panes.*loadAgent*();  
 Panes.*checkForExpiringFormulars*(false);  
 Panes.*openWindow*("/Views/AgentViews/WarehouseAgentGUI.fxml", WarehouseAgentGUI.class);  
}

Admin GUI:

Owner GUI:

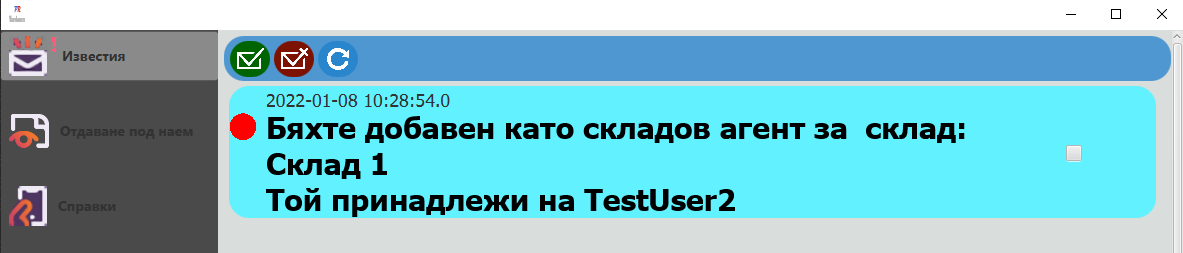
Agent GUI: 

Трите основни прозореца съдържат сходни контроли. Всеки има бутон за известия, справки , настройки, лог оут. Също така в дясно всеки има Pane, в който се зареждат резултатите от натискането на тези бутони.

<AnchorPane fx:id="ContentAnchorPane" minHeight="-Infinity" minWidth="-Infinity" prefHeight="200.0" prefWidth="200.0" GridPane.columnIndex="1" GridPane.hgrow="ALWAYS" />

Бутоните съдържат HBox, чрез който се разпределят текст и икона в него:

<Button fx:id="warehouseButton" blendMode="SRC\_ATOP" mnemonicParsing="false" onAction="#warehousesOnAction" prefHeight="24.0" prefWidth="197.0" styleClass="btns">  
 <graphic>  
 <HBox prefHeight="52.0" prefWidth="182.0" spacing="5.0">  
 <children>  
 <ImageView fitHeight="40.0" fitWidth="40.0" pickOnBounds="true" preserveRatio="true">  
 <image>  
 <Image url="@../../Images/Cubes-Складове.png" />  
 </image>  
 </ImageView>  
 <Label alignment="CENTER" prefHeight="62.0" prefWidth="141.0" text="Складове в системата" textOverrun="CLIP" wrapText="true">  
 <font>  
 <Font name="System Bold" size="14.0" />  
 </font>  
 <HBox.margin>  
 <Insets left="6.0" />  
 </HBox.margin>  
 </Label>  
 </children>  
 </HBox>  
 </graphic>  
</Button>

Нотификации:

Разпределени са три бутона: за маркиране като прочетено, за изтриване и опресняване.

Всяка нотификация съдържа отметка, чрез която се прочита или изтрива.

Бутонът за прочитане:

<Button minWidth="23.0" mnemonicParsing="false" onAction="#MarkAsRead" prefHeight="35.0" prefWidth="40.0" style="-fx-background-radius: 70; -fx-background-color: #006400;">  
 <font>  
 <Font name="Tahoma Bold" size="12.0" />  
 </font>  
 <graphic>  
 <ImageView fitHeight="26.0" fitWidth="27.0" pickOnBounds="true" preserveRatio="true">  
 <image>  
 <Image url="@../../Images/readNotification.png" />  
 </image>  
 </ImageView>  
 </graphic>  
 <tooltip>  
 <Tooltip text="Mark as Read" />  
 </tooltip>  
</Button>

Функцията от контролера извиква service-a, който я отбелязва в базата като прочетена.

Визуално показва на потребителя, че нотификацията е прочетена.

