Федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и технологий

Кафедра «Распределенные вычисления и компьютерные сети»

**ОТЧЕТ**

Электронная фондовая биржа

по дисциплине

«Разработка корпоративных распределенных web-приложений

с помощью технологии Enterprise Java Beans»

Выполнил

студент гр. 63507/1 И.А. Рапенок

Руководитель

асс. И.В. Стручков

Санкт-Петербург

2015

Содержание

[Анализ задания 2](#_Toc470141886)

[Постановка задачи 2](#_Toc470141887)

[Варианты использования 2](#_Toc470141888)

[Модель предметной области 4](#_Toc470141889)

[Диаграмма классов 4](#_Toc470141890)

# Анализ задания

## Постановка задачи

В данной работе разрабатывается корпоративное распределенное web-приложение (далее электронная фондовая биржа) с помощью технологии Enterprise Java Beans.

Электронная фондовая биржа: торговля акциями компании X. Операции удаленного объекта: выставить лот на продажу по цене Pпр, сделать запрос на покупку по цене Pпок, получить список предложений, совершить сделку (продать или купить). Сериализуемые объекты: предложение и сделка (тип, цена).

Exchange app разрабатывается с учетом надежности, безопасности, а так же возможности его дальнейшей доработки. Клиентом данного приложения является пользователь. Пользователь может покупать и продавать акции.

## Варианты использования

Электронная фондовая биржа включает в себя следующие основные сценарии:

1. **Регистрация**:

Основной сценарий:

1. Система предоставляет форму для добавления учетной записи
2. Пользователь вводит имя учетной записи, пароль и подтверждение пароля
3. Пользователь выбирает элемент интерфейса «Зарегистрироваться»
4. Система добавляет учетную запись

Расширения:

* 1. Пользователь вводит существующее имя пользователя

4.1.1 Система выводит сообщение об ошибке над полем ввода и предлагает пользователю заполнить поля еще раз

4.2 Пароль и подтверждение пароля не совпадают или поле пароль не заполнено

4.2.1 Система выводит сообщение об ошибке над полем ввода и предлагает пользователю заполнить поля еще раз

1. **Авторизация**:

Основной сценарий:

* 1. Система предоставляет форму для авторизации
  2. Пользователь вводит существующий логин и соответствующий пароль
  3. Пользователь выбирает элемент интерфейса «Войти»
  4. Пользователь переходит на главную страницу эллектронной биржи под своей учетной записью

Расширения:

* 1. Пользователь вводит не существующее имя пользователя или некорректный пароль

4.1.1 Система выводит сообщение об ошибке над полем ввода и предлагает пользователю заполнить поля еще раз

1. **Создать заявку на покупку или продажу нескольких акций по некоторой цене:**

Основной сценарий:

1. Система предоставляет форму для создания заявки
2. Пользователь выбирает название акции из списка всех присутствующих в системе акций, выбирает тип заявки (покупка/продажа), количество и цену
3. Пользователь выбирает элемент интерфейса «Создать заявку»
4. Система добавляет новую сделку, уменьшает количество акций данного типа на количество в заявке, для заявок на продажу или уменьшает количество денег пользователя на цену предложенную в заявке, для заявок на покупку
5. **Получить список предложений:**

Основной сценарий:

1. Система предоставляет список всех актуальных на текущий момент предложений (название акции, рыночная цена, количество, цена). Сначала идут заявки текущего пользователя, затем предложения о продаже и далее предложения о покупке
   1. Рядом с каждой заявкой пользователя присутствует элемент интерфейса «Отменить»
   2. Рядом с каждой заявкой на продажу присутствует элемент интерфейса «Купить»
   3. Рядом с каждой заявкой на покупку присутствует элемент интерфейса «продать»
2. **Отменить лот или предложение на покупку:**

Основной сценарий:

1. Система предоставляет список предложений (вариант использования 4)
2. Пользователь выбирает элемент интерфейса «Отменить» рядом с заявкой, которую требуется отменить
3. Система удаляет заявку и возвращает пользователю акции или деньги в зависимости от типа заявки

Расширения:

* 1. Заявку успели принять в промежуток с обновления списка до удаления заявки
  2. Заявка пропадает из списка

1. **Совершить сделку (купить акции):**

Основной сценарий:

1. Система предоставляет список предложений (вариант использования 4)
2. Пользователь выбирает элемент интерфейса «Купить» рядом с заявкой, которую требуется подтвердить
3. Количество денег пользователя уменьшается на цену указанную в заявке, количество денег инициатора заявки увеличивается на соответствующую сумму. Количество акций (указанных в заявке) увеличивается на количество указанное в заявке.
4. Заявка пропадает из списка

Расширения:

* 1. Заявку успели принять в промежуток с обновления списка до подтверждения заявки
  2. Заявка пропадает из списка

1. **Совершить сделку (продать акции):**

Основной сценарий:

1. Система предоставляет список предложений (вариант использования 4)
2. Пользователь выбирает элемент интерфейса «Продать» рядом с заявкой, которую требуется подтвердить
3. Количество денег пользователя увеличивается на цену указанную в заявке,. Количество акций (указанных в заявке) уменьшается на количество указанное в заявке, количество соответствующих акций инициатора уменьшается на соответствующее количество
4. Заявка пропадает из списка

Расширения:

* 1. Заявку успели принять в промежуток с обновления списка до подтверждения заявки
  2. Заявка пропадает из списка

## Модель предметной области

На рисунке 1 представлена модель предметной области, где:

* User – Пользователь системы;
* Share – Акция;
* Trade – Сделка включающая некоторое количество акций одного вида, инициатора и клиента.



Рисунок 1. Модель предметной области.

