Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчёт

по курсу «Программное обеспечение распределённых вычислительных систем»

по теме «Разработка системы "Биржевая торговля"»

Выполнил студент гр. 3540901/81502:
Медведев М. А.

Проверил преподаватель:
Стручков И. В.

Санкт-Петербург
2019 г.

Содержание

1 Анализ задания

1.1 Формулировка задания

Необходимо спроектировать и реализовать систему "Биржевая торговля"(Exchange Trading), которая предназначена для автоматизации торговли на бирже. Система должна предоставлять пользователям возможность покупки акций за счет денежных средств. Также система должна позволять выполнять обмен акций на денежные средства. В системе должны быть использованы несколько валют для денежных средств и акций. Также должна быть некоторая система по переводу из денежных средств в акции и наоборот.

1.2 Функциональные требования

Клиент - пользователь системы, связан с биржей через брокера. Клиент может отправлять брокеру следующие заявки:

- открытие/закрытие брокерского счета
- заключение/продление/разрыв договора с брокером
- ввод/вывод средств (покупку/продажу акций)

Брокер - связывающее звено между клиентом и администратором биржи. В обязанности брокера входит:

- приём и обработка любых заявок от клиента
- выполнение заявок клиента

Администратор биржи - сотрудник биржи, выполняющий следующие действия:

- обмен средств клиента на определенные акции
- размещение средств в хранилище
- обновление статуса курса акций на биржи, после любых транзакций

1.3 Описание бизнес-процессов

1.4 Обмен денежных средств на акции

Участники

- Клиент
- Брокер
- Администратор биржи

Этапы

- Подача клиентом заявки на внесение средств на брокерский счет
- Проверка заявки брокером и её одобрение/отклонение
- Подтверждение внесения средств администратором биржи
- Размещение средств в хранилище
- Перевод администратором биржи акций на счет клиента и уведомление брокера

1.5 Обмен акций на денежные средства

Участники

- Клиент
- Брокер
- Администратор биржи

Этапы

- Подача клиентом заявки на обмен акций в денежные средства
- Проверка заявки брокером и её одобрение/отклонение
- Проверка выводимой суммы брокером
- Отправление средств администратору биржи
- Подтверждение вывода средств администратором биржи
- Получение средств администратором биржи
- Зачисление администратора биржи средств на счет клиента

1.6 Варианты использования

1.6.1 Клиент

Подача заявки на открытие брокерского счета

- Клиент отправляет заявку в систему на открытие брокерского счета
- Клиент предоставляет личные данные (ФИО, телефон, серия и номер паспорта)
- Клиент оплачивает годовую комиссию за брокерский счет
- Система подтверждает отплату комиссии

Альтернатива 1 - Некорректные личные данные Клиент предоставляет некорректные личные данные. Необходимо заполнить, соответствующую форму ещё раз.
Альтернатива 2 - Нет оплаты комиссии. Клиент не оплатил комиссию в заданный период или оплата не прошла. Система удаляет данные клиента, и клиенту необходимо отправить новую заявку.
Подача заявки на закрытие брокерского счета

- Клиент отправляет заявку в систему на закрытие брокерского счета
- Клиент указывает свои личные данные, а также реквизиты брокерского счета
- Система проверяет наличие средств на счету:
  - При наличии средств на счете, система присылает клиенту уведомление, о том, что счет будет закрыт втечении заданного периода времени, за который клиент должен успеть вывести средства (отправить заявку на вывод средств)
  - При отсутствии средств на счете, система уведомляет, о том что счет будет закрыт через заданный период времен
- Система блокирует доступ к брокерскому счету, а также к личному кабинету

Альтернатива 1 - Некорректные данные. Клиент предоставляет некорректные данные. Система отправляет уведомление, указывающее на некорректные данные.
Альтернатива 2 - Клиент не успел вывести средства со счета. Если клиент не успел вывести средства с брокерского на счета на момент закрытия, система присваивает себе средства и клиент не может себе их вернуть.
Подача заявки на заключение договора с брокером

- Клиент отправляет заявку в систему на заключение договора с брокером
- Клиент указывает свои личные данные, а также реквизиты брокерского счета
- Система выбирает брокера и закрепляет его за клиентом
- Система высылает договор клиенту, заключенный на определенный срок

Альтернатива 1 - Некорректные данные. Клиент предоставляет некорректные данные. Система отправляет уведомление, указывающее на некорректные данные. Клиент должен отправить заявку с корректными данными.
Подача заявки на продление договора с брокером

- Клиент отправляет заявку в систему на продление договора с брокером за 7 дней до конца срока действия текущего договора
- Система подтверждает заявку и высылает уведомление клиенту о продлении договора на заданный период
-

Альтернатива 1 - Отказ заявки. Брокер не подтверждает заявку, в связи с уходом с биржи или иными обстоятельствами. Система отправляет клиенту уведомление о необходимости подать заявку на заключение договора с новым брокером.
Подача заявки на разрыв договора с брокером

- Клиент отправляет заявку в систему на разрыв договора с брокером в любое время действия договора
- Система обрабатывает заявку и разрывает договор брокера с клиентом
- Система высылает уведомление брокеру о разрыве договора с клиентом

- Система высылает уведомление клиенту о разрыве договора с брокером, а также уведомляет о необходимости подать заявку на заключение договора с новым брокером.

Подача заявки на ввод средств на брокерский счет

- Клиент отправляет заявку брокеру на обмен денежных средств на акции, указывая сумму
- Брокер отправляет клиенту утверждение на внесение средств на брокерский счет
- Клиент отправляет денежные средства на брокерский счет по реквизитам
- Брокер проверяет количество средств на счету
- Брокер отправляет заявку администратору о внесении клиентом средств на брокерский счет
- Администратор обрабатывает заявку брокера
- Администратор запрашивает текущий курс акций у системы
- Администратор рассчитает количество акций, эквивалентное внесенным средствам клиента
- Администратор переводит средства с брокерского счета в хранилище
- Администратор переводит количество акций, эквивалентное внесенным средствам клиента, на брокерский счет и отправляет уведомление брокеру о переводе акций на счет
- Брокер отправляет уведомление клиенту об успешном внесение средств и получении акций

Альтернатива 1 - Некорректная сумма. Клиент указал некорректную сумму. Брокер отправляет уведомление клиенту о необходимости исправить эту ошибку. Текущая заявка будет отклонена.

Альтернатива 2 - Нет оплаты комиссии. Клиент не заплатил годовую комиссию за использование брокерского счета. Брокер отправляет уведомление клиенту о необходимости заплатить комиссию и повторить заявку. Текущая заявка будет отклонена.

Альтернатива 3 - Не совпадение суммы. При проверки брокер заметил о не совпадении суммы, отправленной клиентом и суммы, находящимся на счету. Брокер отправляет уведомление клиенту о том, что только обнаруженная на счету сумма будет обменена на акции.

Подача заявки на вывод средств с брокерского счета

- Клиент отправляет заявку брокеру на вывод средств с брокерского счета, указывая количество акций
- Брокер проверяет оплату годовой комиссии брокерского счета клиента
- Брокер проверяет количество акций для вывода
- Брокер отправляет заявку администратору о продаже акций и выводе средств
- Администратор обрабатывает заявку брокера
- Администратор запрашивает текущий курс акций у системы
- Администратор рассчитывает количество средств, эквивалентное выводимым акциям клиента
- Администратор переводит акции с брокерского счета в хранилище
- Администратор переводит количество средств, эквивалентное количеству акций клиента, на брокерский счет и отправляет уведомление брокеру о переводе средств на счет
- Брокер подтверждает вывод средств на сторонний счет клиента, в течение заданного периода времени
- Брокер отправляет уведомление клиенту об успешном переводе акций в средства, а также в какой период времени клиент может перевести средства на свой сторонний счет

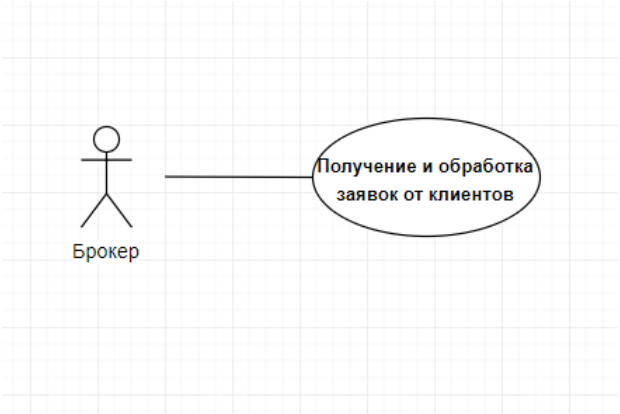
Альтернатива 1 - Некорректная сумма. Клиент указал некорректную сумму. Брокер отправляет уведомление клиенту о необходимости исправить эту ошибку. Текущая заявка будет отклонена.

Альтернатива 2 - Нет оплаты комиссии. Клиент не заплатил годовую комиссию за использование брокерского счета. Брокер отправляет уведомление клиенту о необходимости заплатить комиссию и повторить заявку. Текущая заявка будет отклонена.

Альтернатива 3 - VIP сумма. Если клиент намеревается вывести сумму более заданной, то осуществляются следующие проверки

- Если клиент не является VIP - разрешается вывести не более 100 тыс. акций за одну транзакцию (заявку) и не более 500 тыс. акций за текущий месяц.
- Если клиент является VIP - разрешается вывести не более 300 тыс. акций за одну транзакцию (заявку) и не более 1 млн. акций за текущий месяц.