@FXML  
public void MarkAsRead(){  
 if(selected.size() > 0){ // if size is 0, do nothing  
 for(int i = 0; i < selected.size(); i++){  
 userNotificationService.markAsRead(selected.get(i).id);  
 selected.get(i).notificationCheck.setSelected(false);  
 selected.get(i).isReadDot.setVisible(false); // REMOVE DOT  
 }  
 reload();  
 selected.clear();  
 }  
}

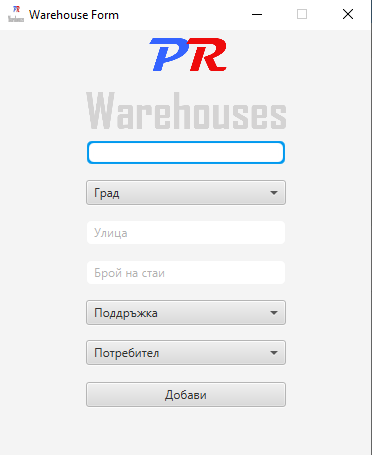
Според ролята на потребителя изгледа на складовете се зарежда по различен начин.

Ако потребителят е админ:



Зареждат се всички складове с информация на кого принадлежат. Стаите в тях, но без възможността да се добавят нови. Не се зарежда листа от агенти. Админът има възможност да добавя нови складове и да ги изтрива чрез контекст меню.

Попълват се нужната информация, като име на склада, града в който се намира, адрес, брой стаи, поддръжка (може да се остави празно), собственикът на който принадлежи.



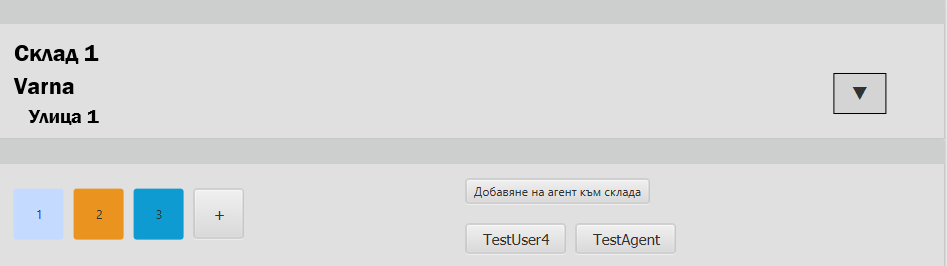
След като данните се валидират, се извиква service-a, който добавя новия склад в базата данни. Ако има поддръжка, се отбелязва като нает.

Броят складове на собственикът се увеличава.

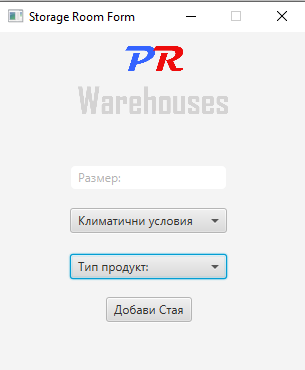
warehouseService.addWarehouse(warehouseNameTF.getText(), (City) cityBox.getValue(), streetTF.getText(), Integer.*parseInt*(roomAmountTF.getText()), 1, true, owner);

userService.increaseOwnerWarehouses(owner);

warehouseService.setEmploymentStatusMaintenance(maintenance, true);

Ако потребителят е собственик: 

Те могат да добавят(изтриват) стаи, докато не запълнят зададения капацитет на склада. Имат бутон, чрез който добавят складови агенти към конкретния склад.

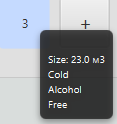


if (warehouseRooms.get(i).getClimateId().getClimate().equals("Dry")) {  
 room.setStyle("-fx-background-color: #EB931F");  
}

Формата за добавяне на стаи работи по същия принцип както формата за склад.

warehouseService.createNewRoom(Double.*parseDouble*(sizeTF.getText()), (Climate) climateBox.getValue(),(ProductType) productBox.getValue(), warehouseService.getWarehouseByID(warehouseID));

Стаите получават цвят според климатичните условия и при hover на мишката показват информация за себе си.