## Диаграмма классов

На рисунке 2 представлена диаграмма классов:

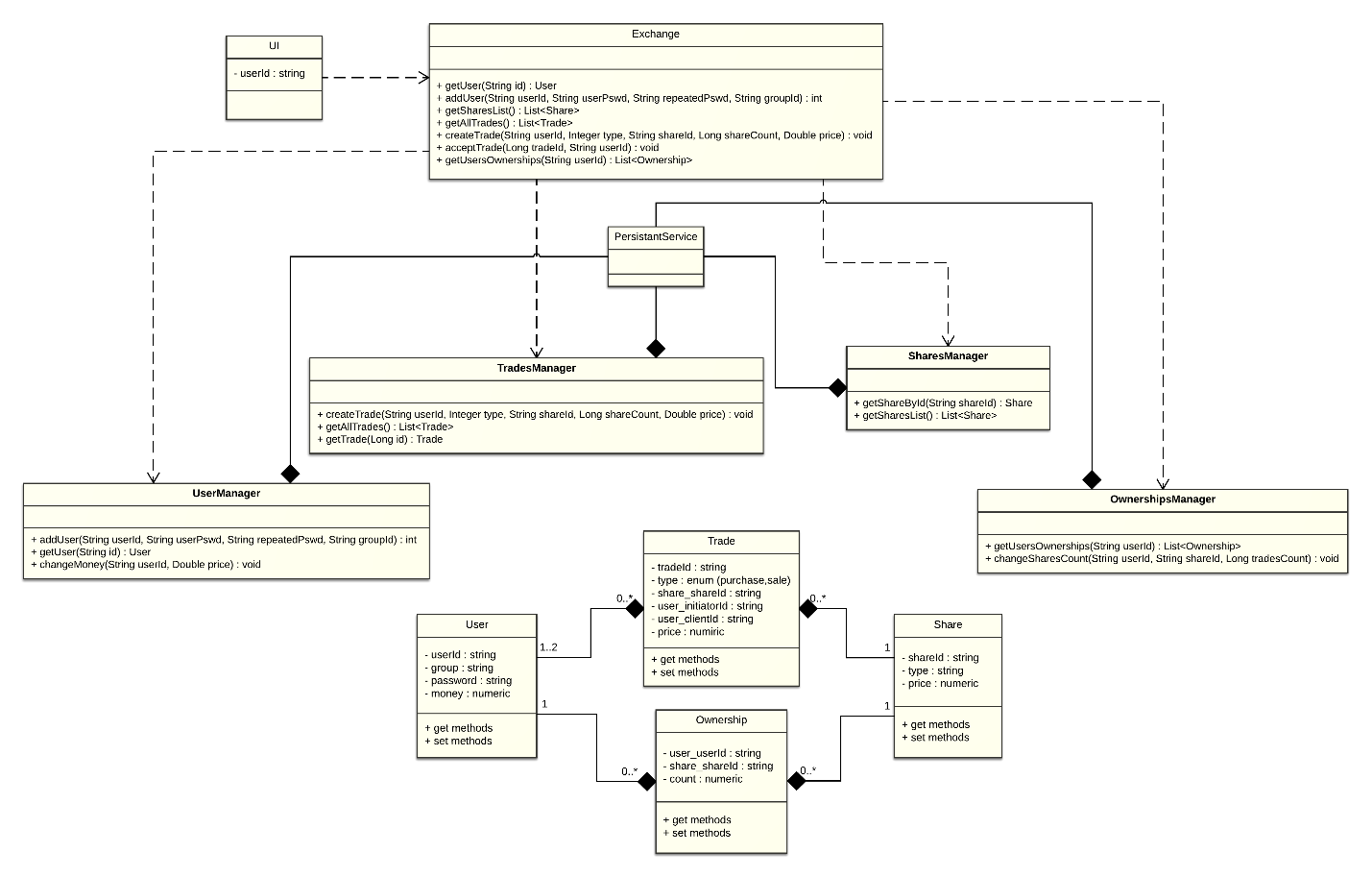


Рисунок 2. Диаграмма классов.

# Реализация задания с помощью технологии «EJB»

## Объектно-ориентированное проектирование

На рисунке 3 представленна диграмма классов учитывающая особенности технологии Enterprise Java Beans. На диаграмме:

* Класс **HttpServlet** – принимает и обрабатывает запросы поступившие от пользователя, на основе их решает как поступить дальше и какой ответ дать пользователю, предоставляет реализацию по умолчанию всех методов интерфейса Servlet ;
* Аннотация **WebServlet** – регистрирует сервлет на сервере. В urlPatterns() указываются адреса вызовов, которые будет обрабатывать сервлет, объявленный с данной аннотацией;
* Класс **CommonController** – обрабатывает запросы пользователя к ресурсам для доступа к которым не требуется авторизация, в данном случае это: страницыприветствия и регистрации.

Методы:

* *void doGet(response, request)* – обрабатыевает GET запросы пользователя. Для страницы приветствия возвращает в ответе клиенту сообщение с предложением перейти к авторизации. Для страницы регистрации возвращает форму регистрации.
* *void doPost(response, request)* – для страницы регистрации обрабатывает введенные данные и инициирует регистрацию пользователя.