1.6.2 Брокер



Получение и обработка заявок от клиентов

- Клиент отправляет заявку брокеру
- Брокер получает заявку и сохраняет её в базе данных
- Брокер одобряет и выполняет действия в заявке
-

Альтернатива 1 - Отказ заявки. Брокер не одобряет заявку и не выполняет её. Заявка будет отклонена, клиент будет уведомлен

1.6.3 Администратор

Обмен средств клиента на акции (и наоборот: акции -> средства)

- Администратор получает заявку клиента от брокера
- Администратор сохраняет её в базе данных
- Администратор запрашивает текущий курс акций у системы

- Администратор рассчитывает количество акций, эквивалентное внесенным средствам клиента
- Администратор переводит средства с брокерского счета в хранилище
- Администратор переводит количество акций, эквивалентное внесенным средствам, на брокерский счет клиента
- Администратор отправляет уведомление брокеру о переводе акций на счет

Размещение средств в хранилище

- Администратор получает заявку клиента от брокера с конкретной суммой
- Администратор сохраняет её в базе данных
- Администратор размещает средства в хранилище

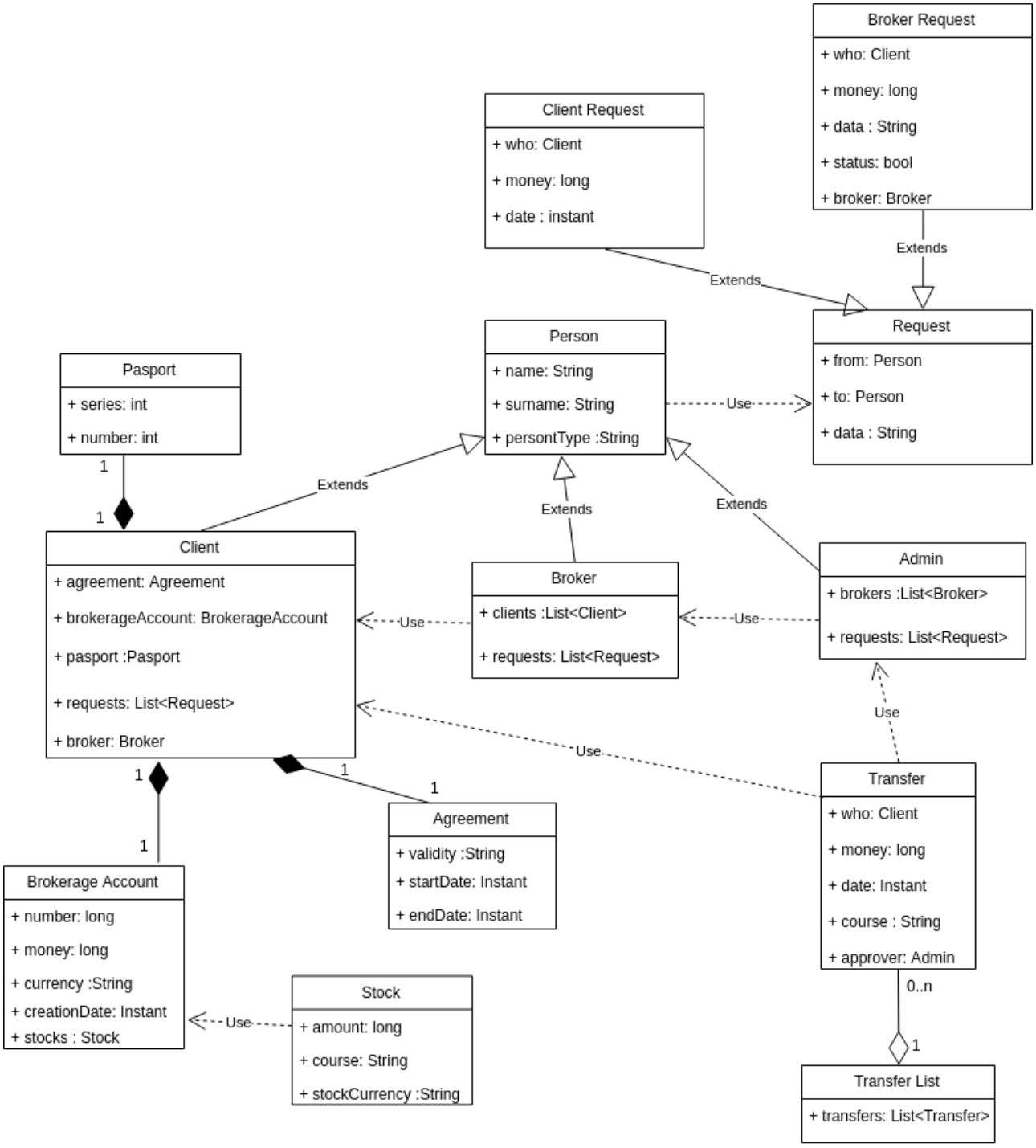
Обновление курса акций на бирже

- Администратор проверяет курс на время последнего изменения курса
- Администратор извлекает из базы данных изменения за последний час
- Администратор анализирует изменения
- Администратор на основе анализа обновляет курс акций на бирже

2 Реализация

2.1 Объектно-ориентированное проектирование с учётом особенностей технологии

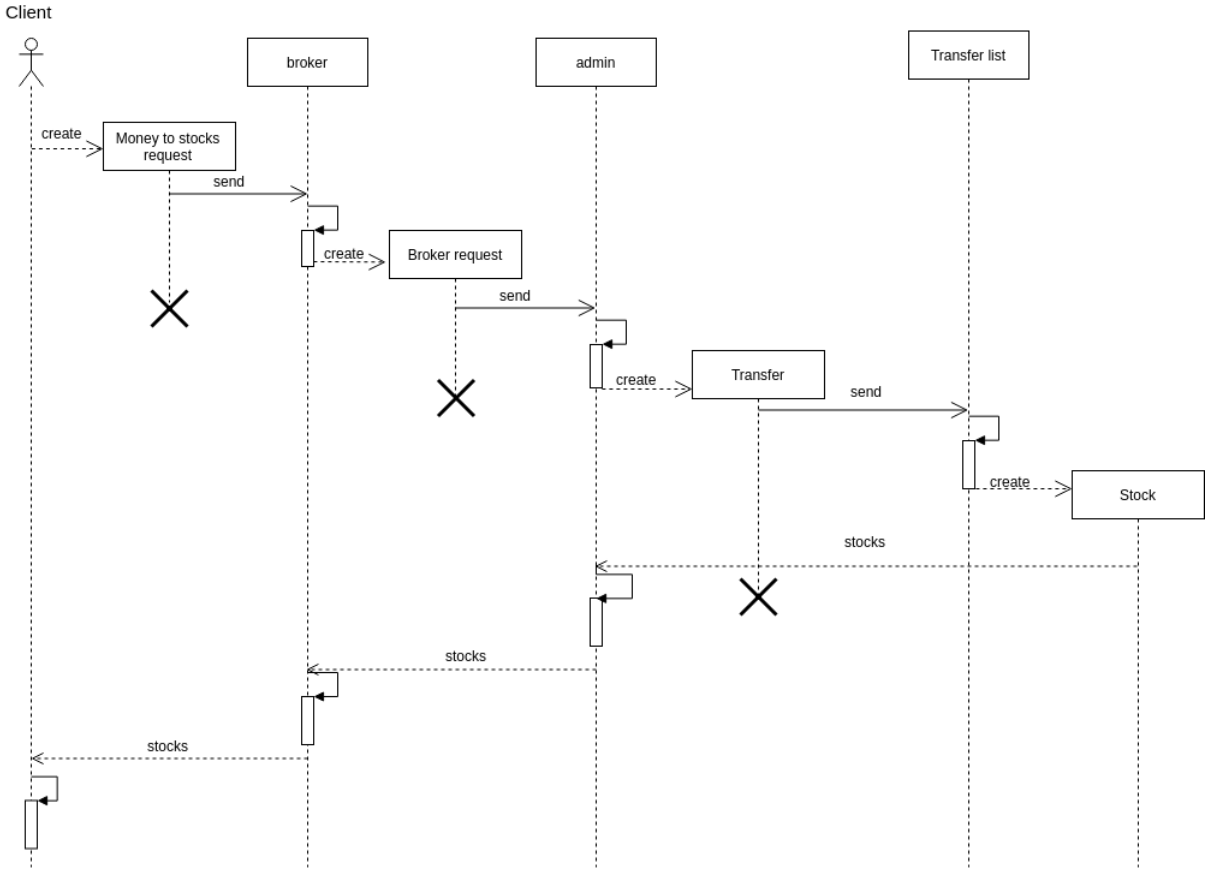
2.1.1 Статическая модель предметной области



На данной диаграмме видно, что есть общая сущность Person от которой наследуются 3 основных актора системы: клиент, брокер и администратор. Клиент взаимосвязан со следующими сущностями: паспорт, брокерский аккаунт, соглашение с брокером. Также на диаграмме присутвует Transfer - эта модель является связкой администратора и клиента, а именно содержит информацию об успешно выполненом денежном обмене. В системе присутвует запросы, которые пересылаются между акторами для передачи информации.