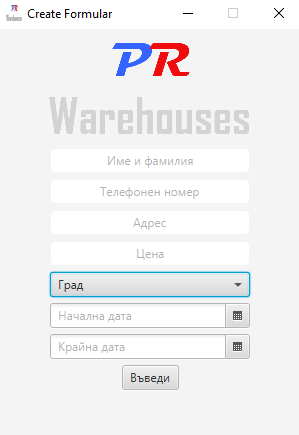


tooltip.setText(String.*format*("\nSize: %1$s\n" +  
 "%2$s\n" +  
 "%3$s\n" +  
 "%4$s\n", warehouseRooms.get(i).getSize() + " м3", warehouseRooms.get(i).getClimateId().getClimate(), warehouseRooms.get(i).getProductId().getType(), warehouseRooms.get(i).isRented() ? "Rented until " + warehouseRooms.get(i).getFormulars().get(warehouseRooms.get(i).getFormulars().size() - 1).getPeriodEnd().toString() : "Free")  
);

Ако потребителят е складов агент:

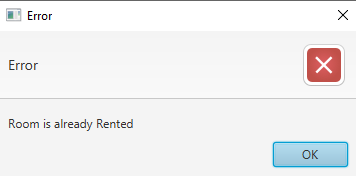
Той получава същият изгледа като на админ без опцията за добавяне на склад.

Има възможността чрез двоен клик на стая да я отдава под наем ако тя е свободна.



formularService.createFormular(room, id, startDate.getValue(), endDate.getValue(), Double.*parseDouble*(priceTF.getText()));

Ако стаята вече е заета се извеща съобщение за грешка.

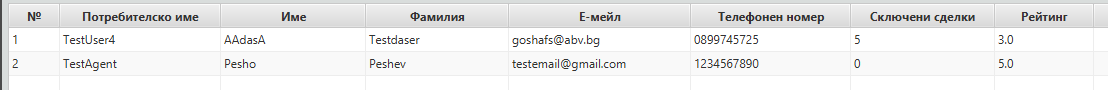


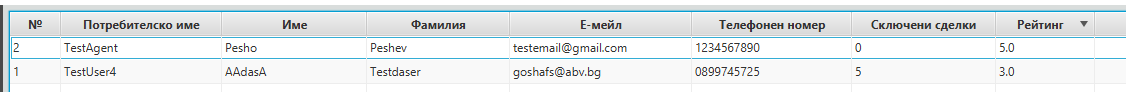
Справки:



Всяка роля има различен брой справки, но всички следват един и същ принцип.

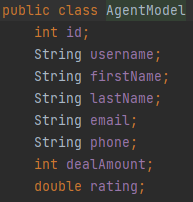
Зарежда се таблица с данни взети от БД. За всяка справка са създадени модели, по които да се map – ват данните. TableView контролът има вграден метод за сортиране на данните.





Пример за модел :

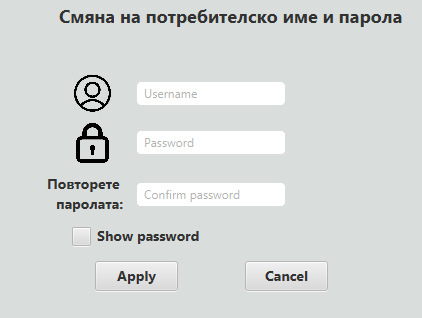
Той съдържа обичайните гетъри и сетъри, както и експлицитен конструктор.



for (int i = 0; i < agentEntities.size(); i++) {  
 List<Rating> agentRating = new ArrayList<>(agentEntities.get(i).getReceivedRatings());  
 agentModels.add(new AgentModel(i + 1,agentEntities.get(i).getIdFromUser().getUsername(),  
 agentEntities.get(i).getIdFromUser().getFirstName(),  
 agentEntities.get(i).getIdFromUser().getLastName(),  
 agentEntities.get(i).getIdFromUser().geteMail(),  
 agentEntities.get(i).getIdFromUser().getPhone(),  
 agentEntities.get(i).getDealAmount(),  
 agentEntities.get(i).getReceivedRatings().size() == 0 ? 0.0 : Panes.*getRating*(agentRating)));  
  