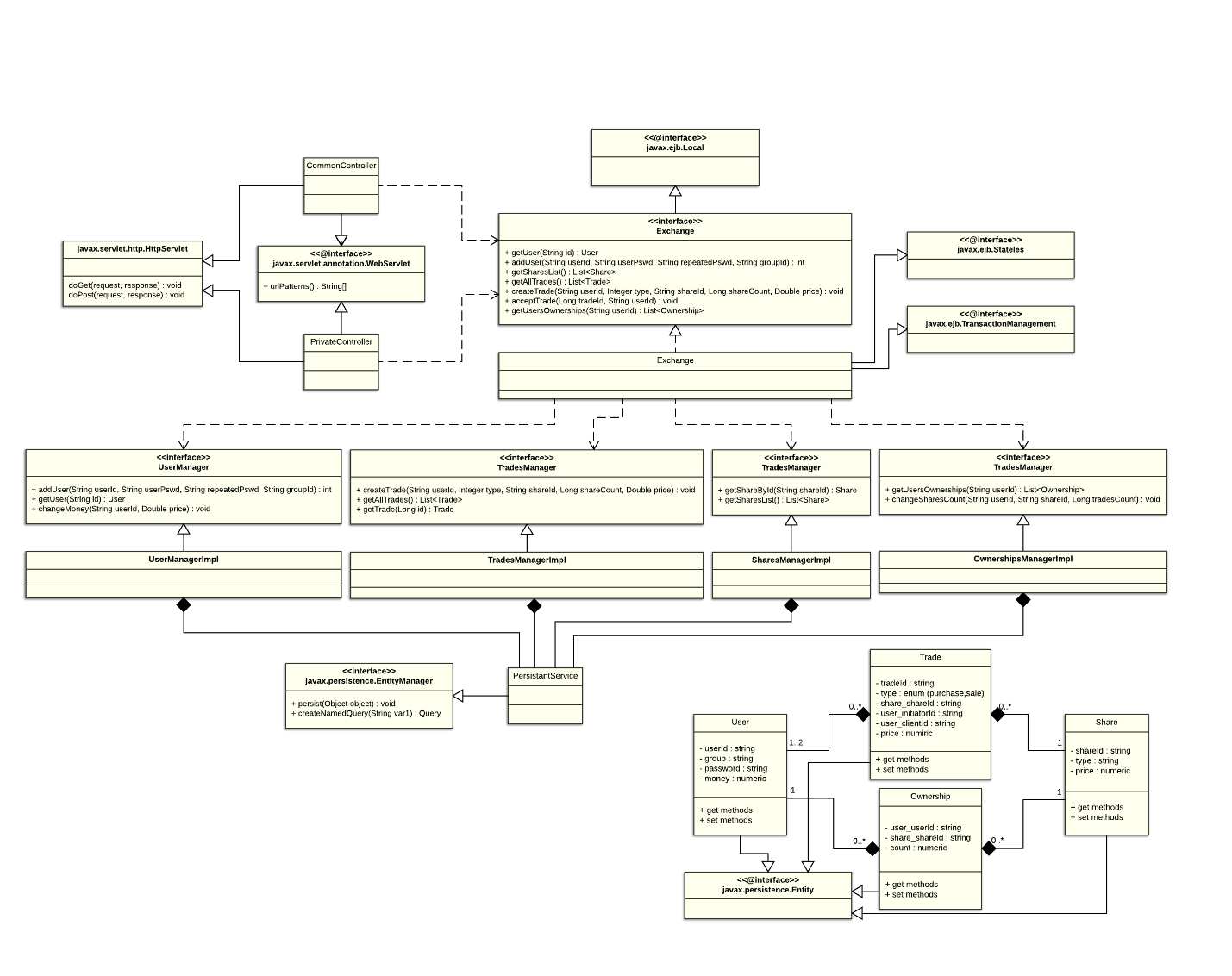


Рисунок 3. Диаграмма классов

* Класс **PrivateController** – обрабатывает запросы пользователя к ресурсам для доступа к которым требуется авторизация, в данном случае это: страницы «биржа» и «личные ресурсы».

Методы:

* *void doGet(response, request)* – обрабатыевает GET запросы пользователя. Для страницы «биржа» предоставляет пользователю интерфейс для просмотра, подтверждения и отммены заявок. Для страницы «личные ресурсы» предосталяет интерфейс для просмтра акций пользователя, а так же интерфейс для создания новой заявки.
* *void doPost(response, request)* – для страницы «биржа» обрабатывает введенные данные и инициирует отмену заявки или подтверждение заявки, созданной другим пользователем. Для страницы «личные ресурсы» обрабатывает введенные пользователем данные и инициирует создание новой акции.
* Аннотация **Local –** локальный бизнесс-интерфейс, позволяет обращаться к компоненту, только из других компонентов, расположенных на том же сервере.
* Аннотация **Entity** – показывает, что класс является сущностью. За отражение объектно-ориентированной структуры на реляционную отвечает служба сохранности.
* Класс **User** – сущностный класс, содержит информацию о пользователях системы:
* *userId* – логин, уникальный идентификатор пользователя;
* *group* – группа нужна для распределения прав доступа (пока только одна);
* *password* – пароль пользователя;
* *money* – баланс пользователя.
* Класс **Share** – сущностный класс, содержит информацию об акциях доступных в системе:
* *shareId* – название, уникальный идентификатор акции;
* *type* – тип акции;
* *price* – некоторая рыночная цена акции;
* Класс **Trade** – сущностный класс, содержит информацию о сделках:
* *tradeId* – уникальный идентификатор сделки;
* *type* – типа покупка/продажа;
* *shareId* – идентификатор акции;
* *shareCount* – количество акций в сделке;
* *price* – общая цена;
* *userInitiator* – идентификатор инициатора сделки;
* *userClient* – идентификатор клиента (null если сделка не подтверждена).
* Класс **Ownership** – сущностный класс отражает информацию об акциях пользователей:
* *userId* – идентификатор владельца акциями;
* *shareId* – идентификатор акции;
* *count* – количество данный акций у пользователя.

Все сущностные классы обладают методами get и set для каждого поля не идентификатора.

* Интерфейс **Exchange** – определяет взаимодействие между контроллерами и модлью.
* Класс **ExchangeImpl** – реализует интерфейс Exchange.