2.1.2 Динамическая модель предметной области

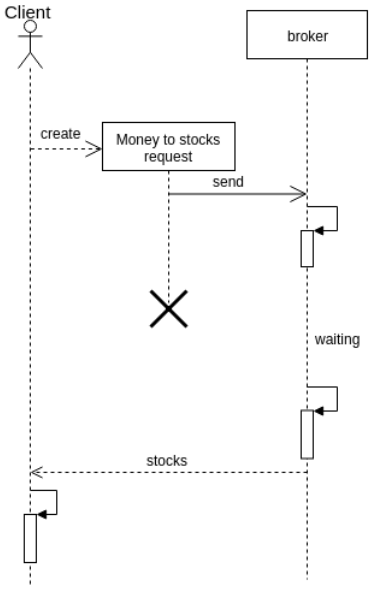
Общая схема



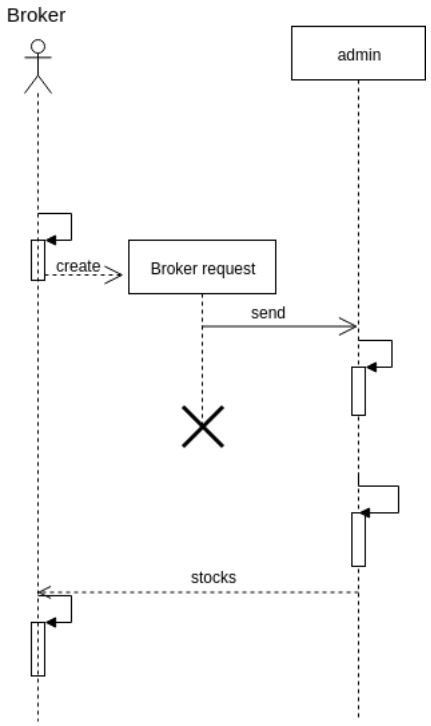
На данной диаграмме можно наблюдать процесс обмена денежных средств на акции. Клиент создает запрос на обмен и отправляет его брокеру. Брокер обрабатывает данный запрос, и после валидации отправляет запрос дальше - администратору, обновляя статус запроса ("одобрено брокером"). Администратор принимает запрос от брокера, просматривает информацию и одобряет его. В этот момент создаётся объект акции, который закрепляется за клиентом и обновляется статус запроса ("завершено").

Ниже на трёх диаграммах представлены отдельные части общей диаграммы, непосредственно отображающие связи между клиентом - брокером, брокера - администратора и администратора - трансфера.

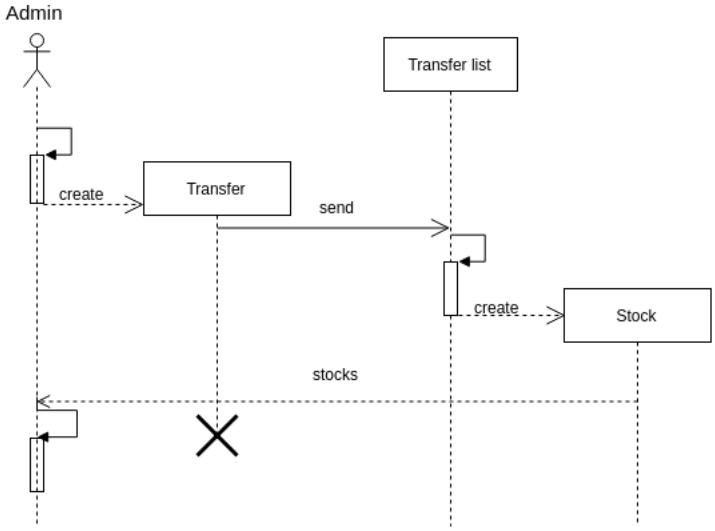
Клиент-Брокер



Брокер-Админ



Админ-Трансфер



3 Описание программы

3.1 Backend

Для реализации бекенда, поставленной задачи был использован Spring Framework. Прежде всего были описаны модели, которые приводились выше. Отличительной чертой в реализации следует отметить наследование от базового класса AbstractEntity, который содержит в себе только уникальный идентификатор. Все модели расширяют данный класс, следовательно у каждой модели есть ID, по которому можно осуществлять поиск в базе данных.

Для взаимодействия с базой данных был написан слой репозиториев. Каждый репозиторий расширяет базовый интерфейс CommonRepository, который в свою очередь расширяет CrudRepository. Это сделано для того, что для каждого репозитория были доступны CRUD-методы.

Над слоем репозиториев был написан слой сервисов. При реализации данного слоя также был выполнен базовый сервис, от которого отнаследованы все остальные сервисы. В сервисах написана вся бизнес-логика данной системы. В каждом сервисе есть методы, которые описывают действия, которые могут быть выполнены конкретной сущностью.

Над слоем сервисов был написан слой REST-контролеров. В данном слое, также была вынесена асбтракция базового контролера, для возможности использования CRUD-методов. В каждом контролере присутствуют методы, которые являются endpoint приложения. В реализации данных методов вызываются соответствующие методы из слоя сервисов.

Все константы, используемые в реализации данной системы были вынесены в специальный файл. Также была настроена swagger-конфигурация для удобного отображения и взаимодействия с endpoint-ами системы. Кроме того, была настроена CORS-конфигурация, - то есть указано, кто может посылать запросы системы извне.

3.2 Frontend

Для реализации фронтенда, поставленной задачи был использован Angular. Было создано приложение, состоящая из компонентов. Каждый компонент представляет из себя совокупность файлов верстки, стилей и бизнес-логики, написанной на языке Typescript. Каждый компонент является отдельным экраном и инкапсулирует в себе соответствующую логику. Также в данном приложение были написаны сервисы, которые посылают запросы и принимают ответы с соответсвующих endpoint'ов бекенда. Константы также были вынесены в отдельный файл. Данное приложение взаимодействует с приложением бекенда.

4 Методика и результаты тестирования

4.1 Варианты использования

Методика тестирования представляла собой перечисление всех возможных вариантов использования системы для всех действующих лиц. Ниже представлены и описаны варианты использования, а также последовательности действий для их выполнения:

4.1.1 Система

- регистрация в системе - необходимо ввести следующую информацию
  - имя действующего лица
  - фамилия действующего лица
  - логин для входа в систему
  - пароль для входа в систему
  - тип действующего лица (клиент, брокер, администратор)
- вход в систему - необходимо ввести следующую информацю
  - логин для входа в систему
  - пароль для входа в систему
  - тип действующего лица (клиент, брокер, администратор)

4.1.2 Клиент

- просмотр личной информации
- открытие брокерского счета
- закрытие брокерского счета
- заключение соглашения с брокером
  - длительность соглашения
- продление соглашения с брокером
  - длительность соглашения
- прекращение соглашения с брокером
- осуществление обмена денежных средств на акции
  - валюта
  - тип акции
  - количество
- осуществление обмена акций на денежные средства
  - тип акции
  - валюта
  - количество
- просмотр существующих запросов
- просмотр успешно завершившихся транзакций

4.1.3 Брокер

- просмотр личной информации
- просмотр информации о всех соглашениях с клиентами
- просмотр всех запросов от клиентов
- валидация каждого запроса от клиента
- одобрение запроса от клиента
- отклонение запроса от клиента

4.1.4 Администратор

- просмотр личной информации
- просмотр информации о закрепленных брокерах
- просмотр информации о наличии в банке всех денежных средств
- просмотр информации о курсе обмена
- просмотр всех запросов, одобренных брокером
- одобрение запроса, одобренного брокером
- отклонение запроса, одобренного брокером

4.2 Ручное тестирование

Было проведено ручное тестирование двух частей системы: бекенда и фронтенда.

4.2.1 Backend

В качестве ручного тестирования со стороны бекенда были выполнены все возможные запросы с REST-клиента. В данных запросах были заданы соответствующие endpoint'ы и заполнены необходимые параметры. В результате были получены ожидаемые ответы, что показывает верную работу севрверной части системы. Для данного вида тестирования был использован REST-клиент Insomnia.

4.2.2 Frontend

В качестве ручного тестирования со стороны фронтенда были выполнены все возможные сценарии для каждого пользователя, используя пользовательский интерфейс. Все сценарии были успешно завершены, тем самым подтверждая корректную работу всей системы.

5 Инструкция системному администратору по развёртыванию приложения

Для развертывания данной системы необходимо наличие любой операционной системы, например Windows/Linux/MacOS. Далее перечислены все средства, требуемые к установки для развёртывания системы локально на машине:

- Intellij IDEA - обычная установка, согласно прилагающейся инструкции
- WebStorm - обычная установка, согласно прилагающейся инструкции
- Postgres - обычная установка, согласно прилагающейся инструкции, версия 9.5 и выше
- Java - обычная установка, согласно прилагающейся инструкции, версия 1.8 и выше
- Node.js - обычная установка, согласно прилагающейся инструкции, версия 12.13.0 и выше
- любой браузер, например Google Chrome или Yandex Browser

После установки всех средств, представленных выше, необходимо включить запустить службу (Windows) или процесс (Linux/MacOS) postgres и создать базу данных со следующим названием: **exchange-trading**. После создания базы данных можно приступать к запуску системы.