}  
number.setCellValueFactory(new PropertyValueFactory<>("id"));  
username.setCellValueFactory(new PropertyValueFactory<>("username"));  
firstName.setCellValueFactory(new PropertyValueFactory<>("firstName"));  
lastName.setCellValueFactory(new PropertyValueFactory<>("lastName"));  
email.setCellValueFactory(new PropertyValueFactory<>("email"));  
phone.setCellValueFactory(new PropertyValueFactory<>("phone"));  
dealAmount.setCellValueFactory(new PropertyValueFactory<>("dealAmount"));  
rating.setCellValueFactory(new PropertyValueFactory<>("rating"));

Настройки:

Всеки потребител има възможност да смени потребителското си име и парола и да достъпи обща информация за акаунта си.



String CHANGE\_QUERY = "SELECT u FROM User u WHERE userId='" + UserSession.*getUserID*() + "'";   
Date date=new Date();  
try {  
 User result = (User) session.createQuery(CHANGE\_QUERY).getSingleResult();  
 result.setUsername(username);  
 result.setPassword(pass);  
 result.setUpdatedDate(new Timestamp(date.getTime()));  
 UserSession.*setUpdatedDate*(new Timestamp(date.getTime()));



IIIiiii. Реализация на модул за регистриране на събития в системата – log4J.

Логърът се конфигурира при стартиране на програмата.

PropertyConfigurator.*configure*(Main.class.getResource(Constants.Configuration.*LOG4J\_PROPERTIES*));

В конфигурационният файл се специфират неговите настройки. В случая задаваме нивото на логване да бъде ERROR и да се append-ва към конзолата и файл (специфиран на последният ред).

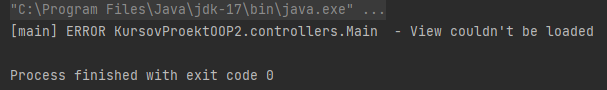
log4j.rootLogger=ERROR, console, file  
  
log4j.appender.console=org.apache.log4j.ConsoleAppender  
log4j.appender.console.layout=org.apache.log4j.PatternLayout  
log4j.appender.console.layout.ConversionPattern=[%t] %-5p %c %x - %m%n  
  
log4j.appender.file=org.apache.log4j.RollingFileAppender  
log4j.appender.file.layout=org.apache.log4j.PatternLayout  
log4j.appender.file.layout.ConversionPattern=[%t] %-5p %c %x - %m%n  
log4j.appender.file.File=application.log

В приложението се използва главно при работа с базата от данни. По време на разработка на приложението се използва като средство за откриване на грешки. Предимството на логването е, че запазва грешките във файл, което позволява тяхното по-късно разглеждане, запазва се и мястото на възникване на съответната грешка. Извежда грешки, ако има такива при естествената работа на приложението без нужда от специална интервенция.

IV. Тестови резултати

При тестовете се подава примерен очакван резултат, който се сравнява с резултат изкаран от изпълнението на дадена функция.

1. Тестване на login: Тук тестваме вписването на акаунт с празни стойности (няма въведени име и парола) като накрая очакваме съобщение за неуспешно вписване и след тества получаваме очаквания резултат.
2. Тестване на main: Тук тестваме инициализацията на main, като би трябвало да се отвори LogiMenu-то, но в случая искаме да не се отвори и да излезе съобщение, че не може се зареди визуализацията, като след изпълнението на теста получаваме очаквания резултат.



1. Тестване на формата за добавяне на склад в системата: Тук тестваме добавянето на нов склад в системата, като се очаква да се въведат няколко описателни параметъра за него. В случая всички стойности са празни и след теста получаваме очаквания резултат за неправилно въвеждане на нужните данни.
2. Тестване на формата за добавяне на стая в определен склад: Тук тестваме добавянето на нова стая към даден склад, като трябва да се въведат няколко параметъра В случая всички стойности са празни и след теста получаваме очаквания резултат за неправилно въвеждане на нужните данни.