Методы:

* *User* ***getUser****(String id)* – запрашивает информацию о пользователе по его id у UserManager
* *Integer* ***addUser****(String userId, String userPswd, String repeatedPswd, String groupId)* – запрашивает добавить нового пользователя у UserManager
* *List<Share>* ***getSharesList****()* – запрашивает список всех акций у SharesManager
* *List<Trade>* ***getAllTrades****()* – запрашивает список всех актуальных предложений о торговле у TradesManager
* *void* ***createTrade****(String userId, Integer type, String shareId, Long shareCount, Double price)* – создает новую заявку. Метод должен выполняться в рамках одной транзакции. Сначала запрашивается создание новой заявки у TradesManager. Затем создаются запросы на изменение количества ресурсов текущего пользователя у OwnershipsManager или UsersManager.
* *void* ***acceptTrade****(Long tradeId, String userId)* – подтверждение транзакции. Метод должен выполняться в рамках одной транзакции. Сначала запрашивается закрытие транзакции у TradesManager. Затем создаются запросы на изменение количества ресурсов текущего пользователя и инициатора сделки у OwnershipsManager или UsersManager.
* *List<Ownership> getUsersOwnerships(String userId)* – запрашиваются все акции пользователя у OwnershipsManeger.
* Интерфейсы **UsersManager, SharesManager, TradesManager, OwnershipsManager** – определяют взаимодействие с соответствующими сущностными классами (данными).
* Класс **UsersManagerImpl** – реализует интерфейс UsersManager, отвечает за доступ и изменение данных пользователя:

Методы:

* *Integer addUser(String userId, String userPswd, String repeatedPswd, String groupId)* – проверка данных и добавление нового пользователя;
* *User getUser(String id)* – возвращает информацию о пользователе по идентификатору;
* *void changeMoney(String userId, Double price)* – изменение количества денег пользователя.
* Класс **SharesManagerImpl** – реализует интерфейс SharesManager, отвечает за доступ и изменение данных об акциях:

Методы:

* *Share getShareById(String shareId)* – возвращает информацию об акции по идентификатору.
* *List<Share> getSharesList()* – возвращает список всех акций в системе;
* Класс **TradesManagerImpl** – реализует интерфейс TradesManager, отвечает за доступ и изменение сделок:

Методы:

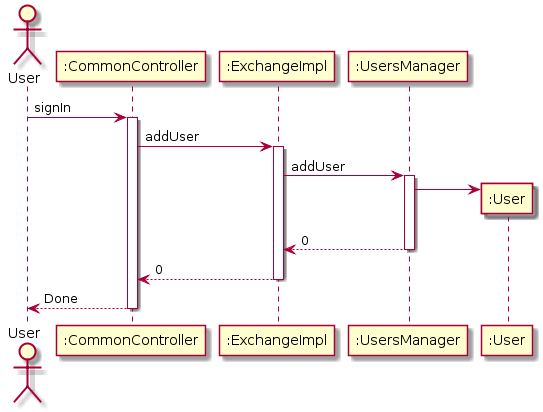
* *void createTrade(String userId, Integer type, String shareId, Long shareCount, Double price)* – логика создания новой сделки;
* *List<Trade> getAllTrades() –* возвращает список всех актуальных сделок;
* *Trade getTrade(Long id) –* возвращает информацию о сделке по идентификатору.
* А
* Класс **OwnershipsManagerImpl** – реализует интерфейс OwnershipsManager, отвечает за доступ и изменение информации об акциях пользователя:

Методы:

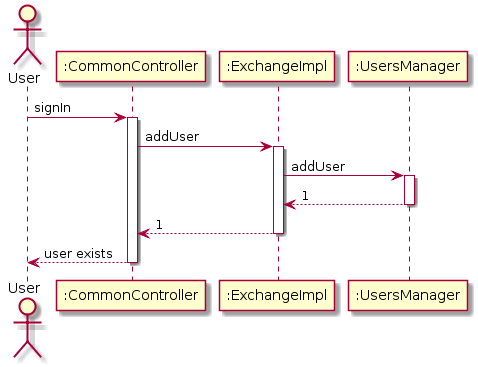
* *List<Ownership> getUsersOwnerships(String userId) -* возвращает список всех акций пользователя;
* *void changeSharesCount(String userId, String shareId, Long tradesCount)* – изменяет количество акций пользователя.
* Интерфейс **EntityManager** служит для управления временем жизни хранимых сущностей (созданием, поиском, удалением и т.д.) Основными методами интерфейса **EntityManager** являются:
* *public void persist(Object entity)* – данный метод позволяет сохранить сущность, представленную объектом сущностного класса в БД;
* *public Query createNamedQuery(String name)* – создать заранее предопределенный запрос к БД (name – название предопределенного запроса).

## Диаграммы последовательности

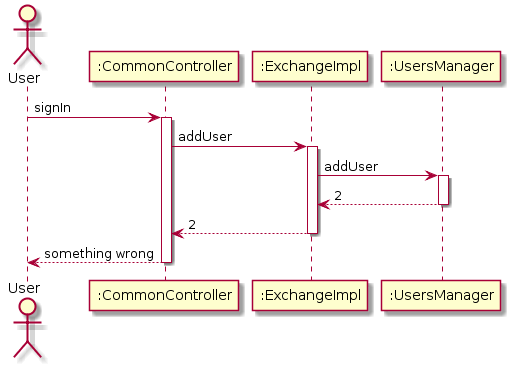
Успешная регистрация:



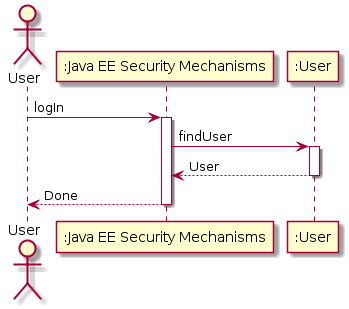
Регистрация существующего пользователя:



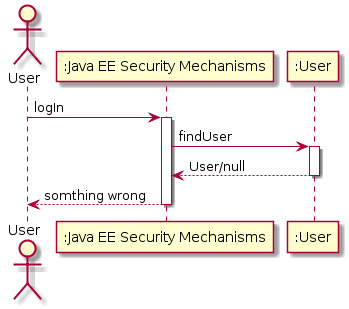
Регистрация, ввод неверных данных:



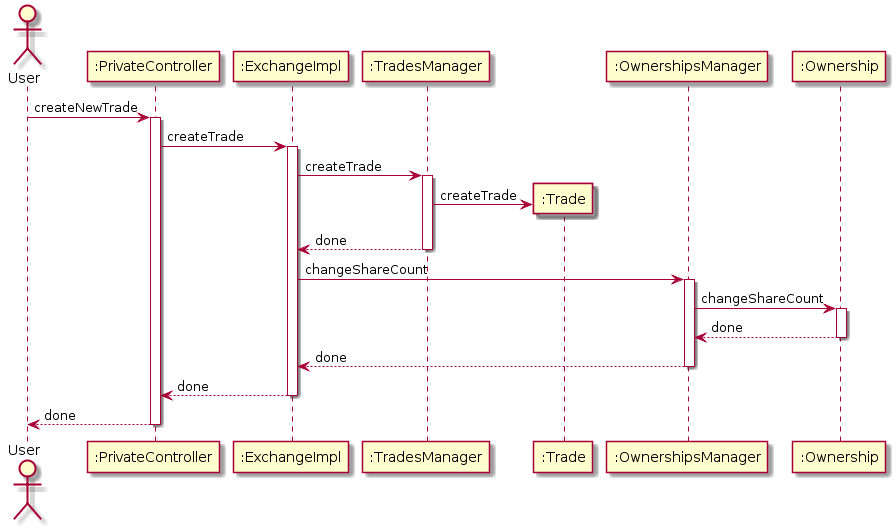
Успешная авторизация:



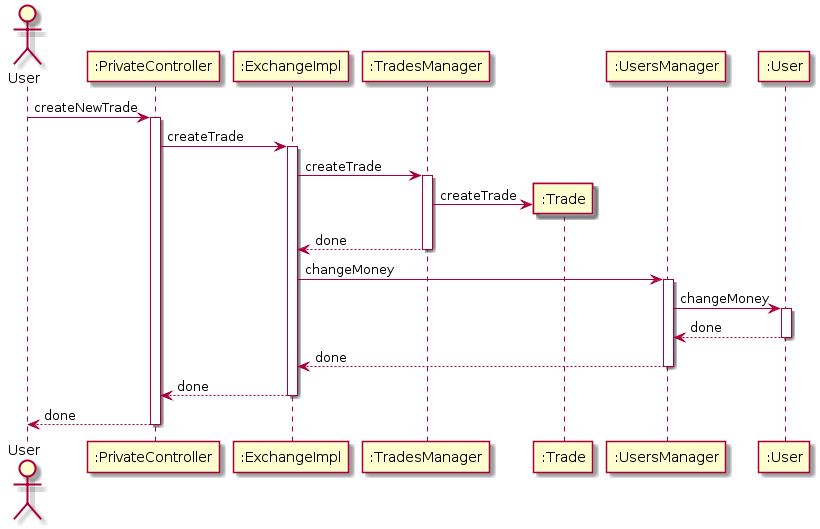
Авторизация, неверные данные:



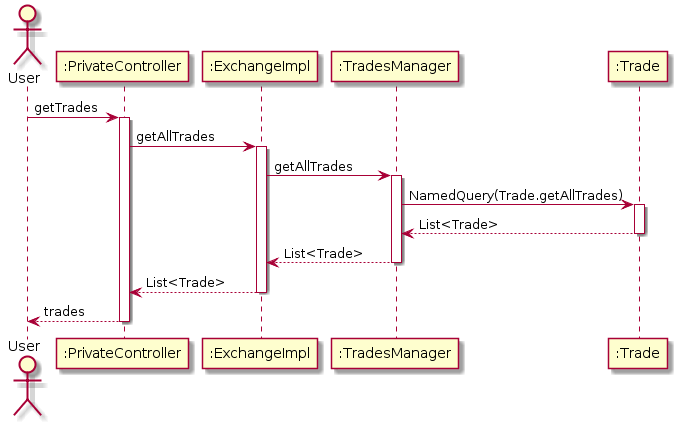
Создание заявки на продажу:



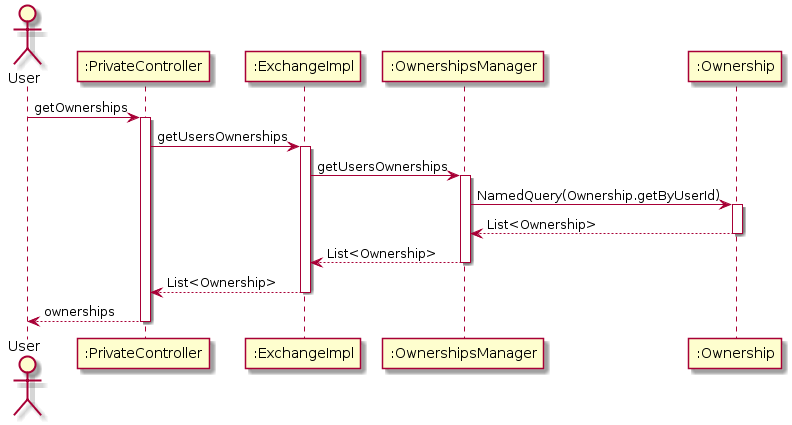
Создание заявки на покупку:



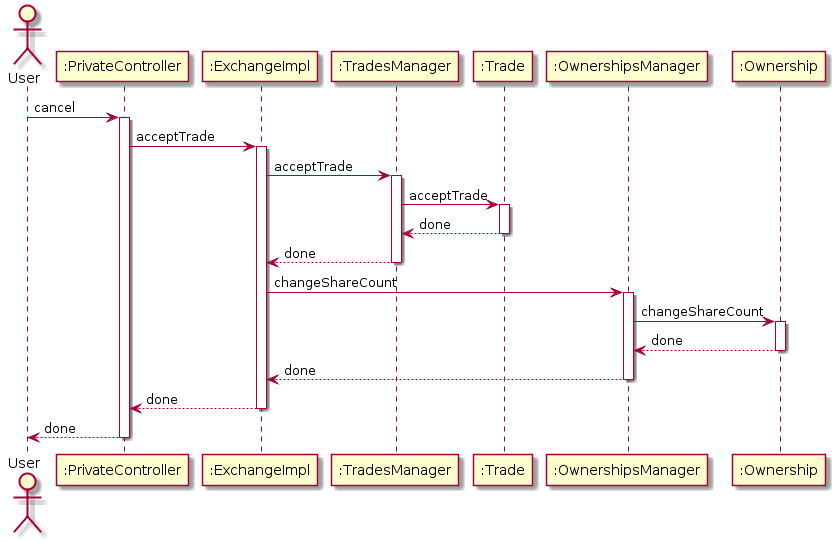
Получить список предложений:



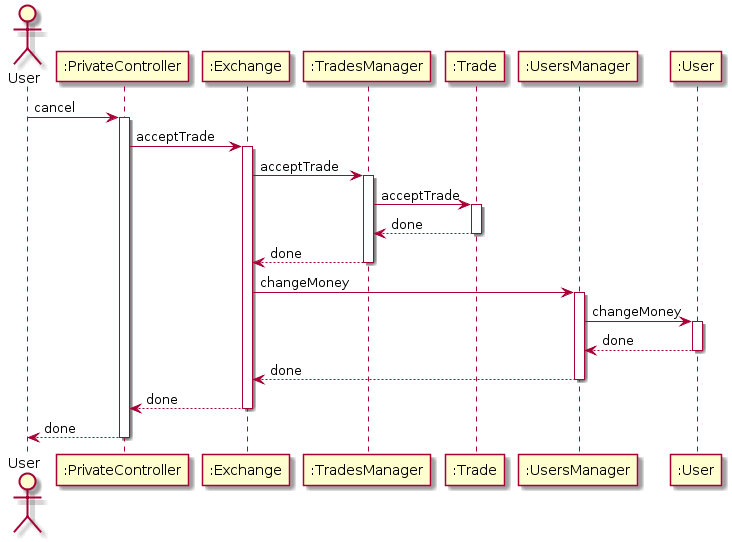
Получить список акций пользователя:



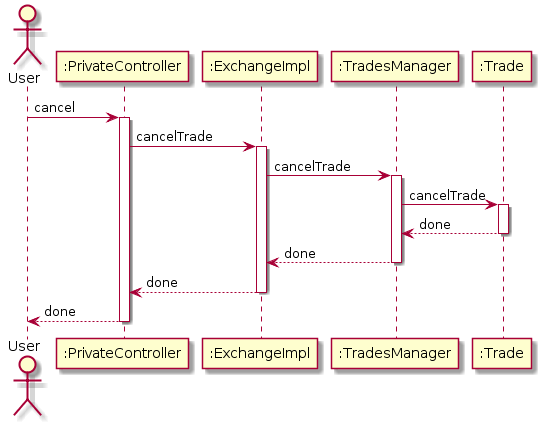
Успешная отмена заявки на продажу:



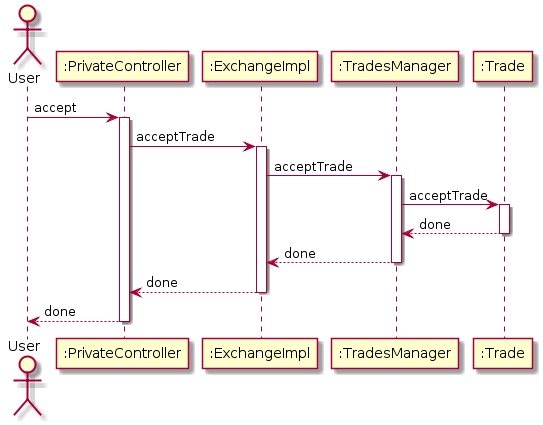
Успешная отмена заявки на покупку:



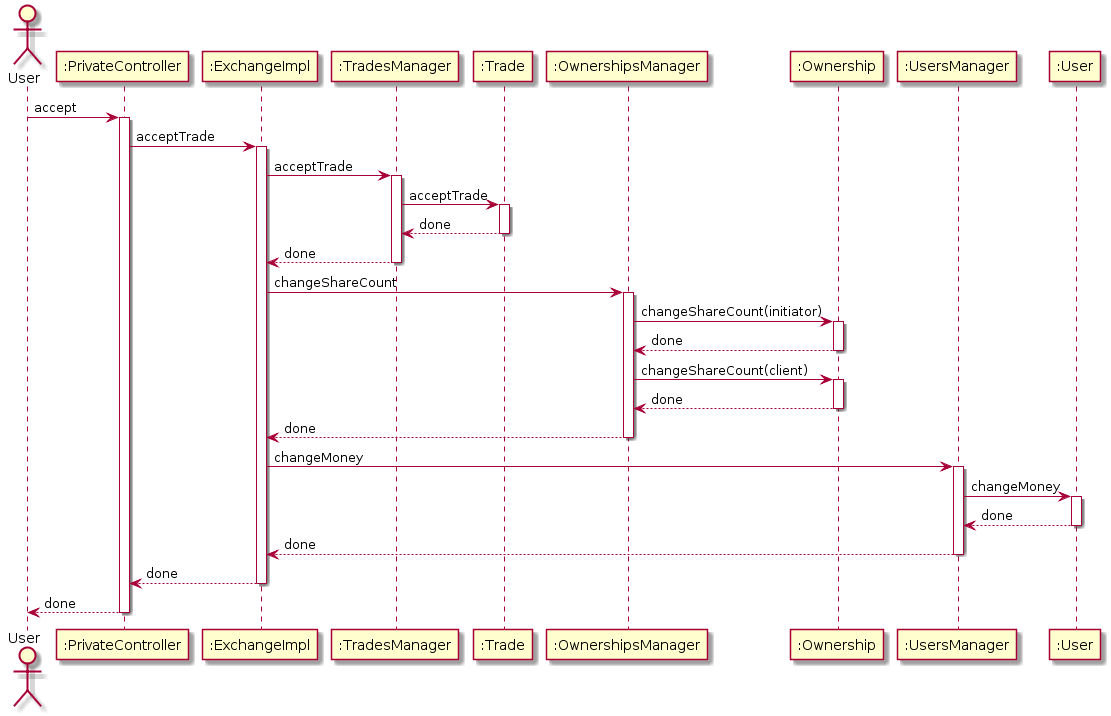
Отмена заявки, заявку уже подтвердили:



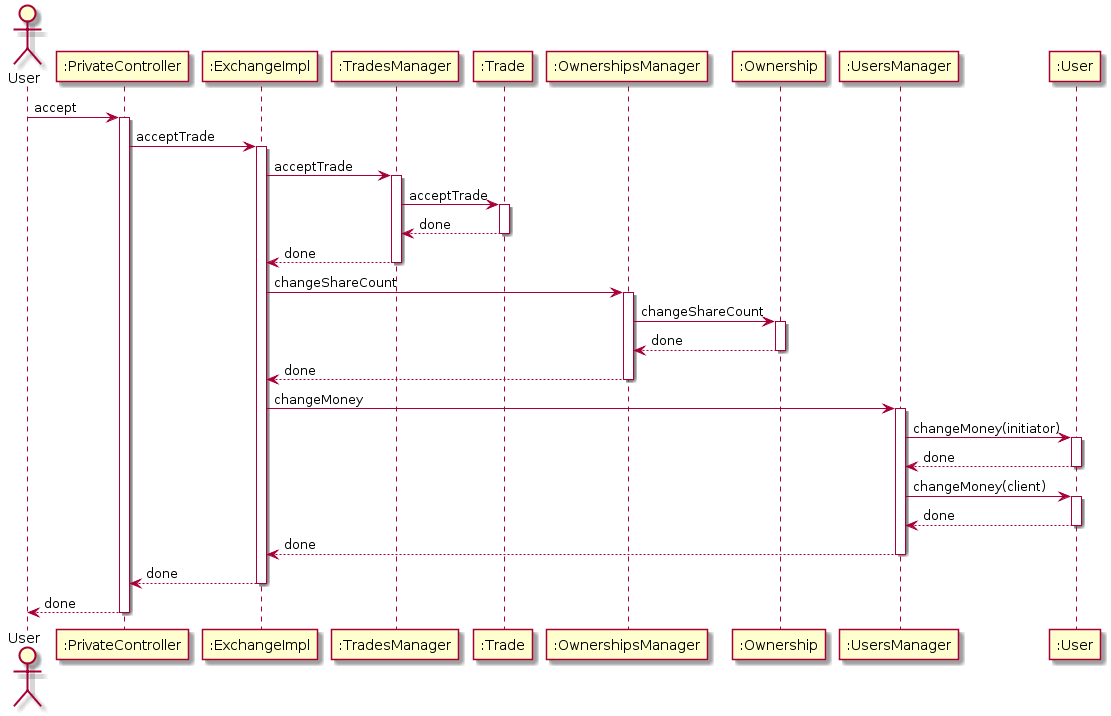
Подтверждение сделки, сделку уже подтвердили или отменили:



Успешное подтверждение сделки на покупку:



Успешное подтверждение сделки на продажу:



## Описание программы

Для того что разграничивать доступ к контенту сайта, был создан раздел доступный только для зарегистрированных пользователей. Был выбран механизм аутентификации на основе форм. Механизм аутентификации основан на сервисе Java Authentication and Authorization Service (JAAS ).

При создании проекта использовался Maven. В pom.xml файле описано из чего состоит проект. Также определены плагины – инструменты сборки проекта. В репозиториях артефактов хранятся продукты сборки программных модулей вместе с метаданными.

В качестве реляционной базы данных используется Oracle Database 12c, приложение использует jdbc-ресурс с jndi-имененм указанным в фале resources \META\_INF\persistence.xml

<**persistence-unit name="em"**>  
 <**jta-data-source**>jdbc/oracle\_pool</**jta-data-source**>  
 <**class**>entities.User</**class**>  
</**persistence-unit**>

Для получения и обработки запросов от пользователя использовались сервлеты CommonController и PrivateController. Для пердставления данных клиенту использовали jsp-страницы (Java Server Pages), которые представляют из себя шаблоны с уже заготовленными HTML-тэгами, между которыми надо вставить нужные данные. В проекте следующие jsp-страницы:

* Для авторизации: authorization.jsp;
* Для регистрации: registration.jsp;
* Для совершения сделок: exchange.jsp;
* Для просмотра ресурсов пользователя и создания новых заявок: trade.jsp.