6 Инструкция пользователю по запуску приложения

Для запуска системы необходимо последовательно выполнить следующие действия:

- открыть проект, находящийся в папке /core в среде Intellij IDEA
- собрать проект, выбрав пункт меню Build/Build
- запустить проект, нажав на кнопку Run
- проект настроен таким образом, что в базе данных автоматически создадутся все необходимые таблицы
- открыть проект, находящийся в папке /ui в среде WebStorm
- выполнить команду **npm install**
- выполнить команду **npm build**
- выполнить команду **npm serve**
- открыть, установленный браузер и ввести в адресную строку следующий адрес: **http://localhost:4200/**

После последовательного выполнения всех действий, описанных выше можно будет увидеть домашнюю страницу системы биржевой торговли. После этих действий необходимо обязательно зарегистрироваться администратору и выполнить 2 запроса по установке средств в банке, а также по установлению курсов обмена. Данные запросы можно выполнить с помощью REST-клиента или же установленного браузера. Ниже представлены данные запросы:

- http://localhost:8080/api/system/setRates
- http://localhost:8080/api/system/setBankAssets

После выполнения этих запросов необходимо зарегистрировать хотя бы одного брокера. Далее в системе могут регистрироваться клиенты, брокеры и администраторы в любом порядке. После выполнения всех этих действий можно пользоваться системой в полном объёме.

7 Вывод

В ходе данной работы была разработана информационная система "Exchange Trading"("Биржевая торговля"), предназначенная для удобства выполнения различных операций на бирже, как со стороны клиента, так и со стороны администрации (брокеров и админов). В процессе разработки были изучены архитектурные шаблоны, шаблоны проектирования слоев программного обеспечения. Также были пройдены следующие этапы проектирования информационной системы: выявление функциональных требований, описание бизнес-процессов, разработка вариантов использования. В результате получены полезные знания в области проектирования архитектур программного обеспечения, которые очень пригодятся в работе над реальными проектами.

С точки зрения завершенности можно оценить систему, как готовую для использования. Данный вывод можно сделать исходя из успешного ручного тестирования всех возможных методов серверной части. Сторону пользовательского интерфейса можно доработать и улучшить, используя более новые подходы для разработки таких приложений. Например, реализовать ленивую загрузку всех страниц, выполнять переход от экрана к экрану не с помощью редиректов, а используя особенности canActivate, а также реализаовать SPA.

С точки зрения основных свойств распределенных систем, можно оценить системы следующим образом: система является открытой и готова к расширению, также систему можно назвать прозрачной. Систему также можно масштабировать различными способами, можно как реплицировать, так и шардировать. Особых тестов производительности не проводилось, но можно утверждать, что система точно сможет выдержать нагрузку в несколько десятков тысяч пользователей, так как ограничений для этого нет. С точки зрения удоства использования, система представлена довольно удобной и интуитивно понятной, поэтому система будет понятна сразу, даже если вы ей ещё не пользовались.

8 Приложение - листинги

Ниже представлены некоторые фрагменты кода, весь код можно посмотреть по ссылке: **https://github.com/MikhailMe/exchange-trading**

Листинг 1: AdminController

```
package com.kspt.exchangetrading.controllers.actors;

import com.kspt.exchangetrading.configuration.Constants;
import com.kspt.exchangetrading.controllers.CrudController;
import com.kspt.exchangetrading.models.actors.Admin;
import com.kspt.exchangetrading.models.ClientRequest;
import com.kspt.exchangetrading.models.actors.Broker;
import com.kspt.exchangetrading.models.treasury.BankRecord;
import com.kspt.exchangetrading.models.treasury.Rate;
import com.kspt.exchangetrading.models.treasury.Transaction;
import com.kspt.exchangetrading.services.actors.AdminService;
import org.jetbrains.annotations.NotNull;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;

@RestController
@RequestMapping(Constants.Actor.ADMIN)
public final class AdminController extends CrudController<Admin, AdminService> {

    public AdminController(@NotNull AdminService service) {
        super(service);
    }

    @GetMapping("{adminId}/checkRequests")
    public List<ClientRequest> checkRequests(@PathVariable final Long adminId) {
        return service.checkApprovedByBrokerRequests(adminId);
    }

    // clientRequestId
    @PostMapping("{adminId}/approveRequest")
    public Transaction approveRequest(@PathVariable final Long adminId,
                                      @RequestBody final Map<String, String> data) {
        return service.approveRequest(adminId, data);
    }

    // clientRequestId
    @PostMapping("declineRequest/{clientRequestId}")
    public void declineRequest(@PathVariable final Long clientRequestId) {
        service.declineRequest(clientRequestId);
    }

    @GetMapping("getRates")
    public List<Rate> getRates() {
        return service.getRates();
    }

    @GetMapping("getBankAssets")
    public List<BankRecord> getBankMoney() {
        return service.getBankMoney();
    }

    @GetMapping("{adminId}/getBrokers")
    public List<Broker> getBrokers(@PathVariable final Long adminId) {
        return service.getBrokers(adminId);
    }
}
```

Листинг 2: AdminService

```
package com.kspt.exchangetrading.services.actors;

import com.kspt.exchangetrading.configuration.Constants;
import com.kspt.exchangetrading.models.actors.Admin;
import com.kspt.exchangetrading.models.actors.Broker;
import com.kspt.exchangetrading.models.actors.Client;
import com.kspt.exchangetrading.models.ClientRequest;
import com.kspt.exchangetrading.models.treasury.*;
import com.kspt.exchangetrading.models.system.BrokerageAccount;
import com.kspt.exchangetrading.repositories.ClientRequestRepository;
import com.kspt.exchangetrading.repositories.actors.AdminRepository;
import com.kspt.exchangetrading.repositories.actors.BrokerRepository;
import com.kspt.exchangetrading.repositories.actors.ClientRepository;
import com.kspt.exchangetrading.services.CrudService;
import com.kspt.exchangetrading.services.TreasuryService;
import org.jetbrains.annotations.NotNull;
import org.springframework.stereotype.Service;

import javax.transaction.Transactional;
import java.util.ArrayList;
```

```java
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Service
public class AdminService extends CrudService<Admin, AdminRepository> {

    private final TreasuryService treasuryService;
    private final BrokerRepository brokerRepository;
    private final ClientRepository clientRepository;
    private final ClientRequestRepository clientRequestRepository;

    public AdminService(@NotNull final AdminRepository repository,
                        @NotNull final TreasuryService treasuryService,
                        @NotNull final BrokerRepository brokerRepository,
                        @NotNull final ClientRepository clientRepository,
                        @NotNull final ClientRequestRepository clientRequestRepository) {
        super(repository);
        this.treasuryService = treasuryService;
        this.brokerRepository = brokerRepository;
        this.clientRepository = clientRepository;
        this.clientRequestRepository = clientRequestRepository;
    }

    public List<ClientRequest> checkApprovedByBrokerRequests(@NotNull final Long adminId) {
        Admin admin = repository.findById(adminId).orElse(null);
        return admin != null && admin.getIsAuthenticated()
                ? clientRequestRepository
                .findByStatus(Constants.ClientRequestStatus.APPROVED_BY_BROKER)
                .stream()
                .filter(clientRequest -> admin.getBrokers().contains(clientRequest.getBrokerId()))
                .filter(clientRequest -> clientRequest.getAdminId().equals(adminId))
                .collect(Collectors.toList())
                : null;
    }

    public void declineRequest(@NotNull final Long clientRequestId) {
        ClientRequest clientRequest = clientRequestRepository.findById(clientRequestId).orElse(null);
        if (clientRequest != null) {
            clientRequest.setStatus(Constants.ClientRequestStatus.DECLINED);
            clientRequestRepository.save(clientRequest);
        }
    }

    @Transactional
    public Transaction approveRequest(@NotNull final Long adminId,
                                      @NotNull final Map<String, String> data) {
        final long clientRequestId = Long.parseLong(data.get("clientRequestId"));

        Transaction transaction = null;

        ClientRequest clientRequest = clientRequestRepository.findById(clientRequestId).orElse(null);
        if (clientRequest != null) {

            // generate transaction for client
            transaction = generateTransaction(adminId, clientRequest);
            transaction = treasuryService.transactionRepository.save(transaction);

            // update client request
            clientRequest.setStatus(Constants.ClientRequestStatus.COMPLETED);
            clientRequestRepository.save(clientRequest);

            // update client
            Client client = clientRepository.findById(clientRequest.getClientId()).orElse(null);
            if (client != null) {
                transfer(transaction, client.getBrokerageAccount());
                final List<Long> transactions = client.getTransactions();
                transactions.add(transaction.getId());
                client.setTransactions(transactions);
                clientRepository.save(client);
            }

            Admin admin = repository.findById(adminId).orElse(null);
            if (admin != null) {
                List<Transaction> transactions = admin.getTransactions();
                transactions.add(transaction);
                admin.setTransactions(transactions);
                repository.save(admin);
            }

            treasuryService.bankTransfer(clientRequest.getRequestType(), transaction);
        }
        return transaction;
    }

    public List<Rate> getRates() {
        return treasuryService.getRates();
    }

    public List<BankRecord> getBankMoney() {
        return treasuryService.getBankMoney();
```

```
112     }
113
114     public List<Broker> getBrokers(@NotNull final Long adminId) {
115         List<Broker> brokers = new ArrayList<>();
116         repository.findById(adminId).ifPresent(admin -> admin.getBrokers().forEach(brokerId -> {
117             brokerRepository.findById(brokerId).ifPresent(brokers::add);
118         }));
119         return brokers;
120     }
121
122     @NotNull
123     private Transaction generateTransaction(@NotNull final Long adminId,
124                                             @NotNull final ClientRequest clientRequest) {
125         final Long clientId = clientRequest.getClientId();
126         Transaction transaction = new Transaction(adminId, clientId, clientRequest.getRequestType());
127         switch (clientRequest.getRequestType()) {
128             case Constants.Exchange.MONEY_TO_STOCKS: {
129                 final Double moneyQuantity = clientRequest.getQuantity();
130                 final String currency = clientRequest.getFromType();
131                 final Asset transactionAsset = new Asset(clientId, currency, -moneyQuantity);
132                 transaction.setAsset(transactionAsset);
133                 final Stock transactionStock = new Stock(clientId, clientRequest.getToType());
134                 final Double transferStockQuantity = treasuryService.exchangeMoneyToStocks(
                        transactionAsset, transactionStock);
135                 transactionStock.setQuantity(transferStockQuantity);
136                 transaction.setStock(transactionStock);
137                 return transaction;
138             }
139             case Constants.Exchange.STOCKS_TO_MONEY: {
140                 final String stockType = clientRequest.getFromType();
141                 final Double stockQuantity = clientRequest.getQuantity();
142                 final Stock transactionStock = new Stock(clientId, stockType, -stockQuantity);
143                 transaction.setStock(transactionStock);
144                 final Asset transactionAsset = new Asset(clientId, clientRequest.getToType());
145                 final Double transferAssetQuantity = treasuryService.exchangeStocksToMoney(
                        transactionAsset, transactionStock);
146                 transactionAsset.setQuantity(transferAssetQuantity);
147                 transaction.setAsset(transactionAsset);
148                 return transaction;
149             }
150         }
151         return transaction;
152     }
153
154     private void transfer(@NotNull final Transaction transaction,
155                           @NotNull final BrokerageAccount brokerageAccount) {
156         final Asset transactionAsset = transaction.getAsset();
157         final Stock transactionStock = transaction.getStock();
158
159         // update assets
160         List<Asset> assets = brokerageAccount.getAssets();
161         Asset asset = assets
162                 .stream()
163                 .filter(x -> x.getType().equals(transactionAsset.getType()))
164                 .findFirst().orElse(null);
165         if (asset != null) {
166             final Double currentBalance = asset.getQuantity();
167             final Double newCurrentBalance = currentBalance + transactionAsset.getQuantity();
168             asset.setQuantity(newCurrentBalance);
169         } else {
170             final Asset newAsset = new Asset(
171                     transaction.getClientId(),
172                     transaction.getAsset().getType(),
173                     transaction.getAsset().getQuantity()
174             );
175             assets.add(treasuryService.assetRepository.save(newAsset));
176         }
177         brokerageAccount.setAssets(assets);
178
179         // update stocks
180         List<Stock> stocks = brokerageAccount.getStocks();
181         Stock stock = stocks
182                 .stream()
183                 .filter(x -> x.getStockType().equals(transactionStock.getStockType()))
184                 .findFirst().orElse(null);
185         if (stock != null) {
186             final Double currentBalance = stock.getQuantity();
187             final Double newCurrentBalance = currentBalance + transactionStock.getQuantity();
188             stock.setQuantity(newCurrentBalance);
189         } else {
190             final Stock newStock = new Stock(
191                     transaction.getClientId(),
192                     transaction.getStock().getStockType(),
193                     transactionStock.getQuantity());
194             stocks.add(treasuryService.stockRepository.save(newStock));
195         }
196         brokerageAccount.setStocks(stocks);
197     }
198 }
```

Листинг 3: BrokerController

```
package com.kspt.exchangetrading.controllers.actors;

import com.kspt.exchangetrading.configuration.Constants;
import com.kspt.exchangetrading.controllers.CrudController;
import com.kspt.exchangetrading.models.actors.Broker;
import com.kspt.exchangetrading.models.ClientRequest;
import com.kspt.exchangetrading.services.actors.BrokerService;
import org.jetbrains.annotations.NotNull;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;

@RestController
@RequestMapping(Constants.Actor.BROKER)
public final class BrokerController extends CrudController<Broker, BrokerService> {

    public BrokerController(@NotNull BrokerService service) {
        super(service);
    }

    @GetMapping("{brokerId}/checkRequests")
    public List<ClientRequest> checkRequests(@PathVariable final Long brokerId) {
        return service.checkRequests(brokerId);
    }

    // contract: clientRequestId
    @PostMapping("{brokerId}/validateClientRequest")
    public boolean validateClientRequest(@PathVariable final Long brokerId,
                                         @RequestBody final Map<String, Long> data) {
        return service.validateClientRequest(brokerId, data);
    }

    // contract: clientRequestId
    @PostMapping("{brokerId}/approveClientRequest")
    public void approveClientRequest(@PathVariable final Long brokerId,
                                     @RequestBody final Map<String, Long> data) {
        service.approveOrDeclineClientRequest(brokerId, data, true);
    }

    // contract: clientRequestId
    @PostMapping("{brokerId}/declineClientRequest")
    public void declineClientRequest(@PathVariable final Long brokerId,
                                     @RequestBody final Map<String, Long> data) {
        service.approveOrDeclineClientRequest(brokerId, data, false);
    }

}
```

Листинг 4: BrokerService

```
package com.kspt.exchangetrading.services.actors;

import com.kspt.exchangetrading.configuration.Constants;
import com.kspt.exchangetrading.models.actors.Broker;
import com.kspt.exchangetrading.models.actors.Client;
import com.kspt.exchangetrading.models.ClientRequest;
import com.kspt.exchangetrading.models.treasury.Asset;
import com.kspt.exchangetrading.models.treasury.Stock;
import com.kspt.exchangetrading.repositories.ClientRequestRepository;
import com.kspt.exchangetrading.repositories.actors.BrokerRepository;
import com.kspt.exchangetrading.repositories.actors.ClientRepository;
import com.kspt.exchangetrading.services.CrudService;
import org.jetbrains.annotations.NotNull;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Service
public class BrokerService extends CrudService<Broker, BrokerRepository> {

    private final ClientRepository clientRepository;
    private final ClientRequestRepository clientRequestRepository;

    public BrokerService(@NotNull final BrokerRepository repository,
                         @NotNull final ClientRequestRepository clientRequestRepository,
                         @NotNull final ClientRepository clientRepository) {
        super(repository);
        this.clientRequestRepository = clientRequestRepository;
        this.clientRepository = clientRepository;
    }

    public List<ClientRequest> checkRequests(@NotNull final Long brokerId) {
        Broker broker = repository.findById(brokerId).orElse(null);
        return broker != null && broker.getIsAuthenticated()
                ? clientRequestRepository
```

```
                .findByBrokerId(brokerId)
                .stream()
                .filter(clientRequest -> clientRequest.getStatus().equals(Constants.ClientRequestStatus.
                    PROCESSING))
                .collect(Collectors.toList())
                : null;
    }

    public boolean validateClientRequest(@NotNull final Long brokerId,
                                         @NotNull final Map<String, Long> data) {
        Broker broker = repository.findById(brokerId).orElse(null);
        if (broker != null && broker.getIsAuthenticated()) {
            final long clientRequestId = Long.parseLong(data.get("clientRequestId").toString());
            ClientRequest clientRequest = clientRequestRepository.findById(clientRequestId).orElse(null);
            if (clientRequest != null) {
                Client client = clientRepository.findById(clientRequest.getClientId()).orElse(null);
                if (client != null && client.getBrokerageAccount() != null) {
                    switch (clientRequest.getRequestType()) {
                        case Constants.Exchange.MONEY_TO_STOCKS: {
                            for (Asset asset : client.getBrokerageAccount().getAssets())
                                if (asset.getType().equals(clientRequest.getFromType())
                                        && asset.getQuantity() >= clientRequest.getQuantity())
                                    return true;
                        }
                        case Constants.Exchange.STOCKS_TO_MONEY: {
                            for (Stock stock : client.getBrokerageAccount().getStocks())
                                if (stock.getStockType().equals(clientRequest.getFromType())
                                        && stock.getQuantity() >= clientRequest.getQuantity())
                                    return true;
                        }
                    }
                }
            }
        }
        return false;
    }

    public void approveOrDeclineClientRequest(@NotNull final Long brokerId,
                                              @NotNull final Map<String, Long> data,
                                              final boolean isApproved) {
        final long clientRequestId = Long.parseLong(data.get("clientRequestId").toString());
        ClientRequest clientRequest = clientRequestRepository.findById(clientRequestId).orElse(null);
        if (clientRequest != null) {
            if (isApproved) {
                clientRequest.setStatus(Constants.ClientRequestStatus.APPROVED_BY_BROKER);
                Broker broker = repository.findById(brokerId).orElse(null);
                if (broker != null && broker.getIsAuthenticated()) {
                    clientRequest.setAdminId(broker.getAdminId());
                } else {
                    return;
                }
            } else {
                clientRequest.setStatus(Constants.ClientRequestStatus.DECLINED);
            }
            clientRequestRepository.save(clientRequest);
        }
    }
}
```

Листинг 5: ClientController

```
package com.kspt.exchangetrading.controllers.actors;

import com.kspt.exchangetrading.configuration.Constants;
import com.kspt.exchangetrading.controllers.CrudController;
import com.kspt.exchangetrading.models.actors.Client;
import com.kspt.exchangetrading.models.ClientRequest;
import com.kspt.exchangetrading.models.system.Agreement;
import com.kspt.exchangetrading.models.system.BrokerageAccount;
import com.kspt.exchangetrading.models.treasury.Transaction;
import com.kspt.exchangetrading.services.actors.ClientService;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;

@RestController
@RequestMapping(Constants.Actor.CLIENT)
public final class ClientController extends CrudController<Client, ClientService> {

    public ClientController(ClientService clientService) {
        super(clientService);
    }

    // contract: series, number
    @PostMapping("{clientId}/setPassport")
    public Client setPassport(@PathVariable final Long clientId,
                              @RequestBody final Map<String, String> data) {
        return service.setPassport(clientId, data);
    }
```

```
30
31      @PostMapping("openBrokerageAccount/{clientId}")
32      public BrokerageAccount openBrokerageAccount(@PathVariable final Long clientId) {
33          return service.openBrokerageAccount(clientId);
34      }
35
36      @PostMapping("closeBrokerageAccount/{clientId}")
37      public boolean closeBrokerageAccount(@PathVariable final Long clientId) {
38          return service.closeBrokerageAccount(clientId);
39      }
40
41      // contract: money, currency
42      @PostMapping("{clientId}/putMoneyToAccount")
43      public boolean putMoneyToAccount(@PathVariable final Long clientId,
44                                       @RequestBody final Map<String, String> data) {
45          return service.putMoneyToAccount(clientId, data);
46      }
47
48      // contract: validity
49      @PostMapping("{clientId}/makeBrokerAgreement")
50      public Agreement makeBrokerAgreement(@PathVariable final Long clientId,
51                                           @RequestBody final Map<String, Object> data) {
52          return service.makeBrokerAgreement(clientId, data);
53      }
54
55      // contract: validity
56      @PostMapping("{clientId}/extendBrokerAgreement")
57      public Agreement extendBrokerAgreement(@PathVariable final Long clientId,
58                                             @RequestBody final Map<String, Object> data) {
59          return service.extendBrokerAgreement(clientId, data);
60      }
61
62      @PostMapping("{clientId}/breakBrokerAgreement")
63      public boolean breakBrokerAgreement(@PathVariable final Long clientId) {
64          return service.breakBrokerAgreement(clientId);
65      }
66
67      // contract: quantity, fromType(ruble, dollar, euro), toType(mishcoin, realtyincome, cloudflare)
68      @PostMapping("{clientId}/exchangeMoneyToStocks")
69      public ClientRequest exchangeMoneyForStocks(@PathVariable final Long clientId,
70                                                  @RequestBody final Map<String, Object> data) {
71          return service.exchange(clientId, data, Constants.Exchange.MONEY_TO_STOCKS);
72      }
73
74      // contract: quantity, fromType(mishcoin, realtyincome, cloudflare), toType (ruble, dollar, euro)
75      @PostMapping("{clientId}/exchangeStocksToMoney")
76      public ClientRequest exchangeStocksToMoney(@PathVariable final Long clientId,
77                                                 @RequestBody final Map<String, Object> data) {
78          return service.exchange(clientId, data, Constants.Exchange.STOCKS_TO_MONEY);
79      }
80
81      @GetMapping("{clientId}/getTransactions")
82      public List<Transaction> getTransactions(@PathVariable final Long clientId) {
83          return service.getTransactions(clientId);
84      }
85
86      @GetMapping("{clientId}/transaction/{transactionId}")
87      public Transaction getTransactionById(@PathVariable final Long clientId,
88                                            @PathVariable final Long transactionId) {
89          return service.getTransactionById(clientId, transactionId);
90      }
91
92      @GetMapping("{clientId}/getRequests")
93      public List<ClientRequest> getRequests(@PathVariable final Long clientId) {
94          return service.getRequests(clientId);
95      }
96
97      @GetMapping("{clientId}/request/{requestId}")
98      public ClientRequest getRequestById(@PathVariable final Long clientId,
99                                          @PathVariable final Long requestId) {
100         return service.getRequestById(clientId, requestId);
101     }
102 }
```

Листинг 6: ClientService

```
1  package com.kspt.exchangetrading.services.actors;
2
3  import com.kspt.exchangetrading.configuration.Constants;
4  import com.kspt.exchangetrading.models.actors.Broker;
5  import com.kspt.exchangetrading.models.actors.Client;
6  import com.kspt.exchangetrading.models.ClientRequest;
7  import com.kspt.exchangetrading.models.system.*;
8  import com.kspt.exchangetrading.models.treasury.Asset;
9  import com.kspt.exchangetrading.models.treasury.Stock;
10 import com.kspt.exchangetrading.models.treasury.Transaction;
11 import com.kspt.exchangetrading.repositories.ClientRequestRepository;
12 import com.kspt.exchangetrading.repositories.actors.BrokerRepository;
13 import com.kspt.exchangetrading.repositories.actors.ClientRepository;
14 import com.kspt.exchangetrading.repositories.system.AgreementRepository;
```

```
15 import com.kspt.exchangetrading.repositories.system.BrokerageAccountRepository;
16 import com.kspt.exchangetrading.repositories.system.PassportRepository;
17 import com.kspt.exchangetrading.repositories.treasury.AssetRepository;
18 import com.kspt.exchangetrading.repositories.treasury.StockRepository;
19 import com.kspt.exchangetrading.repositories.treasury.TransactionRepository;
20 import com.kspt.exchangetrading.services.CrudService;
21 import org.jetbrains.annotations.NotNull;
22 import org.springframework.stereotype.Service;
23
24 import javax.transaction.Transactional;
25 import java.time.Instant;
26 import java.time.temporal.ChronoUnit;
27 import java.util.*;
28 import java.util.stream.Collectors;
29
30 @Service
31 public class ClientService extends CrudService<Client, ClientRepository> {
32
33     private final AssetRepository assetRepository;
34     private final StockRepository stockRepository;
35     private final BrokerRepository brokerRepository;
36     private final PassportRepository passportRepository;
37     private final AgreementRepository agreementRepository;
38     private final TransactionRepository transactionRepository;
39     private final ClientRequestRepository clientRequestRepository;
40     private final BrokerageAccountRepository brokerageAccountRepository;
41
42     public ClientService(@NotNull final AssetRepository assetRepository,
43                          @NotNull final StockRepository stockRepository,
44                          @NotNull final BrokerRepository brokerRepository,
45                          @NotNull final ClientRepository clientRepository,
46                          @NotNull final PassportRepository passportRepository,
47                          @NotNull final AgreementRepository agreementRepository,
48                          @NotNull final TransactionRepository transactionRepository,
49                          @NotNull final ClientRequestRepository clientRequestRepository,
50                          @NotNull final BrokerageAccountRepository brokerageAccountRepository) {
51         super(clientRepository);
52         this.assetRepository = assetRepository;
53         this.stockRepository = stockRepository;
54         this.brokerRepository = brokerRepository;
55         this.passportRepository = passportRepository;
56         this.agreementRepository = agreementRepository;
57         this.transactionRepository = transactionRepository;
58         this.clientRequestRepository = clientRequestRepository;
59         this.brokerageAccountRepository = brokerageAccountRepository;
60     }
61
62     @Override
63     public Client update(@NotNull final Long id,
64                          @NotNull final Client client) {
65         if (repository.existsById(id)) {
66             if (client.getIsAuthenticated()) {
67                 client.setId(id);
68                 return repository.save(client);
69             }
70         }
71         return null;
72     }
73
74     public Client setPassport(@NotNull final Long clientId,
75                               @NotNull final Map<String, String> data) {
76         final int series = Integer.parseInt(data.get("series"));
77         final int number = Integer.parseInt(data.get("number"));
78         final Passport passport = passportRepository.save(new Passport(series, number));
79         Client client = repository.findById(clientId).orElse(null);
80         if (client != null) {
81             client.setPassport(passport);
82             repository.save(client);
83         }
84         return client;
85     }
86
87     public BrokerageAccount openBrokerageAccount(@NotNull final Long clientId) {
88         BrokerageAccount brokerageAccount = null;
89         Client client = repository.findById(clientId).orElse(null);
90         if (client != null) {
91             if (client.getIsAuthenticated() && client.getPassport() != null && client.getBrokerageAccount
                    () == null) {
92                 brokerageAccount = new BrokerageAccount();
93                 List<Asset> assets = brokerageAccount.getAssets();
94                 if (assets != null) {
95                     Asset savedAsset = assetRepository.save(new Asset(client.getId(), Constants.Currency.
                            RUBLE, -100d));
96                     assets.add(savedAsset);
97                     brokerageAccount.setAssets(assets);
98                     brokerageAccount.setClientPassportId(client.getPassport().getId());
99                     client.setBrokerageAccount(brokerageAccount);
100                    brokerageAccount = repository.save(client).getBrokerageAccount();
101                }
102            }
103        }
```

```
104         return brokerageAccount;
105     }
106
107     @Transactional
108     public boolean closeBrokerageAccount(@NotNull final Long clientId) {
109         Client client = repository.findById(clientId).orElse(null);
110         if (client != null) {
111             if (client.getIsAuthenticated() && client.getAgreement() == null) {
112                 BrokerageAccount brokerageAccount = client.getBrokerageAccount();
113                 if (brokerageAccountRepository.existsById(brokerageAccount.getId())) {
114                     client.setBrokerageAccount(null);
115                     repository.save(client);
116                     brokerageAccountRepository.delete(brokerageAccount);
117                     for (Asset asset: brokerageAccount.getAssets()) assetRepository.delete(asset);
118                     brokerageAccount.setAssets(null);
119                     for (Stock stock: brokerageAccount.getStocks()) stockRepository.delete(stock);
120                     brokerageAccount.setStocks(null);
121                     return true;
122                 }
123             }
124         }
125         return false;
126     }
127
128     public boolean putMoneyToAccount(@NotNull final Long clientId,
129                                      @NotNull final Map<String, String> data) {
130         final double money = Double.parseDouble(data.get("money"));
131         final String currency = data.get("currency");
132
133         Client client = repository.findById(clientId).orElse(null);
134         if (client != null) {
135             if (client.getIsAuthenticated() && client.getBrokerageAccount() != null) {
136                 BrokerageAccount brokerageAccount = client.getBrokerageAccount();
137                 final long brokerageAccountId = brokerageAccount.getId();
138                 if (!brokerageAccountRepository.existsById(brokerageAccountId)) {
139                     return false;
140                 }
141                 List<Asset> assets = brokerageAccount.getAssets();
142                 if (assets != null) {
143                     Asset asset = assets.stream().filter(x -> x.getType().equals(currency)).findFirst().
                             orElse(null);
144                     if (asset != null) {
145                         assets.remove(asset);
146                         Double currentBalance = asset.getQuantity();
147                         asset.setQuantity(currentBalance + money);
148                         assets.add(asset);
149                     } else {
150                         Asset newAsset = assetRepository.save(new Asset(clientId, currency, money));
151                         assets.add(newAsset);
152                     }
153                     brokerageAccount.setAssets(assets);
154                     client.setBrokerageAccount(brokerageAccount);
155                     this.update(clientId, client);
156                     return true;
157                 }
158             }
159         }
160         return false;
161     }
162
163     public Agreement makeBrokerAgreement(@NotNull final Long clientId,
164                                          @NotNull final Map<String, Object> data) {
165         Agreement agreement = null;
166         final String validity = data.get("validity").toString();
167         final Client client = repository.findById(clientId).orElse(null);
168         if (client != null) {
169             if (client.getIsAuthenticated()) {
170                 final Broker vacantBroker = brokerRepository.findAll().isEmpty()
171                         ? null
172                         : getVacantBroker(brokerRepository.findAll(), clientId);
173                 if (vacantBroker != null) {
174                     // create agreement
175                     agreement = new Agreement(clientId, vacantBroker.getId(), validity, Instant.now());
176                     agreementRepository.save(agreement);
177                     // update broker
178                     List<Agreement> brokerAgreements = vacantBroker.getAgreements();
179                     if (brokerAgreements == null) {
180                         brokerAgreements = Collections.singletonList(agreement);
181                     } else if (!brokerAgreements.contains(agreement)) {
182                         brokerAgreements.add(agreement);
183                     }
184                     vacantBroker.setAgreements(brokerAgreements);
185                     brokerRepository.save(vacantBroker);
186                     // update client
187                     client.setAgreement(agreement);
188                     repository.save(client);
189                 }
190             }
191         }
192         return agreement;
193     }
```

```
194 |
195 |     public Agreement extendBrokerAgreement(@NotNull final Long clientId,
196 |                                            @NotNull final Map<String, Object> data) {
197 |         Agreement newClientAgreement = null;
198 |         final String newValidity = data.get("validity").toString();
199 |         final Client client = repository.findById(clientId).orElse(null);
200 |         if (client != null) {
201 |             Agreement clientAgreement = client.getAgreement();
202 |             if (clientAgreement != null) {
203 |                 clientAgreement.setStartDate(Instant.now());
204 |                 clientAgreement.setValidity(newValidity);
205 |                 agreementRepository.save(clientAgreement);
206 |                 newClientAgreement = clientAgreement;
207 |             }
208 |         }
209 |         return newClientAgreement;
210 |     }
211 |
212 |     public boolean breakBrokerAgreement(@NotNull final Long clientId) {
213 |         Client client = repository.findById(clientId).orElse(null);
214 |         if (client != null) {
215 |             Agreement agreement = client.getAgreement();
216 |             if (client.getIsAuthenticated() && agreement != null) {
217 |                 // update broker
218 |                 final long brokerId = agreement.getBrokerId();
219 |                 Broker currentBroker = brokerRepository.findById(brokerId).orElse(null);
220 |                 if (currentBroker != null) {
221 |                     List<Agreement> brokerAgreements = currentBroker.getAgreements();
222 |                     brokerAgreements.remove(agreement);
223 |                     currentBroker.setAgreements(brokerAgreements);
224 |                     brokerRepository.save(currentBroker);
225 |                     // update client
226 |                     client.setAgreement(null);
227 |                     repository.save(client);
228 |                     // remove agreement
229 |                     agreementRepository.delete(agreement);
230 |                     return true;
231 |                 }
232 |             }
233 |         }
234 |         return false;
235 |     }
236 |
237 |     public ClientRequest exchange(@NotNull final Long clientId,
238 |                                   @NotNull final Map<String, Object> data,
239 |                                   @NotNull final String requestType) {
240 |         final Double quantity = Double.parseDouble(data.get("quantity").toString());
241 |         final String fromType = data.get("fromType").toString();
242 |         final String toType = data.get("toType").toString();
243 |
244 |         ClientRequest clientRequest = null;
245 |         Client client = repository.findById(clientId).orElse(null);
246 |         if (client != null && checkAgreement(clientId)) {
247 |             final Long brokerId = client.getAgreement().getBrokerId();
248 |             clientRequest = new ClientRequest(
249 |                     brokerId, clientId,
250 |                     fromType, toType,
251 |                     quantity, requestType);
252 |             clientRequest = clientRequestRepository.save(clientRequest);
253 |             List<Long> clientRequests = client.getRequests();
254 |             clientRequests.add(clientRequest.getId());
255 |             client.setRequests(clientRequests);
256 |             repository.save(client);
257 |         }
258 |         return clientRequest;
259 |     }
260 |
261 |     public List<Transaction> getTransactions(@NotNull final Long clientId) {
262 |         return transactionRepository.findByClientId(clientId);
263 |     }
264 |
265 |     public Transaction getTransactionById(@NotNull final Long clientId,
266 |                                           @NotNull final Long transactionId) {
267 |         Transaction transaction = null;
268 |         Client client = repository.findById(clientId).orElse(null);
269 |         if (client != null) {
270 |             transaction = transactionRepository.findByIdAndClientId(transactionId, clientId);
271 |         }
272 |         return transaction;
273 |     }
274 |
275 |     public List<ClientRequest> getRequests(@NotNull final Long clientId) {
276 |         return clientRequestRepository.findByClientId(clientId);
277 |     }
278 |
279 |     public ClientRequest getRequestById(@NotNull final Long clientId,
280 |                                         @NotNull final Long requestId) {
281 |         ClientRequest clientRequest = null;
282 |         Client client = repository.findById(clientId).orElse(null);
283 |         if (client != null) {
284 |             clientRequest = clientRequestRepository.findByIdAndClientId(requestId, clientId);
```

```
        }
        return clientRequest;
    }

    private boolean checkAgreement(@NotNull final Long clientId) {
        Client client = repository.findById(clientId).orElse(null);
        if (client != null) {
            Agreement clientAgreement = client.getAgreement();
            if (clientAgreement != null) {
                Instant startDate = clientAgreement.getStartDate();
                switch (clientAgreement.getValidity()) {
                    case Constants.Validity.YEAR:
                        if (startDate.plus(Constants.ValidityInts.YEAR, ChronoUnit.DAYS).isAfter(Instant.
                            now())) {
                            return true;
                        }
                    case Constants.Validity.HALF_YEAR:
                        if (startDate.plus(Constants.ValidityInts.HALF_YEAR, ChronoUnit.DAYS).isAfter(
                            Instant.now())) {
                            return true;
                        }
                    case Constants.Validity.MONTH:
                        if (startDate.plus(Constants.ValidityInts.MONTH, ChronoUnit.DAYS).isAfter(Instant
                            .now())) {
                            return true;
                        }
                }
            }
        }
        return false;
    }

    private Broker getVacantBroker(@NotNull final List<Broker> brokers,
                                   @NotNull final Long clientId) {
        Broker minBroker = brokers.get(0);
        int brokerAgreementsSize = Integer.MAX_VALUE;
        for (Broker broker : brokers) {
            if (broker.getAgreements() != null && broker.getAgreements().size() < brokerAgreementsSize) {
                brokerAgreementsSize = broker.getAgreements().size();
                minBroker = broker;
            }
        }
        if (!minBroker.getAgreements()
                .stream()
                .map(Agreement::getClientId)
                .collect(Collectors.toList())
                .contains(clientId)) {
            return minBroker;
        } else {
            brokers.remove(minBroker);
            return getVacantBroker(brokers, clientId);
        }
    }
}
```

Листинг 7: CommonService

```
package com.kspt.exchangetrading.services;

import com.kspt.exchangetrading.models.AbstractEntity;

import java.util.List;
import java.util.Optional;

public interface CommonService<T extends AbstractEntity> {

    long count();

    T create(final T entity);

    void deleteAll();

    void deleteById(final Long id);

    boolean existById(final Long id);

    List<T> getAll();

    Optional<T> getById(final Long id);

    T update(final Long id, final T entity);
}
```

Листинг 8: CrudService

```
package com.kspt.exchangetrading.services;

import com.kspt.exchangetrading.models.AbstractEntity;
import com.kspt.exchangetrading.repositories.CommonRepository;
```

```
import org.jetbrains.annotations.NotNull;

import javax.transaction.Transactional;
import java.util.List;
import java.util.Optional;

public abstract class CrudService<T extends AbstractEntity, R extends CommonRepository<T>>
        implements CommonService<T> {

    @NotNull
    protected final R repository;

    public CrudService(@NotNull final R repository) {
        this.repository = repository;
    }

    public long count() {
        return repository.count();
    }

    public T create(final T entity) {
        return repository.save(entity);
    }

    public void deleteAll() {
        repository.deleteAll();
    }

    @Transactional
    public void deleteById(final Long id) {
        repository.deleteById(id);
    }

    public boolean existById(final Long id) {
        return repository.existsById(id);
    }

    public Optional<T> getById(final Long id) {
        return repository.findById(id);
    }

    public List<T> getAll() {
        return repository.findAll();
    }

    public T update(final Long id, final T entity) {
        if (!repository.existsById(id)) {
            return null;
        }

        entity.setId(id);
        return repository.save(entity);
    }
}
```

Листинг 9: ExchangeTradingApplication

```
package com.kspt.exchangetrading;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.jpa.convert.threeten.Jsr310JpaConverters;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

import javax.annotation.PostConstruct;
import java.util.TimeZone;

@EnableJpaAuditing
@EntityScan(basePackageClasses = {
        ExchangeTradingApplication.class,
        Jsr310JpaConverters.class
})
@SpringBootApplication
public class ExchangeTradingApplication {

    @PostConstruct
    void init() {
        TimeZone.setDefault(TimeZone.getTimeZone("UTC"));
    }

    public static void main(String[] args) {
        SpringApplication.run(ExchangeTradingApplication.class, args);
    }

}
```

Листинг 10: admin.service

```
import {Injectable} from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {environment} from '../core/enviroment';
import {Admin, BankRecord, ClientRequest, Rate, Transaction} from '../models';
import {Broker} from '../models';

@Injectable()
export class AdminService {
    protected readonly http: HttpClient;

    constructor(http: HttpClient) {
        this.http = http;
    }

    checkRequests(adminId: number) {
        const url = this.urlWithAdminId(environment.checkAdminRequests, adminId);
        return this.http.get<ClientRequest[]>(url);
    }

    approveRequest(adminId: number, clientRequestId: number) {
        let url = this.urlWithAdminId(environment.approveRequest, adminId);
        return this.http.post<Transaction>(url, {clientRequestId});
    }

    declineRequest(clientRequestId: number) {
        const url = this.urlWithRequestId(environment.declineRequest, clientRequestId);
        return this.http.post(url, {});
    }

    getRates() {
        const url = environment.getRates;
        return this.http.get<Rate[]>(url);
    }

    getBankMoney() {
        const url = environment.getBankAssets;
        return this.http.get<BankRecord[]>(url);
    }

    getById(adminId: number) {
        const url = this.urlWithAdminId(environment.getAdminById, adminId);
        return this.http.get<Admin>(url);
    }

    getBrokers(adminId: number) {
        const url = this.urlWithAdminId(environment.getAdminBrokers, adminId);
        return this.http.get<Broker[]>(url);
    }

    private urlWithAdminId(urlWithoutId: string, adminId: number): string {
        return urlWithoutId.replace(':adminId', `${adminId}`);
    }

    private urlWithRequestId(urlWithoutId: string, clientRequestId: number): string {
        return urlWithoutId.replace(':clientRequestId', `${clientRequestId}`);
    }
}
```

Листинг 11: auth.service

```
import {Injectable} from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {environment} from '../core/enviroment';
import {Person} from '../models';

@Injectable()
export class AuthService {
    protected readonly http: HttpClient;

    constructor(http: HttpClient) {
        this.http = http;
    }

    signIn(login: string, password: string, personType: string) {
        const url = environment.signIn;
        return this.http.post<any>(url, {login, password, personType});
    }

    signUp(login: string, password: string, personType: string, name: string, surname: string) {
        const url = environment.signUp;
        return this.http.post<Person>(url, {login, password, personType, name, surname});
    }

    signOut(id: number, personType: string) {
        const url = environment.signOut;
        return this.http.post(url, {id, personType});
    }
}
```

Листинг 12: broker.service

```
import {Injectable} from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {environment} from '../core/enviroment';
import {Broker, ClientRequest} from '../models';

@Injectable()
export class BrokerService {
    protected readonly http: HttpClient;

    constructor(http: HttpClient) {
        this.http = http;
    }

    getById(brokerId: number) {
        const url = this.urlWithBrokerId(environment.getBrokerById, brokerId);
        return this.http.get<Broker>(url);
    }

    checkRequests(brokerId: number) {
        const url = this.urlWithBrokerId(environment.checkBrokerRequests, brokerId);
        return this.http.get<ClientRequest[]>(url);
    }

    validateClientRequest(brokerId: number, clientRequestId: number) {
        const url = this.urlWithBrokerId(environment.validateClientRequest, brokerId);
        return this.http.post<boolean>(url, {clientRequestId});
    }

    approveClientRequest(brokerId: number, clientRequestId: number) {
        let url = this.urlWithBrokerId(environment.approveClientRequest, brokerId);
        return this.http.post(url, {clientRequestId});
    }

    declineClientRequest(brokerId: number, clientRequestId: number) {
        let url = this.urlWithBrokerId(environment.declineClientRequest, brokerId);
        return this.http.post(url, {clientRequestId});
    }

    private urlWithBrokerId(urlWithoutId: string, brokerId: number): string {
        return urlWithoutId.replace(':brokerId', `${brokerId}`);
    }

}
```

Листинг 13: client.service

```
import {Injectable} from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {environment} from '../core/enviroment';
import {Agreement, BrokerageAccount, Client, ClientRequest, Transaction} from '../models';

@Injectable()
export class ClientService {
    protected readonly http: HttpClient;

    constructor(http: HttpClient) {
        this.http = http;
    }

    public getById(clientId: number) {
        const url = this.urlWithClientId(environment.getClientById, clientId);
        return this.http.get<Client>(url);
    }

    public setClientPassport(clientId: number, series: number, number: number) {
        const url = `${environment.getClientInfo}${clientId}/${environment.setClientPassport}`;
        return this.http.post<Client>(url, {series, number});
    }

    openBrokerageAccount(clientId: number) {
        const url = this.urlWithClientId(environment.openBrokerageAccount, clientId);
        return this.http.post<BrokerageAccount>(url, {});
    }

    closeBrokerageAccount(clientId: number) {
        const url = this.urlWithClientId(environment.closeBrokerageAccount, clientId);
        return this.http.post<boolean>(url, {});
    }

    putMoneyToAccount(clientId: number, money: number, currency: string) {
        const url = this.urlWithClientId(environment.putMoneyToAccount, clientId);
        return this.http.post<boolean>(url, {money, currency});
    }

    makeBrokerAgreement(clientId: number, validity: string) {
        const url = this.urlWithClientId(environment.makeBrokerAgreement, clientId);
        return this.http.post<Agreement>(url, {validity});
    }
```

```
43
44      extendBrokerAgreement(clientId: number, validity: string) {
45          const url = this.urlWithClientId(environment.extendBrokerAgreement, clientId);
46          return this.http.post<Agreement>(url, {validity});
47      }
48
49      breakBrokerAgreement(clientId: number) {
50          const url = this.urlWithClientId(environment.breakBrokerAgreement, clientId);
51          return this.http.post<boolean>(url, {});
52      }
53
54      exchangeMoneyForStocks(clientId: number, quantity: number, fromType: string, toType: string) {
55          const url = this.urlWithClientId(environment.exchangeMoneyToStocks, clientId);
56          return this.http.post<ClientRequest>(url, {quantity, fromType, toType});
57      }
58
59      exchangeStocksToMoney(clientId: number, quantity: number, fromType: string, toType: string) {
60          const url = this.urlWithClientId(environment.exchangeStocksToMoney, clientId);
61          return this.http.post<ClientRequest>(url, {quantity, fromType, toType});
62      }
63
64      getTransactions(clientId: number) {
65          const url = this.urlWithClientId(environment.getClientTransactions, clientId);
66          return this.http.get<Transaction[]>(url);
67      }
68
69      getTransactionById(clientId: number, transactionId: number) {
70          const url = this.urlWithClientIdAndPropertyId(environment.getClientTransactionById, clientId,
                transactionId);
71          return this.http.get<Transaction>(url);
72      }
73
74      getRequests(clientId: number) {
75          const url = this.urlWithClientId(environment.getClientRequests, clientId);
76          return this.http.get<ClientRequest[]>(url);
77      }
78
79      getRequestById(clientId: number, requestId: number) {
80          const url = this.urlWithClientIdAndPropertyId(environment.getClientRequestById, clientId,
                requestId);
81          return this.http.get<ClientRequest>(url);
82      }
83
84      private urlWithClientId(urlWithoutId: string, clientId: number): string {
85          return urlWithoutId.replace(':clientId', `${clientId}`);
86      }
87
88      private urlWithClientIdAndPropertyId(urlWithoutId: string, clientId: number, id: number): string {
89          return urlWithoutId
90              .replace(':clientId', `${clientId}`)
91              .replace(':id', `${id}`);
92      }
93  }
```

Листинг 14: store.service

```
1   import {Injectable} from '@angular/core';
2
3   @Injectable()
4   export class StoreService {
5
6       private id: number;
7       private propertyId: number;
8
9       constructor() {
10      }
11
12      getId() {
13          return this.id;
14      }
15
16      setId(id: number) {
17          this.id = id;
18      }
19
20      getPropertyId() {
21          return this.propertyId;
22      }
23
24      setPropertyId(propertyId: number) {
25          this.propertyId = propertyId;
26      }
27
28  }
```