Чтобы роли сервера Glassfish из базы данных можно было называть произвольным образом в директорию WEB-INF был добавлен файл glassfish-web.xml с сопоставлениями ролям сервера ролей приложения:

<**glassfish-web-app**>  
 <**security-role-mapping**>  
 <**role-name**>private</**role-name**>  
 <**group-name**>user</**group-name**>  
 </**security-role-mapping**>  
</**glassfish-web-app**>

При создании сущности Ownership для хранения составного первичного ключа (userOwner, shareOwn), которые соответственно являются сущностями User и Share, было необходимо использовать дополнительный класс OwnershipId проаннотированный аннотацией @Embeddable с двумя полями типа String. Далее было необходимо смапить эти поля уже на необходимые:

@Embeddable  
**class** OwnershipId **implements** Serializable {  
  
 @Column  
 **private** String userOwner;  
 @Column  
 **private** String shareOwn;

. . .

**public class** Ownership **implements** Serializable {  
  
 @EmbeddedId  
 **private final** OwnershipId ownershipId = **new** OwnershipId();  
  
 @MapsId  
 @JoinColumn(name = **"SHARE\_OWN\_ID"**, referencedColumnName = **"SHARE\_ID"**)  
 @ManyToOne(optional = **false**, cascade = CascadeType.PERSIST)  
 **private** Share shareOwn;  
  
 @MapsId  
 @JoinColumn(name = **"USER\_OWNER\_ID"**, referencedColumnName = **"USER\_ID"**)  
 @ManyToOne(optional = **false**, cascade = CascadeType.PERSIST)  
 **private** User userOwner;

. . .

## Инструкция системного администратора

В данном разделена представлена инструкция для разворачивания приложения:

Шаг 1. Необходимое окружение:

* Сервер GlassFish версии 4.1 и выше, подойдет и другой сервер приложений реализующий спецификации Java EE. Далее будет описана инструкция для сервера GlassFish;
* База данных Oracle Database 11с (скорее всего подойдет и гораздо более ранняя версия)

Шаг 2. Установка приложения:

* Накат схемы данных. Для наката схемы данных необходимо выполнить sql-скрипт createbase.sql находящийся в папке дистрибутива db\_src\Oracle\createbase.sql.

Текст скрипта представлен в приложении 1.

* Настройка соединения с базой данных:
  1. Добавить в каталог [domain\_name]\lib\ext\ домена на котором планируется разворачивать приложение драйвер ojdbc6.jar для взаимодействия с базой данных Oracle. Драйвер находится в папке дистрибутива по пути glassfish\lib\ojdbc6.jar.
  2. Добавьте в файл [domain\_name]\config\domain.xml описание нового Connection pool:

</resources>

<jdbc-connection-pool datasource-classname="oracle.jdbc.pool.OracleDataSource" name="oracle\_pool" res-type="javax.sql.DataSource">

<property name="connectionAttributes" value=";create=true"></property>

<property name="user" value="[db\_user\_name]"></property>

<property name="password" value="[db\_user\_password]"></property>

<property name="portNumber" value="[db\_port]"></property>

<property name="databaseName" value="orcl"></property>

<property name="serverName" value="[db\_host]"></property>

<property name="driverType" value="thin"></property>

</jdbc-connection-pool>

<jdbc-resource pool-name="oracle\_pool" jndi-name="jdbc/oracle\_pool"></jdbc-resource>

</resources>

## Инструкция пользователя

## Методика и результаты тестирования

## Текст программы

# Выводы

# Приложение

#### Приложение 1. Текст скрипта наката базы банных:

#createbase.sql

create table SHARES\_TABLE

(

share\_id VARCHAR2(100) not null,

share\_type VARCHAR2(100),

price NUMBER(20,4)

);

alter table SHARES\_TABLE

add primary key (SHARE\_ID);

alter table SHARES\_TABLE

add constraint PRICE\_NOT\_NULL

check ("PRICE" IS NOT NULL);

create table USERS\_TABLE

(

user\_id VARCHAR2(100) not null,

user\_pswd VARCHAR2(100),

group\_id VARCHAR2(100),

money NUMBER(20,4)

);

alter table USERS\_TABLE

add primary key (USER\_ID);

alter table USERS\_TABLE

add constraint MONEY\_NOT\_NULL

check ("MONEY" IS NOT NULL);

create table OWNERSHIPS

(

user\_owner\_id VARCHAR2(100) not null,

share\_own\_id VARCHAR2(100) not null,

share\_count NUMBER(10)

);

alter table OWNERSHIPS

add primary key (USER\_OWNER\_ID, SHARE\_OWN\_ID);

alter table OWNERSHIPS

add constraint OWNERFK foreign key (USER\_OWNER\_ID)

references USERS\_TABLE (USER\_ID);

alter table OWNERSHIPS

add constraint OWNSHAREFK foreign key (SHARE\_OWN\_ID)

references SHARES\_TABLE (SHARE\_ID);

create table TRADES

(

trade\_id NUMBER(10) default "EWA"."TRADES\_SEQ"."NEXTVAL" not null,

trade\_type NUMBER(2),

share\_share\_id VARCHAR2(100) not null,

share\_count NUMBER(10) not null,

user\_initiator\_id VARCHAR2(100) not null,

user\_client\_id VARCHAR2(100),

price NUMBER(20,4) not null

);

alter table TRADES

add primary key (TRADE\_ID);

alter table TRADES

add constraint CLIENTFK foreign key (USER\_CLIENT\_ID)

references USERS\_TABLE (USER\_ID);

alter table TRADES

add constraint INITIATORFK foreign key (USER\_INITIATOR\_ID)

references USERS\_TABLE (USER\_ID);

alter table TRADES

add constraint SHAREFK foreign key (SHARE\_SHARE\_ID)

references SHARES\_TABLE (SHARE\_ID);

alter table TRADES

add check (trade\_type in (0, 1));

#### Приложение 2. Исходный код программы: