МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение высшего образования
«Южный федеральный университет»

Институт высоких технологий и пьезотехники

Кафедра информационных и измерительных технологий

Дранкин Евгений Вячеславович

ОБУЧАЮЩИЙ БИРЖЕВОЙ ТОРГОВЫЙ ПРИВОД НА JAVA-ПЛАТФОРМЕ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ

РАБОТА БАКАЛАВРА

по направлению 09.03.02 – Информационные системы и технологии

Научный руководитель –

Жмайлов Борис Борисович

Ростов-на-Дону – 2018

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ВЫСОКИХ ТЕХНОЛОГИЙ И ПЬЕЗОТЕХНИКИ

Кафедра информационных и измерительных технологий

З А Д А Н И Е

на выпускную квалификационную работу

Студент гр. 4-3 Дранкин Е.В.

1. Тема: Разработка системы управления контентом сайта

2. Срок сдачи законченной работы 30 мая 2018г.

3. Исходные данные:

Государственные стандарты оформления библиографического описания литературы (ГОСТ 7.1-2003 «Библиографическая запись. Библиографическое описание. Общие требования и правила составления», ГОСТ 7.82-2001 «Библиографическая запись. Библиографическое описание электронных ресурсов», ГОСТ 7.12-1993 «Библиографическая запись. Сокращение слов на русском языке. Общие требования и правила», ГОСТ 7.11-1978 «Сокращение слов и словосочетаний на иностранных европейских языках в библиографическом описании»).

Основное требование — разработать систему представления данных в пользовательском интерфейсе.

4. Перечень вопросов, подлежащих разработке:

1) Анализ предметной области.
2) Обоснование технических решений.
3) Проектирование элементов системы.
4) Реализация компонентов системы.
5) Разработка методики тестирования.
6) Тестирование системы.

5. Перечень графического материала:

Подготовка графических материалов для презентации работы

6. Консультанты по работе:

_____

_____

7. Дата выдачи задания: 26 апреля 2018г.

8. Руководитель _____ Жмайлов Б.Б.

Подпись *ФИО*

9. Задание принято к исполнению

_____ _____

Дата *Подпись студента*

АННОТАЦИЯ

В работе рассмотрен процесс проектирования и разработки универсального торгового привода. При проектировании системы использовалась методология ООАП и RUP(Rational Unified Process — рациональный унифицированный процесс разработки) подход. В результате получены канонические UML диаграммы описывающие различные аспекты моделируемой системы. На основании полученных диаграмм была выполнена реализация системы средствами java. Также приведены результаты функционального и нагрузочного тестирования разработанной системы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....7
ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ.....13
1.1 Описательная постановка задачи.....13
1.2 Формальная постановка задачи.....14
1.3 Декомпозиция задачи.....14
1.4 Аналитический обзор существующих методов решения данной проблемы.....16
1.5 Функциональные свойства приложения.....17
ГЛАВА 2 ПРОЕКТИРОВАНИЕ.....18
2.1 Основания для разработки технического задания.....18
2.2 Оценка и выбор перспективных направлений разработки.....18
2.3 Обоснование выбора инструментальных средств.....18
2.4 Проектные решения.....19
2.4.1 Обоснованность разделения приложения на клиента и сервер….19
2.4.2 Диаграмма вариантов использования.....19
2.4.3 Диаграмма классов.....25
2.4.4 Диаграмма компонентов.....27
2.4.5 Диаграмма деятельности.....30
ГЛАВА 3 РЕАЛИЗАЦИЯ.....32
3.1 Реализация основных функций.....32
3.2 Отображение сущностей на базу данных.....37
3.3 Реализация взаимодействия с базой данных.....37
3.4 Реализация интерфейса пользователя.....39
ГЛАВА 4 ТЕСТИРОВАНИЕ.....46
4.1 Тестирование CRUD-операций.....46
4.2 Тестирование REST API.....50
4.3 Разработка программы испытаний тестирования конечного продукта 51

ЗАКЛЮЧЕНИЕ.....54
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....55
Приложение А ТЕХНИЧЕСКОЕ ЗАДАНИЕ.....57
Приложение Б ИНСТРУКЦИЯ СИСТЕМНОГО АДМИНИСТРАТОРА…59
Приложение В ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ.....60

ВВЕДЕНИЕ

Развитие информационных технологий положило начало для появления и развития различных направлений человеческой деятельности, которые зачастую напрямую не были связаны с информационными технологиями, а лишь пользовались их возможностями. Одним из этих направлений стал Трейдинг.

Трейдинг — непосредственная работа трейдера: анализ текущей ситуации на рынке и заключение торговых сделок.

Тре́йдер (от англ. Trader — торговец) — торговец, действующий по собственной инициативе и стремящийся извлечь прибыль непосредственно из процесса торговли. Обычно подразумевается торговля ценными бумагами (акциями, облигациями, фьючерсами, опционами) на фондовой бирже. Трейдерами также называют торговцев на валютном (форекс) и товарном рынках (например, «зернотрейдер»). Торговля осуществляется трейдером как на биржевом, так и на внебиржевом рынках[1].

Не следует путать трейдера с другими торговцами, которые проводят сделки по заявкам клиентов или в их интересах (дилер, брокер, дистрибьютор).

Виды трейдеров:

1. По форме собственности:
    a. Профессиональные торговцы — работают в финансовых учреждениях или предприятиях (банки, страховые компании, ПИФы, брокеры, дилеры). Обычно имеют специализированное образование и лицензию на соответствующую деятельность. Выполняют операции за деньги и в интересах своих компаний или их клиентов. По российскому законодательству такие торговцы обязаны иметь персональные аттестаты (ранее их выдавала ФСФР, ныне этим ведает Банк России).

b. Частные торговцы, независимые трейдеры — выполняют операции за свои деньги и в своих интересах (работают на себя), для доступа к системам торговли пользуются услугами посредников (брокеров, дилеров). Проводимые ими операции обычно не требуют лицензирования. Часто не имеют специализированного образования, пользуются услугами консультантов, в том числе профессиональных торговцев.

2. По целям сделок:

a. Работа — обеспечение проведения иных операций или исполнение заявок клиентов (например, покупка на бирже валюты для оплаты закупки оборудования или продажа валютной выручки для возможности выплаты заработной платы). Обычно это выполняют профессиональные торговцы.

b. Инвестор — рассматривает сделку в качестве инвестиции.

c. Спекулянт — сделка ради извлечения прибыли из разницы цен.

d. Арбитражёр — заключает встречные сделки (одна покупка, другая продажа) со связанными инструментами с целью получения прибыли на движении цен одного актива относительно другого. Общее рыночное движение цен конкретного актива нивелируется.

e. Хеджер — сделка заключается ради уменьшения или фиксации уровня риска, например, риска изменения закупочных цен на сельхозпродукцию или валютных котировок. Чаще всего применяется товаропроизводителями в форме опционов или фьючерсов для обеспечения возможности финансового планирования внутри производственного цикла.

3. По расположению рабочего места:

a. Трейдер на полу, трейдер в яме — обычно это внутридневные частные торговцы, торгующие непосредственно в биржевом зале. Их рабочее место расположено в самой низкой точке биржевого зала (в яме). Обычно они заключают сделки только по одной и той же ценной бумаге. До компьютеризации торговли их плохо было видно, поэтому аренда места «на полу» стоила значительно меньше, чем на ступеньках амфитеатра биржевой ямы. Трейдер на полу заключает сделку в надежде, что через минуты или даже секунды сможет приобрести возмещающий контракт и получить с этого небольшую прибыль. Например, на рынках зерна трейдеры на полу часто входят в сделку ради разницы в 0,0025 доллара за бушель.

b. Трейдер в зале — обычно это профессиональные торговцы, представляющие интересы большого числа клиентов или крупные заявки. Их рабочие места располагались выше уровня пола биржевой ямы, их лучше было видно, им лучше было видно не только других торговцев, но и информационные мониторы.

c. Трейдер у монитора торговлю ведёт через специализированные торговые терминалы, которые позволяют видеть заявки других трейдеров и выставлять собственные, читать новости, просматривать историю котировок, производить её математический анализ и строить различные графики. Не требуется личного присутствия в биржевом зале. Ликвидируется разница между трейдерами на полу и в зале. В последнее время в качестве канала связи торгового терминала с брокером или непосредственно с биржей используется Интернет. Именно Интернет-трейдинг сейчас является наиболее распространённой формой торговли.

4. По длительности:

   a. Дневной трейдер (дейтрейдер) — заключает встречные сделки внутри одного торгового дня (одной торговой сессии), закрывает все позиции перед закрытием операционного дня. Часто имеет относительно небольшой капитал. Закрытие позиций обычно мотивируется опасением гэпов («разрывов» между ценой закрытия предыдущего и ценой открытия нового торгового дня)[2].

   b. Скальпер, Пипсовщик — совершает большое количество сделок малой продолжительности: от нескольких секунд до десятка минут (скальпинг). Как правило, результативность отдельной сделки невелика, но велико число сделок (см. также Высокочастотный трейдинг).

   c. Позиционный трейдер (краткосрочный) — заключает сделки, предполагая закрытие позиций через несколько дней, закрывает все позиции перед периодами уменьшения ликвидности (праздниками, летними каникулами и т. п.)

   d. Среднесрочный трейдер — заключает несколько сделок в год, закрывает позиции при изменении недельных трендов.

   e. Долгосрочный инвестор — открытые позиции могут держаться несколько лет, закрывает позиции при изменении глобальных трендов.

Считается, что дневные и позиционные трейдеры больше опираются на технический анализ рынков, а среднесрочные и долгосрочные инвесторы — на фундаментальный анализ[1].

В ходе развития информационных технологий появилась возможность работы людей с финансовыми активами в режиме “здесь и сейчас”. Классическое инвестирование, имеющее целью инвестиционную деятельность на долгий срок всегда будет иметь место, но с развитием

информационных технологий всё большую популярность начал получать скальпинг(одна из стратегий внутридневных спекулятивных операций на фондовом, валютном, товарном рынках, особенностью которых является закрытие сделки при достижении небольшой прибыли в несколько пунктов. Как правило, сделка при этом совершается в небольшой промежуток времени, типичной длительностью от нескольких минут в конце XX века до долей секунды в XXI веке[3]) или краткосрочные(спекулятивные) сделки. Появилась возможность заключать краткосрочные сделки на покупку и продажу с возможностью избавиться от финансового инструмента в любой удобный момент времени.

Сделки стали заключаться не на несколько лет, целью которых был рост курса актива, получение дивидендов или купонов и порой возможность вносить свой голос в собраниях акционеров. Появилась возможность краткосрочных инвестиций, когда покупался актив на несколько месяцев или несколько дней, и возможность скальпинга, когда актив приобретался на несколько секунд или минут с целью получения прибыли максимально быстро.

В настоящий момент существуют 3 основных метода скальпинга[3]:

1. Стаканный (классический) скальпинг заключается в определении дисбаланса между объёмом спроса и предложения, способного привести к направленному движению котировок, даже незначительному. Распространен на инструментах, имеющих конкретную базу (фьючерсы на акции).

2. Импульсный скальпинг заключается в постоянной оценке внешних рынков и инструментов, способных вызвать импульсивное направленное движение торгуемого инструмента (при торговле в России оценивается движение американских фьючерсов, европейских фьючерсов,

нефти, доллара и прочее). Распространён на фьючерсах на фондовые индексы.

3. Гибридный скальпинг сочетает черты первых двух методов.

В связи с развитием этой сферы появилась острая нужда людей в обучении трейдинговому делу и в особенности скальпингу, где необходимо уметь применять технический анализ рынка и заключать проводить ордера максимально быстро, но в силу рабочей занятости некоторого слоя населения днём, они порой не могут себе позволить выходить на рынок в часы его работы.

ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ

1.1 Описательная постановка задачи

Необходимо создать скальперский(торговый) привод, направленный на обучение пользователей.

Скальперский(торговый) привод — это софт (или патч к соответствующему торговому терминалу) с удобным пользовательским интерфейсом для совершения быстрых сделок. Скальпинг — особый вид трейдинга. Человек, который им занимается, редко планирует находиться в позиции даже 1-2 минуты. В большинстве случаев это время гораздо меньше и на ликвидных инструментах исчисляется лишь несколькими секундами. В связи с этим скорость выполнения операций является для скальперов основным фактором успешной торговли. Чтобы упростить процесс ввода заявки, трейдеры используют специальные программы — приводы для скальпинга[4].

Торговый привод будет направлен на обучение пользователей торговле при помощи графика и стакана цен.

Стакан цен — это торговая площадка, на которой отображаются все имеющиеся сделки по купле и продаже. На графике равномерное накопление имеет сходство с обычным кухонным стаканом. Им необходимо воспользоваться по разным причинам:

- упрощение анализа различной информации;
- расширение возможностей для торговли при высокой ликвидности;
- использование новых инструментов для решения поставленных задач[5].

Алгоритм генерации цен будет работать, основываясь на значениях цен, прошедших ранее. Это позволит пользователю на практике попробовать использовать теоретические знания, полученные ранее, для прогнозирования дальнейшего ценообразования выбранного актива.

Поэтому целью создания приложения является реализация торгового привода, работающего независимо от фондовой биржи, который позволит её пользователям изучить и понять работу фондового рынка, научит строить прогнозы движения активов.

1.2 Формальная постановка задачи

В данной задаче рассматривается определённое число объектов:

- Пользователи — информация о пользователях;
- Счёт — информация о счёте пользователя;
- Сделки — информация о прошедших сделках пользователя;
- Тариф — текущий тариф пользователя;
- Заявки — текущие и отложенные сделки пользователя;
- Тренд — движение тренда.

Каждый из данных объектов имеет проекцию в базе данных. Для объекта тренд будет создан специфический алгоритм, который будет, основываясь на прошедшем тренде, генерировать значения сделок для дальнейшего движения тренда, который в свою очередь будет принимать вид флета(преимущественно горизонтальное движение) или тренда(преимущественно вертикальное движение).

Требуется также создать пользовательский интерфейс, который позволит просматривать пользователю движение цен и стакана заявок в режиме текущего времени.

Требуется, чтобы приложение работало на платформах Windows и Linux. Минимальные системные требования: 500МБ ОЗУ, 100МБ ПЗУ, Процессор 1.3ГГц, Скорость интернет-соединения 1Мб/с. Требуется, чтобы пользователь мог работать с пользовательским интерфейсом, мышкой и клавиатурой.

1.3 Декомпозиция задачи

Все вышеизложенные задачи будут разрабатываться по следующему примерному плану:

1) Разработка сущностей.
2) Определение основных классов и их создание.
3) Разработка DataProvider для работы с csv, xml и DataBase.
4) Разработка CLI.
5) Разработка клиентской части:
    a) Выбор необходимых библиотек;
    b) Изучение графических библиотек.
6) Создание графического интерфейса:
    a) Создание первой графической части приложения — графика японских свечей с минутным таймфреймом;
    b) Создание первой графической части приложения — стакана заявок.
7) Обеспечение взаимодействия графического интерфейса с csv:
    a) Обеспечение взаимодействия графика;
    b) Обеспечение взаимодействия стакана заявок.
8) Разработка rest:
    a) Выбор необходимых библиотек;
    b) Создание rest.
9) Обеспечение взаимодействия между сервером и клиентом.
    a) Связывание стакана заявок с серверной частью приложения, с использованием API серверной части;
    b) Связывание графика цен с серверной частью приложения, с использованием API серверной части.
10) Определение с местом работы основных генерационных алгоритмов.
11) Реализация простейшего алгоритма, генерирующего числа.
12) Обеспечение взаимодействия генерационного алгоритма с базой данных и UI.

1.4 Аналитический обзор существующих методов решения данной проблемы

Как показал обзор решений данной предметной области, в текущий момент существуют следующие варианты решения поставленной задачи:

EasyScalp — современный торговый терминал, разработанный для скальпинга и торговли внутри дня. Терминал имеет простой и понятный интерфейс в сочетании с множеством настроек. Практически каждый элемент терминала может быть настроен под себя. EasyScalp позволяет быстро совершать сделки горячими клавишами и мышью. Сразу после открытия позиции вы будете видеть её в стакане. Открытая позиция может быть защищена автоматически выставленным стоп-лоссом. Стоп-лосс и тейк-профит можно быстро выставить горячими клавишами и переставлять с помощью мыши. Заявки выставляются одним щелчком мыши по стакану, а выставленные заявки можно перемещать мышью[6]. Предоставляется бесплатная пробная лицензия для работы с данной платформой сроком на 14 дней. Она даёт доступ к полному функционалу с ограничением истории в 7 дней. Системные требования: Windows 7 SP1, 8, 8.1, 10 (x64); .NET Framework 4.6; Торговая платформа QUIK, Plaza2, SmartCOM, Transaq, MetaTrader 51[7].

Торговый привод QScalp — это инструмент для анализа и скоростного выполнения операций на рынке при краткосрочной и высокочастотной биржевой торговле. Программа обеспечивает наглядное представление текущей рыночной ситуации и позволяет выполнять комплексные торговые операции в одно нажатие. Благодаря предоставляемым ею возможностям вы сможете в прямом смысле держать руку на пульсе рынка и эффективно реализовать практически любую торговую стратегию[8]. Бесплатный срок использования функционала привода предоставляется сроком на 15 дней. Системные требования:

установленный пакет Microsoft .NET Framework 4 Client Profile, биржевой терминал QUIK версии 5.17 или более поздней[9].

TradingView — веб-сервис и социальная сеть для трейдеров, в основе которой лежит платформа технического анализа. Проект был запущен в сентябре 2011 года. TradingView доступен как в платном, так и в ограниченном бесплатном варианте[10].

Представленные приложения предлагаю решения данной проблемы только используя стакан заявок цен или же при помощи графика.

1.5 Функциональные свойства приложения

Приложение будет иметь следующие функциональные возможности: Просмотр в графическом виде в графике цен и стакане заявок данных числовых, возможность работы с графическим видом графика цен.

Также будет предоставляться возможность взаимодействия приложения с различными хранилищами данных, такими как csv и xml.

ГЛАВА 2 ПРОЕКТИРОВАНИЕ

2.1 Основания для разработки технического задания

Конечный продукт, которым является торговый привод, должен быть похож на основные торговые приводы, которые используются в настоящий момент различными брокерскими компаниями.

Он должен позволяет наглядно увидеть движение рынка ценных бумаг на данном примере, который реализуется разработчиком и с этой разработанной системой должен иметь возможность взаимодействовать трейдер, который в свою очередь имеет целью обучить тот контингент людей, которые заинтересованы в изучении и понимании движения фондовых рынков.

2.2 Оценка и выбор перспективных направлений разработки

Решение разрабатывать систему без использования ORM-средств было принято исходя из того, что платформа требует высокое быстродействие и время отклика, поэтому все функции по взаимодействию с базами данных, серверной и клиентской частью были реализованы вручную.

2.3 Обоснование выбора инструментальных средств

Принято было решение разрабатывать серверную и клиентскую часть на Java, т.к. это в настоящий момент наиболее востребованный и распространённый язык программирования, на который написано огромное множество библиотек, которые помогут быстро и легко реализовать различные решения.

СУБД была выбрана PostgreSQL, т.к. она относится к типу программного обеспечения OpenSource, имеет высокое быстродействие, поддерживает целостность базы данных.

Для создания графика цен используется библиотека JFreeChart, потому как она полностью удовлетворяет условиям реализации графика цен активов в виде графика японских свечей.

Стакан заявок создавался при помощи библиотеки JTable, которая позволила представить необходимые для трейдинга числовые данные в требуемом для торговли виде.

Для создания rest, который позволил создать взаимодействие между серверной и клиентской частью, был использован веб-микрофреймворк для Java Spark. Он позволил быстро создать API, который в дальнейшем использовался для связывания между собой rest и ui.

Клиент и сервер являются maven-проектами, который позволил в автоматическом режиме производить всю сборку необходимых для работы компонентов.

2.4 Проектные решения

2.4.1 Обоснованность разделения приложения на клиента и сервер

Предполагается, что приложение будет работать на множестве устройств, где все данные будут централизированы. Основные генерационные алгоритмы должны быть доступны для общего круга пользователей и подчиняться общим правилам, которые будут реализованы на сервере.

2.4.2 Диаграмма вариантов использования

Диаграмма вариантов использования (use case diagram) — диаграмма, на которой изображаются отношения между актерами и вариантами использования.

Вариант использования (use case) — внешняя спецификация последовательности действий, которые система или другая сущность могут выполнять в процессе взаимодействия с актерами .

Актер (actor) — согласованное множество ролей, которые играют внешние сущности по отношению к вариантам использования при взаимодействии с ними[11].

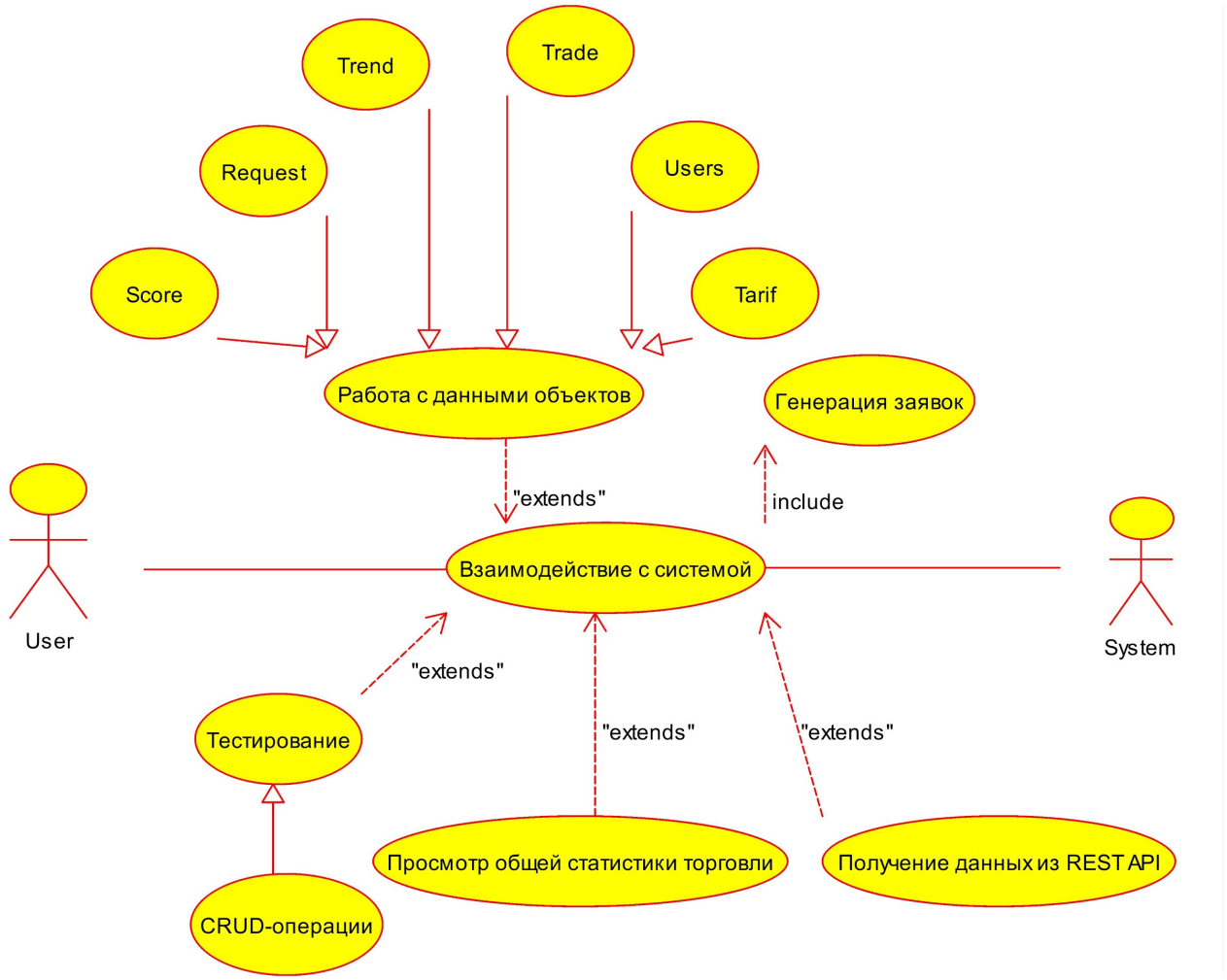

Рисунок 1. Диаграмма вариантов использования серверной части.

Таблица 1. Детализация вариантов использования

Вариант использования	Описание
Взаимодействие с системой	Подключение к системе Выполнение возможных действий
Работа с данными объектов	Получение списка элементов Выбор из предложенного списка элементов
Просмотр общей торговой статистики	Получение данных Выбор результата
Получение данных из REST API	Обращение по URI Получение списка объектов

	Преобразование списка объектов в Json Передача Json
Генерация заявок	Получение списка заявок Генерация случайных в диапозоне Изменение данных заявки по определённой цене в определённом объёме Сохранение изменённых данных
Тестирование	Тестирование основных CRUD-операций

На основе требований технического задания, была создана диаграмма использования, рис.1. Детализация вариантов использования представлена в табл.1.

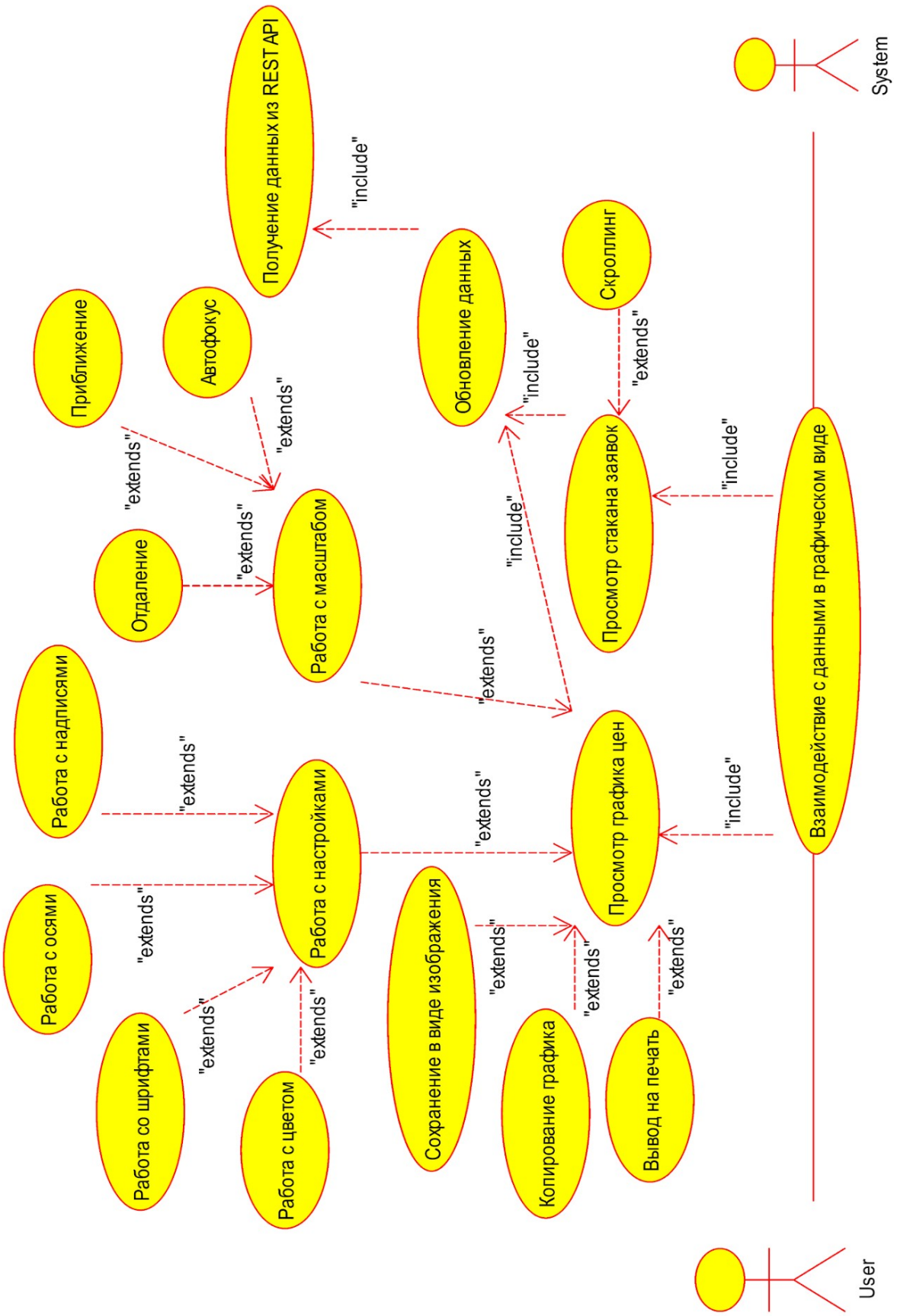

Рисунок 2. Диаграмма вариантов использования клиентской части.

Таблица 2. Детализация вариантов использования

Вариант использования	Описание
Просмотр графика цен	Получение данных Вывод их на форму
Просмотр стакана заявок	Получение данных Вставка в список таблиц Вывод/Обновление списка
Получение данных из REST API	Обращение к REST по URI Получение Json Преобразование данных из Json в список объектов
Скролинг	Выбор стакана заявок Прокрутка таблицы данных
Работа с масштабом	Выбор графика цен Вызов контекстного меню Изменение масштаба
Работа с масштабом	Выбор графика цен Изменение масштаба с помощью колеса прокрутки
Работа с масштабом	Выбор графика цен Выделение необходимого диапазона графика
Работа с настройками	Выбор графика цен Вызов контекстного меню Выбор меню “Настройки...”
Работа с надписями	Выбор меню “Настройки…” Изменение названия графика Изменение названия меток
Работа с осями	Выбор меню “Настройки…” Работа с осевыми отметками
Работа со шрифтами	Выбор меню “Настройки…” Изменение шрифтов названия графика

	Изменение шрифтов меток Изменение шрифтов отметок
Работа с цветом	Выбор меню “Настройки…” Изменение цвета графика заявок
Сохранение в виде изображения	Выбор графика цен Вызов контекстного меню Выбор пункта меню “Сохранить как…”
Копирование графика	Выбор графика цен Вызов контекстного меню Выбор пункта меню “Копировать”
Вывод на печать	Выбор графика цен Вызов контекстного меню Выбор пункта меню “Печать…”

На основе требований технического задания, была создана диаграмма использования для клиентской части, рис.2. Детализация вариантов использования представлена в табл.2.

2.4.3 Диаграмма классов

Диаграмма классов (англ. Static Structure diagram) — диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними[11].

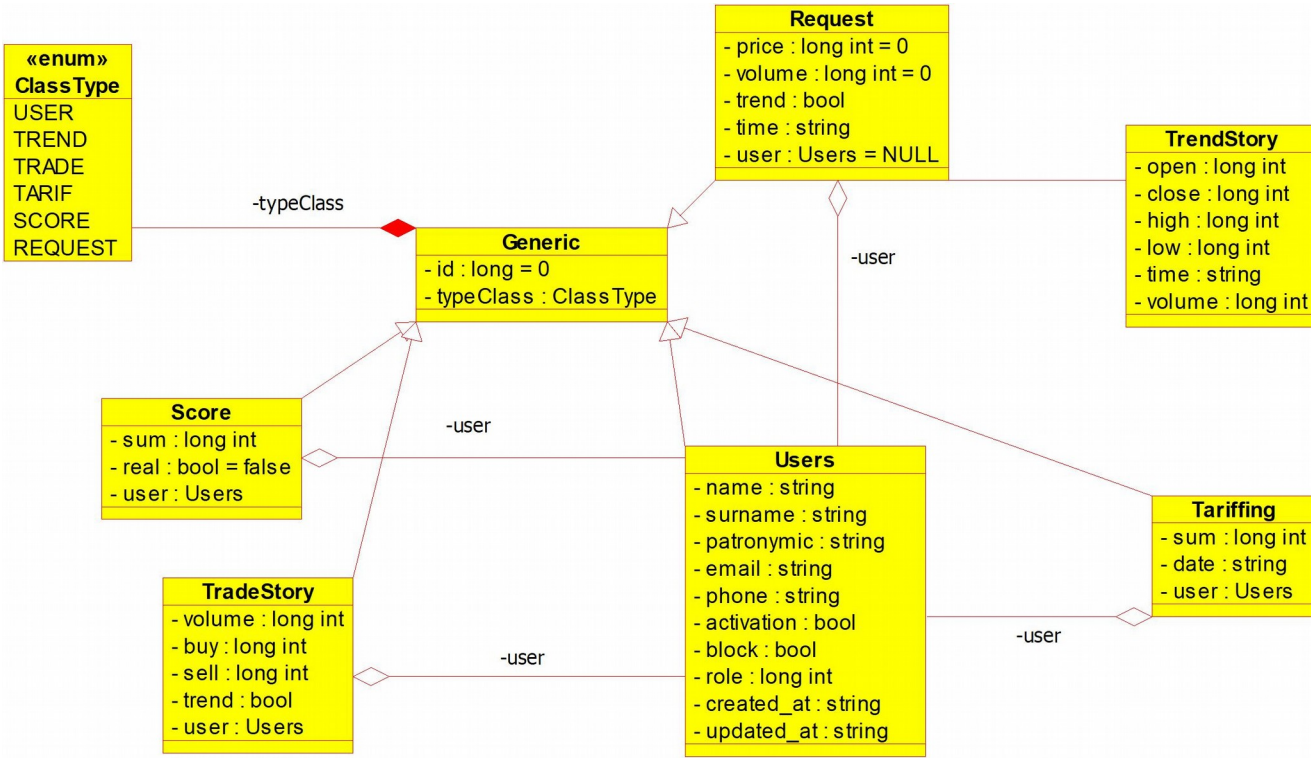

Рисунок 3. Диаграмма классов

Диаграмма классов, представленная в рис.3, построена на основе диаграммы вариантов использования. Класс Generic состоит в отношениях наследования со всеми основными объектами, содержит информацию о типе объекта и его идентификаторе. Сущности Пользователь, Оплата тарифа, Счет, Заявки и История тренда являются наследниками класса Generic.

Стержневые сущности:

1. Пользователи(Users) содержит информацию о пользователе: ФИО пользователя, email, телефон, статус активации, статус блокировки, роль, дата создания и обновления записи. Класс Пользователи является наследником класса Geneic. Класс Пользователи имеет связь один-ко-

многим со следующими классами: Счет, История сделок, Оплата тарифа и Заявки.

2. История тренда(TrendStory) содержит информацию о прошедших объёмах в каждую минуту времени по активу, его цену открытия, закрытия, наибольшую и наименьшую цену в течение прошедшей минуты. Класс История тренда является наследником класса Generic. Класс История тренда не связан ни с одним из других классов напрямую, но данные формируются на основе проходящих данных в классе Заявки каждую минуту.

Характеристические сущности:

1. Счет(Score) содержит информацию о состоянии счёта(сумма средств) пользователя в данный момент, тип счёта(демо или нет) и id пользователя. Счет является наследником класса Generic. Связан с классом Пользователи многие-к-одному.

2. История сделок(TradeStory) содержит информацию об истории сделок, проведённых определённым пользователем. Имеет информацию об объёме проведённой сделки, о цене покупки и продажи актива, о направлении сделки(long или short) и id пользователя, проведённого сделку. История сделок является наследником класса Generic. Связан с классом Пользователи многие-к-одному.

3. Оплата тарифа(Tariffing) содержит информации обо всех оплатах тарифа пользователем. Имеет сумму оплаты, дату оплаты и id пользователя, произвёвшего оплату. Оплата тарифа является наследником класса Generic. Связан с классом Пользователи многие-к-одному.

4. Заявки(Request) содержит ордера на покупку или продажу по текущему активу. Имеет информацию о цене покупки/продаже актива, объёме ордера, о направлении сделки(покупка/продажа), о времени назначения сделки и id пользователя. Заявки являются наследником класса Generic. Связан с классом пользователи многие-к-одному.

Отношения между классами серверной части:

- Стержневая сущность Пользователи(Users) связана со всеми характеристическими классами по полю user_id, которое указывает какому пользователю принадлежит каждая из записей.
- Сущность История тренда(TrendStory) связана с сущностью Заявки(Request) косвенно. В ходе прохождения заявок в течение минутного таймфрейма формируется класс TrendStory с определёнными полями.
- Класс Generator, который создаётся в отдельном потоке и генерирует случайное число заявок со случайным объёмом проходимых заявок(Request) и по окончанию прохождения генерации создаёт новый класс TrendStory, который сохраняет в базу данных.
- Классы UserService и UserController, которые позволяют через REST API получать данные стержневых и характеристических сущностей, сохранённых в базе данных.

Диаграмма классов клиентской части является такой же, как и диаграмма классов серверной части(рис. 3), т.к. необходимо будет преобразовывать данные из Json в объекты данных классов, которые существуют в серверной части приложения.

Отношения между классами клиентской части:

- Класс FxMarketPxFeeder посылает POST-запросы на сервер для получения ответа в формате Json, который содержит данные о Заявках(Request) и Истории тренда(TrendStrory). Всё это происходит в отдельном потоке. После получения данных они преобразуются в объекты соответствующих типов и далее они преобразуются при помощи библиотек JTable и JFreeChart в графическое представление.

2.4.4 Диаграмма компонентов

Диаграмма компонентов, Component diagram — статическая структурная диаграмма, показывает разбиение программной системы на

структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т.п.[11]

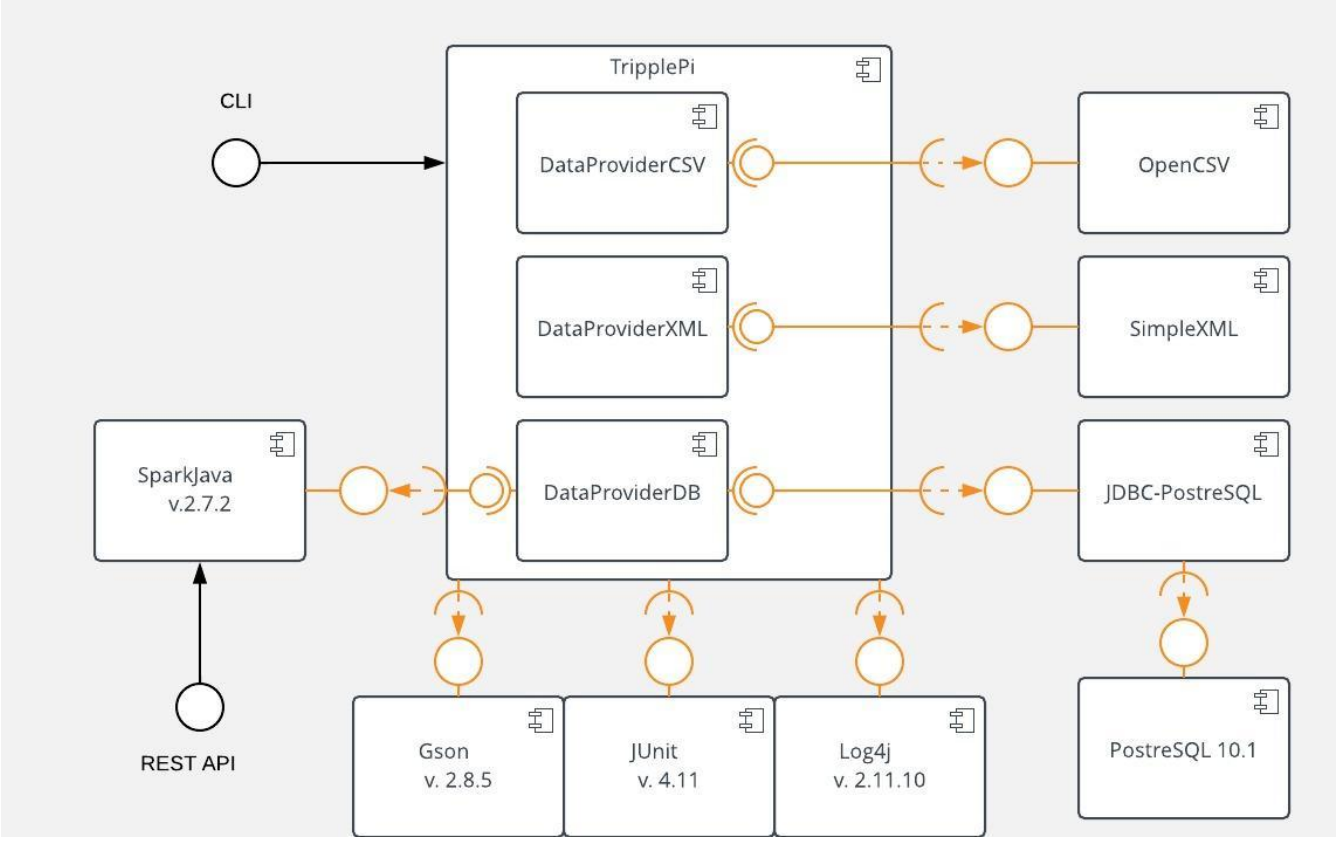

Рисунок 4. Диаграмма компонентов серверной части.

На диаграмме компонентов(рис. 4) представлен набор библиотек, используемых в данном приложении:

OpenCSV — библиотека для работы с csv-файлами.

SimpleXML — библиотека для работы с xml-файлами.

JDBC (Postgres) — библиотека для работы с базами данных. В данном случае используется спецификация, предназначенная для работы с PostgreSQL.

Log4j — библиотека для журнализации событий.

JUnit — библиотека для создания тестов.

Gson — Google-библиотека, позволяющая работать с Json.

SparkJava — веб-микрофреймворк, предназначенный для создания REST API[12].

Также приложение предоставляет CLI.

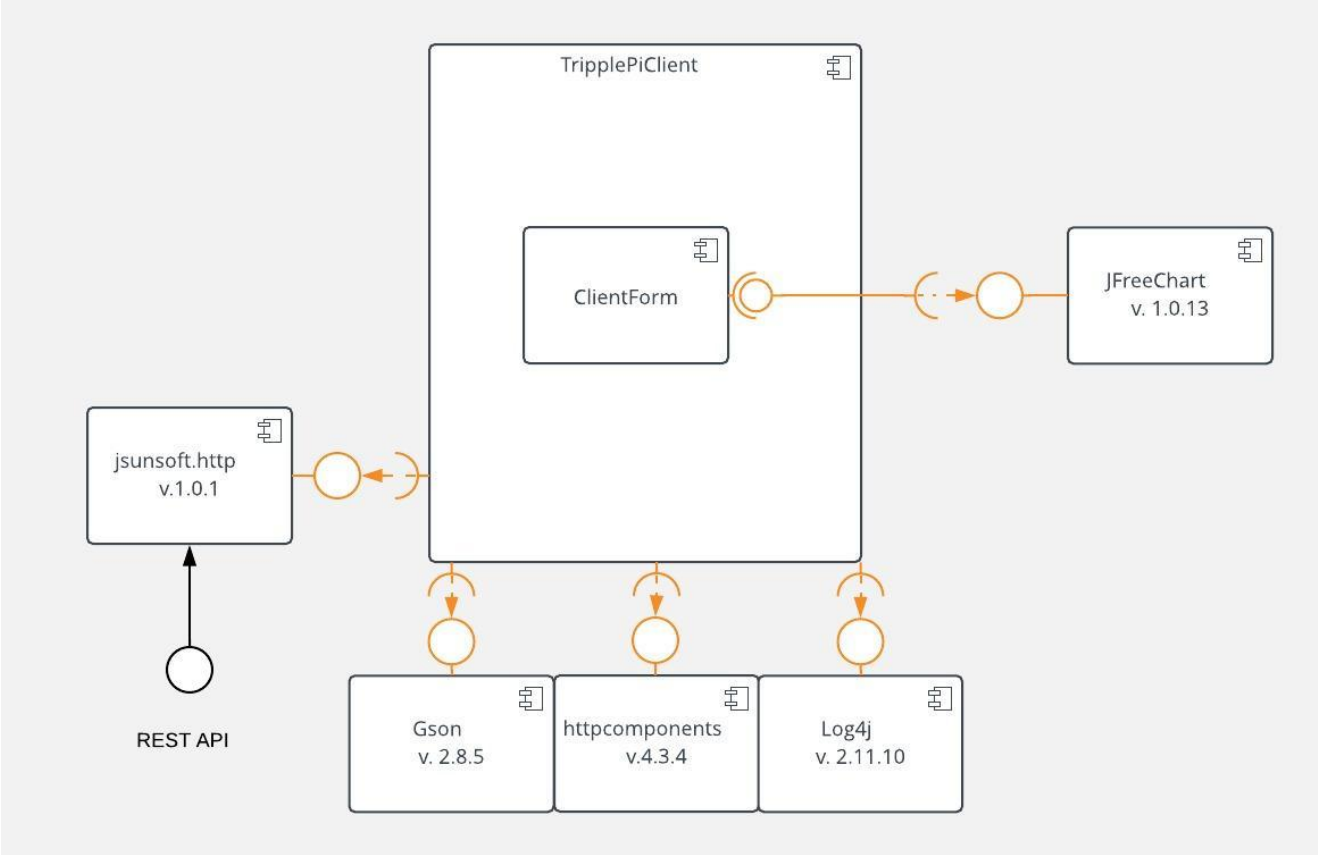

Рисунок 5. Диаграмма компонентов клиентской части.

На диаграмме компонентов(рис. 5) представлен набор библиотек, используемых в клиентской части данного приложения:

JFreeChart — компонент, позволяющий реализовать график цен;

JsunSoft — позволяет работать с серверной частью, обращаясь к ней и получая ответ.

Gson — позволяет преобразовывать получаемый Json к списку объектов.

Httpcomponents — позволяет взаимодействовать с серверной частью, используя различные запросы.

Log4j — библиотека для журнализации событий.

2.4.5 Диаграмма деятельности

Диаграмма деятельности (англ. activity diagram) — UML-диаграмма, представляющая собой набор действий. Под деятельностью (англ. activity) понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий англ. action, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Диаграммы деятельности используются при моделировании бизнес-процессов, технологических процессов, последовательных и параллельных вычислений[11].

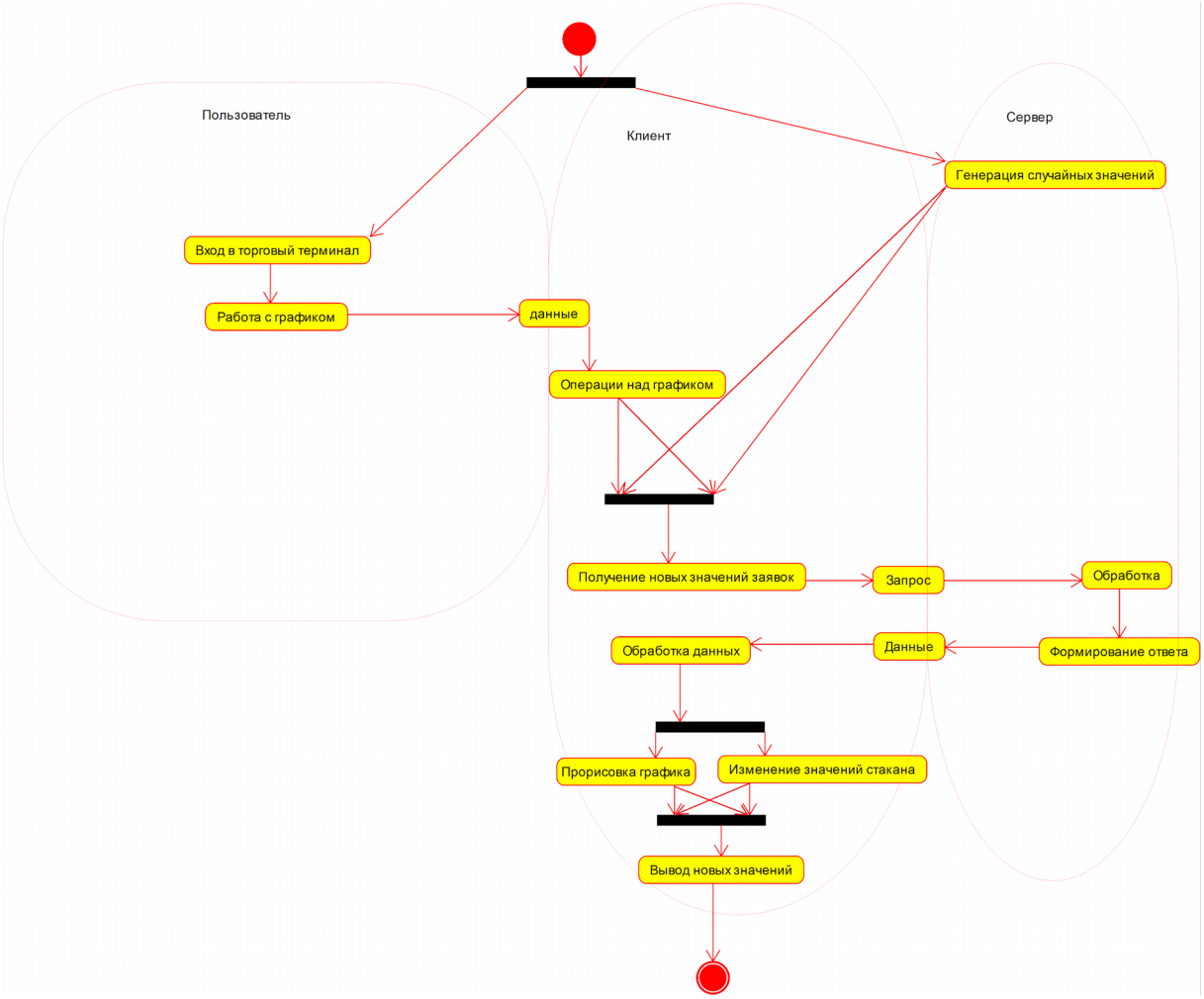

Рисунок 6. Диаграмма деятельности торговли

На диаграмме деятельности торговли(рис. 6) указываются действия, которые будут происходить во время работы пользователя с платформой. Также указанные действия на этой диаграмме в большинстве своём являются полностью асинхронными, т.е. операции с графиком и генерация новых данных между собой никак не связаны и могут происходить как одновременно, так и в различные промежутки времени, не влияя друг на друга.

ГЛАВА 3 РЕАЛИЗАЦИЯ

3.1 Реализация основных функций

Процесс реализации состоял из следующих этапов:

1. Установка основной платформы разработки — Java 8.
2. Установка средства разработки приложения — IDE NetBeans 8.
3. Установка средства для автоматизации сборки проекта — Apache Maven.
4. Для серверной части: подключение Log4j, simple-xml, open-csv, JDBC, JUnit, Spark и Gson.
5. Для клиентской части: подключение Log4j, JFreeChart, HttpComponents, Gson, Http-Request.

Подключение происходит при помощи указания библиотек и их расположения в сети, в файле pom.xml, на основе которого происходит подгрузка необходимых файлов библиотек из указанных путей и собирается maven-проект. Рассмотрим на примере подключения библиотеки openCSV:

```
<dependency>
        <groupId>com.opencsv</groupId>
        <artifactId>opencsv</artifactId>
        <version>4.0</version>
        <type>jar</type>
</dependency>
```

groupId — наименование организации или подразделения и обычно действуют такие же правила как и при именовании пакетов в Java — записывают доменное имя организации или сайта проекта.

artifactId — название проекта.

version — указывается используемая версия проекта.

type — указание на использование подключаемого проекта.

6. Настройка журнализации событий.

Настройка происходит при помощи указания всех необходимых инструкций файле log4j.properties и объявлении переменной логирования в необходимых классах.

7. Подключение файлов enviroment.properties и ConfigurationUtil.

В файле Constants указываются статические константы, которые в свою очередь указывают на именованные свойства, прописанные в enviroment.properties в следующем виде:

<именованное свойство> = <значение>.

Значение свойства приложение получает при помощи класса ConfigurationUtil.

8. Проектирование приложения.

Создание необходимых UML-диаграмм: диаграммы использования, классов, деятельности и компонентов.

9. Создание бинов.

Генерация классов средствами Umbrello на основе составленной диаграммы классов. Генерация setters и getters средствами IDE NetBeans.

10. Создание DataProviders.

Создание класса Result, содержащего status типа StatusType и errorMessage типа String. Создание интерфейса IDataProvider.

В интерфейсе будут записаны следующие методы:

- public Result insert(List<Generic> object) — метод, предназначенный для добавления новых записей.
- public Result update(Generic object) — метод, предназначенный для обновления имеющейся записи.
- public Result delete(Generic object) — метод, предназначенный для удаления имеющейся записи.
- public Optional<List<Generic>> select(ClassType type) — метод, предназначенный для извлечения записей из источника данных.

- public Generic getObjectByID(long id, ClassType type) — метод, предназначенный для получения объекта определённого класса по id.
- public Generic getRequestByID(long id) — метод, предназначенный для получения объекта класса Request по id.
- public Generic getScoreByID(long id) — метод, предназначенный для получения объекта класса Score по id.
- public Generic getTarifByID(long id) — метод, предназначенный для получения объекта класса Tariffing по id.
- public Generic getTradeByID(long id) — метод, предназначенный для получения объекта класса TradeStory по id.
- public Generic getTrendByID(long id) — метод, предназначенный для получения объекта класса TrendStory по id.
- public Generic getUserByID(long id) — метод, предназначенный для получения объекта класса Users по id.

Создание классов DataProviderCsv, DataProviderXml, DataProviderDB, реализующих интерфейс IDataProvider. В этих DataProvider реализовано каскадное удаление. В DataProviderCsv и DataProviderXml каскадное удаление и целостность данных реализованы на уровне бизнес-логики. В DataProviderDB каскадное удаление и целостность данных реализованы на уровне базы данных.

11. Создание CRUD-тестов.

Создание CRUD-тестов для каждого из методов каждого DataProvider, направленных для работы с каждым из бинов. Для каждого DataProvider тесты располагаются в отдельном пакете. Например, test: ru.sfedu.tripplepi.dao.csv.

Тестирование каждой CRUD-операции располагается в отдельном классе теста. Формируется имя класса следующим образом: DataProvider<имя источника данных>[1..4]<Имя тестируемого метода>Test.

Для тестирования каждого из бинов предполагается отдельный метод в классе теста. Формируются методы следующим образом:

test<Название метода теста>[1..6]<Имя класса>.

12. Создание CLI.

Создание возможности работать с программой через командную строку. CLI поддерживает возможность работы с различными источниками данных, такими как csv, xml и DataBase. Через пробел вводятся данные, указывающие на источник используемых данных, бин, с которым будет проводиться работа и метод. Например: csv User select.

13. Создание REST API

Создание REST API, с использованием веб-микрофреймворка Spark.

14. Создание формы с графиком и стаканом заявок.

Создание пользовательской формы с графиком цен, созданным на JFreeChart и стакана заявок, созданного при помощи JTable. Обеспечение их функционирования в отдельном потоке.

15. Создание класса Generator.

Создание класса, который предназначен для генерации заявок случайным образом, на основе которого создаётся таблица с заявками и история тренда.

16. Обеспечение взаимодействия REST API и UI.

Рисунок 7. CLI.

На рисунке 7 представлен CLI, встроенный в серверную часть приложения.

3.2 Отображение сущностей на базу данных

Отображение сущностей на базу данных происходит при помощи DataProviderDB и JDBC-драйвера. DataProviderDB производит преобразование объектной модели в реляционную и наоборот.

Метод insert производит вставку передаваемого списка объектов в базу данных.

Метод update обновляет передаваемый объект по указанному id.

Метод delete удаляет объект или список объектов по указанному id.

Метод select возвращает коллекцию требуемых объектов, которые соответствуют передаваемым условиям.

3.3 Реализация взаимодействия с базой данных

Для взаимодействия с базой данных используется класс DataProviderDB, который реализует интерфейс IDataProvider, в котором описаны все основные методы по взаимодействию с базой данных.

Основные методы по добавлению, получению, обновлению и удалению взаимодействуют с абстрактным родительским классом для всех существующих классов - Generic.

Интерфейс содержит следующие методы:

- public Result insert(List<Generic> object) — метод предназначен для добавления списка объектов в базу данных;
- public Result update(Generic object) — метод, обновляющий кортеж в базе данных по id передаваемого объекта;
- public Result delete(Generic object) — метод, удаляющий кортеж в базе данных по id передаваемого объекта;
- public Optional<List<Generic>> select(ClassType type) — метод, возвращающий список объектов из базы данных из таблица, соответствующей передаваемому ClassType;

- public Generic getObjectByID(long id, ClassType type) — метод, предназначенный для получения объекта по передаваемому id в таблице, соответствующей ClassType;
- public Generic getRequestByID(long id) — метод, предназначенный для получения объекта по передаваемому id в таблице REQUEST;
- public Generic getScoreByID(long id) — метод, предназначенный для получения объекта по передаваемому id в таблице SCORE;
- public Generic getTarifByID(long id) — метод, предназначенный для получения объекта по передаваемому id в таблице TARIF;
- public Generic getTradeByID(long id) — метод, предназначенный для получения объекта по передаваемому id в таблице TRADE;
- public Generic getTrendByID(long id) — метод, предназначенный для получения объекта по передаваемому id в таблице TREND;
- public Generic getUserByID(long id) — метод, предназначенный для получения объекта по передаваемому id в таблице USER;

Также в DataProviderDB реализованы следующие методы дополнительно:

- public Result delete(List<Generic> list) — удаляет из базы данных все кортежи, которые соответствуют id передаваемых в списке объектов в соответствующих объектам таблице;
- public Result delete(long id, ClassType type) — удаляет кортеж с соответствующим номером id в таблице, соответствующей передаваемому ClassType;

- public Optional<List<Generic>> select(ClassType type, String col, String param) — Возвращает список объектов из таблицы, соответствующей ClassType, в соответствии с передаваемым условием сравнения;
- public Optional<List<Generic>> select(ClassType type, String col, String param, String sr) — Возвращает список объектов из таблицы, соответствующей ClassType, в соответствии с передаваемым условием сравнения и операндов сравнения;
- public Generic getLastTrend() — Возвращает объект, соответствующий последнему кортежу в таблице Trend;
- public void dropAndCreate(ClassType type) — Очищает полностью и сбрасывает auto_increment в таблице, соответствующей классу type;
- public void dropAndCreateTrend() — Очищает полностью и сбрасывает auto_increment в таблице Trend и создаёт там первую запись.

3.4 Реализация интерфейса пользователя

Для реализации пользовательского интерфейса использовались библиотеки JFrame, JFreeChart и JTable.

Первичное построение формы пользовательской происходит в методе createAndShowGUI в главном классе UI, который представлен в листинге 1.

Листинг 1. JFreeCandlestickChartDemo

```
public class JfreeCandlestickChartDemo extends JPanel {
    private static void createAndShowGUI() throws
MalformedURLException, IOException {
        JFrame.setDefaultLookAndFeelDecorated(true);
        //Create and set up the window.
        JFrame frame = new JFrame("JFreeChartDemo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Create and set up the chart.
```

```
            JfreeCandlestickChart jfreeCandlestickChart =
new JfreeCandlestickChart("Financial Active");
            new FxMarketPxFeeder(jfreeCandlestickChart, "",
2).run();
            frame.setContentPane(jfreeCandlestickChart);
            //Disable the resizing feature
            frame.setResizable(false);
            //Display the window.
            frame.pack();
            frame.setVisible(true);
        }
        public static void main(String[] args) {
            //creating and showing this application's GUI.
            SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    try {
                        createAndShowGUI();
                    } catch (IOException ex) {
Logger.getLogger(JfreeCandlestickChartDemo.class.getName()).lo
g(Level.SEVERE, null, ex);
                    }
                }
            });
        }
    }
```

Класс JFreeCandlestickChartDemo наследуется от класса JPanel и создаёт форму для пользовательского интерфейса.

В классе FxMarketPxFeeder происходит вызов метода в отдельном потоке, который прорисовывает в ходе работы приложения график цен и стакан заявок. Метод представлен в Листинге 2.

Листинг 2. Метод run класса FxMarketPxFeeder.

```
    public void run() {
        if(stockTradesFile==null || stockTradesFile=="")
```

```
executorService.execute(() -> {
                    try {
                        readrest();
                    } catch (InterruptedException ex) {

Logger.getLogger(FxMarketPxFeeder.class.getName()).log(Level.S
EVERE, null, ex);
                    }
                });
                else executorService.execute(() -> read());
     }


      private void readrest() throws InterruptedException{
            final Gson gson=new Gson();
            long readTrend=0;
            HttpRequest<String> httpRequest =
HttpRequestBuilder.createPost("http://localhost:8080/getTrends
",
String.class).responseDeserializer(ResponseDeserializer.ignora
bleDeserializer()).addContentType(ContentType.APPLICATION_JSON
).build();
            Optional<List<TrendStory>> gen =
gson.fromJson(httpRequest.execute().get(), new
TypeToken<Optional<List<TrendStory>>>(){}.getType());
            gen.get().stream().forEach(obj->{
jfreeCandlestickChart.addCandel(TimeUtils.convertToMillisTime(
obj.getTime()), obj.getOpen(), obj.getHigh(), obj.getLow(),
obj.getClose(), obj.getVolume());
            });
            HttpRequest<String> httpRequestTable;
            Optional<List<Generic>> lg;
            while(true){
     httpRequestTable =
```

```
HttpRequestBuilder.createPost("http://localhost:8080/getReques
ts",String.class).responseDeserializer(ResponseDeserializer.ig
norableDeserializer()).addContentType(ContentType.APPLICATION_
JSON).build();
     lg = gson.fromJson(httpRequestTable.execute().get(), new
TypeToken<Optional<List<Request>>>(){}.getType());
                jfreeCandlestickChart.onTradeDB(lg);
                Thread.sleep(simulationTime*3);
                readTrend+=simulationTime;
                if(readTrend>200){
      id_last = gen.get().get(gen.get().size()-1).getId();
                    readTrend=0;
     httpRequest =
HttpRequestBuilder.createPost("http://localhost:8080/getLastTr
end",String.class).responseDeserializer(ResponseDeserializer.i
gnorableDeserializer()).addContentType(ContentType.APPLICATION
_JSON).build();
        gen = gson.fromJson(httpRequest.execute().get(), new
TypeToken<Optional<List<TrendStory>>>(){}.getType
                    gen.get().stream().forEach((TrendStory
obj)->{
                        if(!obj.equalsId(id_last))
jfreeCandlestickChart.addCandel(TimeUtils.convertToMillisTime(
obj.getTime()), obj.getOpen(), obj.getHigh(), obj.getLow(),
obj.getClose(), obj.getVolume());
                    });
                }
            }
        }
```

Добавление новых элементов происходит в классе jfreeCandlestickChart, где используются 2 метода, один из которых обновляет данные в стакане заявок, а второй в свою очередь добавляет новый элемент в график цен. Методы данные представлены в Листинге 3.

Листинг 3. jfreeCandlestickChart

```
      public void addCandel(long time, double o, double h,
double l, double c, long v) {
                ttm.fireTableDataChanged();
          try {
          FixedMillisecond t = new FixedMillisecond(
READABLE_TIME_FORMAT.parse(TimeUtils.convertToReadableTime(tim
e)));
               ohlcSeries.add(t, o, h, l, c);
               volumeSeries.add(t, v);
          } catch (ParseException e) {
               e.printStackTrace();
          }
     }

      public void onTradeDB(Optional<List<Generic>> lg){
                boolean l=false;
                if(ttm.getRowCount()==0)l=true;
                lg.get().stream()
                        .forEach(obj->{
         Object []ar = {((Request)obj).getPrice(),
((Request)obj).getVolume(),""};
                            ttm.addDataDB(ar);
                        });
                if(l)ttm.fireTableDataChanged();
                ttm.fireTableRowsUpdated(0,ttm.getRowCount());
        }
```

В свою очередь добавление данных в стакан заявок происходит в классе TradeTableModel в методе addDataDB, который представлен в Листинге 4.

Листинг 4. TradeTableModel

```
      public void addDataDB(Object []row){
        Object []rowTable=new Object[getColumnCount()];
```

```
        rowTable=row;
        boolean l=true;
        Object []obj=new Object[getColumnCount()];
        long j;
        for(int i=0;i<dataArrayList.size();i++){
            obj=dataArrayList.get(i);

            if((double)obj[0]<(double)row[0]){
                dataArrayList.add(i,rowTable);
                l=false;
                break;
            }else if((double)obj[0]==(double)row[0]){
                obj[1]=(long)row[1];
                dataArrayList.set(i, obj);
                l=false;
                break;
            }
        }
              if(l) dataArrayList.add(rowTable);
    }
```

В ходе всех данных действий был создан пользовательский интерфейс.

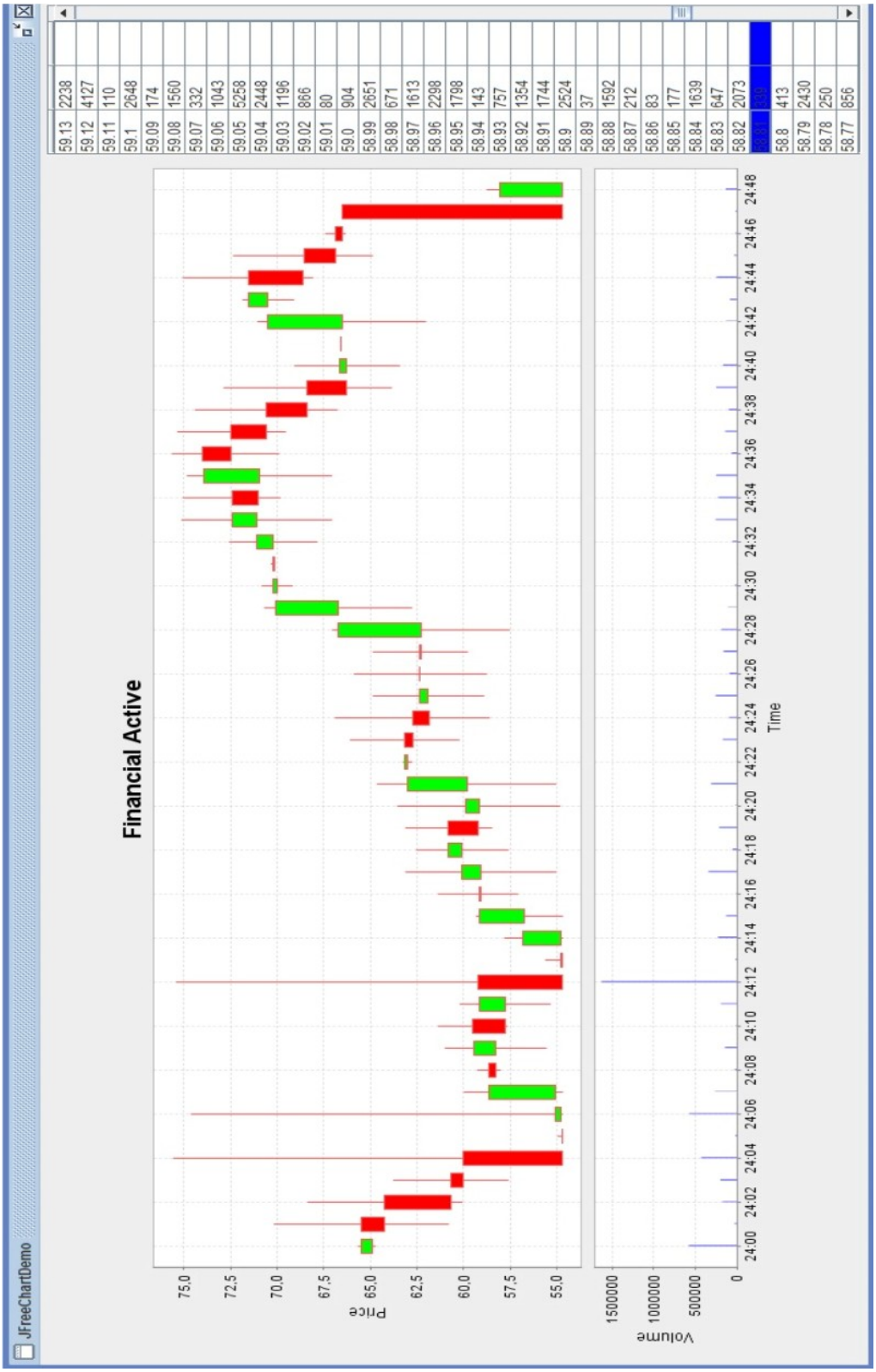

Рисунок 8. Пользовательский торговый интерфейс.

На рисунке 8 представлен пользовательский интерфейс, который позволяет просматривать получаемую числовую информацию в графическом виде.

ГЛАВА 4 ТЕСТИРОВАНИЕ

4.1 Тестирование CRUD-операций

Тестирование в Java-приложении проводится с помощью JUnitTest. Библиотека подключается средствами автоматической сборки maven и используется только во время тестирования.

Были написаны следующие файлы тестов:

DataProviderCsv1InsertTest — тестирование метода Insert для каждого из типов объектов в источнике данных csv.

DataProviderCsv2SelectTest — тестирование методов Select и getById для каждого из типов объектов в источнике данных csv.

DataProviderCsv3UpdateTest — тестирование метода Update для каждого из типов объектов в источнике данных csv.

DataProviderCsv4DeleteTest — тестирование метода Delete для каждого из типов объектов в источнике данных csv.

DataProviderXml1InsertTest — тестирование метода Insert для каждого из типов объектов в источнике данных xml.

DataProviderXml2SelectTest — тестирование методов Select и getById для каждого из типов объектов в источнике данных xml.

DataProviderXml3UpdateTest — тестирование метода Update для каждого из типов объектов в источнике данных xml.

DataProviderXml4DeleteTest — тестирование метода Delete для каждого из типов объектов в источнике данных xml.

DataProviderDB1InsertTest — тестирование метода Insert для каждого из типов объектов в источнике данных DataBase.

DataProviderDB2SelectTest — тестирование методов Select и getById для каждого из типов объектов в источнике данных DataBase.

DataProviderDB3UpdateTest — тестирование метода Update для каждого из типов объектов в источнике данных DataBse.

DataProviderDB4DeleteTest — тестирование метода Delete для каждого из типов объектов в источнике данных DataBase.

В тестовых классах DataProvider*1InsertTest реализованы следующие методы:

- testInsert1User — метод, предназначенный для генерации и записи объектов класса Users;
- testInsert2Score — метод, предназначенный для генерации и записи объектов класса Score;
- testInsert3Tarif — метод, предназначенный для генерации и записи объектов класса Tariffing;
- testInsert4Trade — метод, предназначенный для генерации и записи объектов класса TradeStory;
- testInsert5Trend — метод, предназначенный для генерации и записи объектов класса TrendStory;
- testInsert6Request — метод, предназначенный для генерации и записи объектов класса Request.

В данных тестовых классах используется перед объявлением класса аннотация @FixMethodOrder(MethodSorters.NAME_ASCENDING), которая предназначена для вызова методов класса в алфавитном порядке. В данных классах это важно для соблюдения условия целостности данных.

В тестовых классах DataProvider*2SelectTest реализованы следующие методы:

- testSelectUser — метод, предназначенный для получения объектов класса Users;
- testSelectScore — метод, предназначенный для получения объектов класса Score;
- testSelectTarif — метод, предназначенный для получения объектов класса Tariffing;

- testSelectTrade — метод, предназначенный для получения объектов класса TradeStory;
- testSelectTrend — метод, предназначенный для получения объектов класса TrendStory;
- testSelectRequest — метод, предназначенный для получения объектов класса Request;
- testGetUserById — метод, предназначенный для получения объекта класса User по заданному id;
- testGetScoreById — метод, предназначенный для получения объекта класса Score по заданному id;
- testGetRequestById — метод, предназначенный для получения объекта класса Request по заданному id;
- testGetTarifById — метод, предназначенный для получения объекта класса Tariffing по заданному id;
- testGetTradeById — метод, предназначенный для получения объекта класса TradeStory по заданному id;
- testGetTrendById — метод, предназначенный для получения объекта класса TrendStory по заданному id.

В тестовых классах DataProvider*3UpdateTest реализованы следующие методы:

- testUpdateUser — метод, предназначенный для генерации и изменения объектов класса Users;
- testUpdateScore — метод, предназначенный для генерации и изменения объектов класса Score;
- testUpdateTarif — метод, предназначенный для генерации и изменения объектов класса Tariffing;
- testUpdateTrade — метод, предназначенный для генерации и изменения объектов класса TradeStory;

- testUpdateTrend — метод, предназначенный для генерации и изменения объектов класса TrendStory;
- testUpdateRequest — метод, предназначенный для генерации и изменения объектов класса Request.

В тестовых классах DataProvider*4DeleteTest реализованы следующие методы:

- testDeleteUser — метод, предназначенный для генерации и удаления объектов класса Users. Реализовано каскадное удаление зависимых объектов;..
- testDeleteScore — метод, предназначенный для генерации и удаления объектов класса Score;
- testDeleteTarif — метод, предназначенный для генерации и удаления объектов класса Tariffing;
- testDeleteTrade — метод, предназначенный для генерации и удаления объектов класса TradeStory;
- testDeleteTrend — метод, предназначенный для генерации и удаления объектов класса TrendStory;
- testDeleteRequest — метод, предназначенный для генерации и удаления объектов класса Request.

В тестовых классах DataProvider*2SelectTest проверка положительного завершения тестового метода происходит при помощи функции assertNotNull(result.get()), где result — возвращаемое значение метода select.

В тестовых классах DataProvider*1InsertTest, DataProvider*3UpdateTest, DataProvider*4DeleteTest, проверка положительного завершения тестового метода происходит при помощи функции assertEquals(expResult, result), где result — возвращаемое значение метода соответствующего метода, а expResult — результирующая переменная со статусом “OK”.

При успешном выполнении теста будет, например, такое сообщение: Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.831 sec.

При выполнении всех тестов одновременно выводится следующее сообщение:

Results :

Tests run: 80, Failures: 0, Errors: 0, Skipped: 0

Все тесты, соответственно, выполнились положительно.

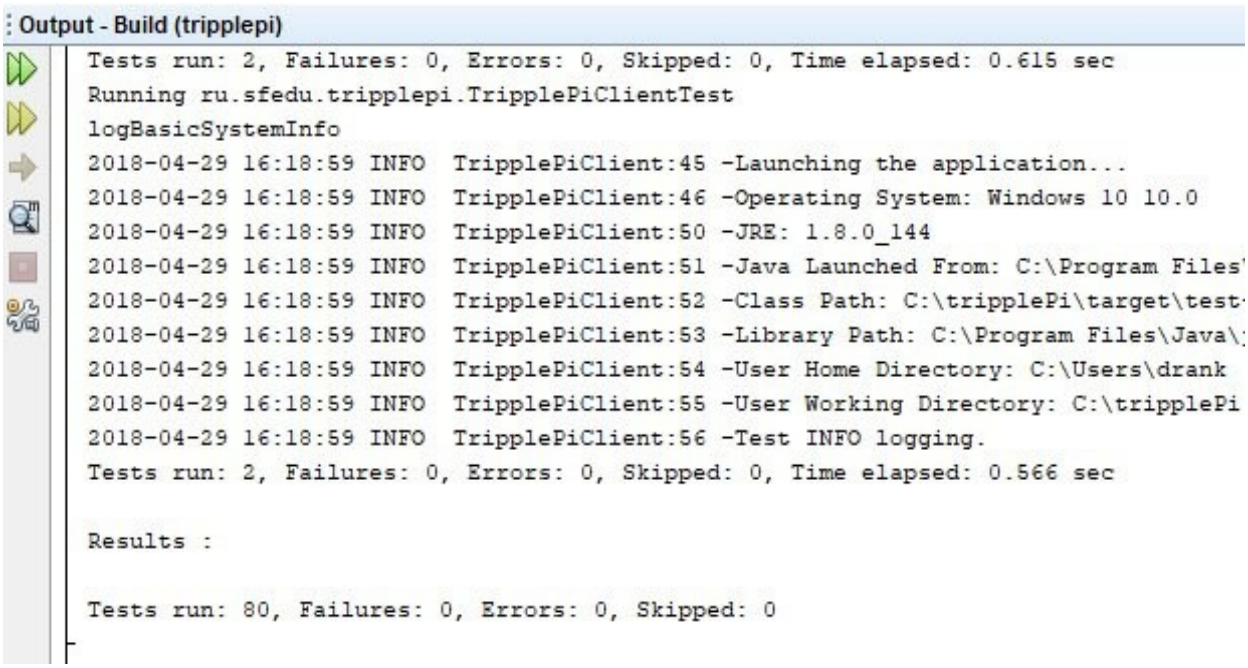

Рисунок 9. Результаты тестирования.

На рисунке 9 представлен результат прохождения всех операций тестирования.

4.2 Тестирование REST API

Для тестирования REST API использовался UI, во время журнализации которого были показаны результаты обращения к REST API по URI. Результаты представлены в таблице 3.

Таблица 3. Результаты тестирования REST API

Пункт прог. испыт.	Наименование проверки	Последовательность действий	Определение успешности проверки	Примеч.
1	Доступ к существующим функциями	Отправка запроса к существующему ресурсу	Получение ответа со статусом 200 (OK)	Ошибок нет
2	Доступ к не существующим функциями	Отправка запроса к не существующему ресурсу	Получение ответа со статусом 404 (Content is not present)	NoSuchContentException
3	Доступ к незапущенному сервису	Отправка запроса, когда сервис выключен	Получение ответа со статусом 503 (Content is not present)	NoSuchContentException

4.3 Разработка программы испытаний тестирования конечного продукта

Тестирование данного продукта заключается в тестировании модуля, который отвечает за взаимодействие с различными DataProviders, такие как csv, xml и DB. Также проверяется работоспособность клиентской части приложения и REST API.

Тестирование DataProviders заключается в проверке их взаимодействия с различными источниками данных при помощи операций добавления, чтения, изменения и удаления записей. Методика тестирования приведена в таблице 4.

Таблица 4. Тестирование DataProviders.

Пункт прогр. испыт.	Наименование показателя	Пункт требований ТЗ	Методика проверки
1	Создание объекта		Проверка наличия объекта
2	Чтение объектов		Проверка наличия объектов
3	Получение объекта по ID		Проверка наличия текущего объекта
4	Редактирование объекта		Сравнение исходных данных и изменённых. Проверка правильности сохранения
5	Удаление объекта		Проверка наличия удалённого объекта.

Тестирование REST API заключается в проверке его взаимодействия с клиентской частью, отправке HTTP-запросов и анализе ответов на них. Методика тестирования приведена в таблице 5.

Таблица 5. Тестирование REST API.

Пункт прогр. испыт.	Наименование показателя	Пункт требований ТЗ	Методика проверки
1	Проверка GET-запроса		Проверка получаемого Json-объекта
2	Проверка POST-запроса		Проверка получаемого Json-объекта

Тестирование UI заключается в проверке динамического создания графика и стакана цен, а также работе контекстного меню. Методика тестирования приведена в таблице 6.

Таблица 6. Тестирование UI.

Пункт прогр. испыт.	Наименование показателя	Пункт требований ТЗ	Методика проверки
1	Рисование графика цен		Проверка графического представления графика
2	Рисование стакана заявок		Проверка графического представления стакана заявок
3	Проверка контекстного меню		Проверка работоспособности каждого пункта контекстного меню

ЗАКЛЮЧЕНИЕ

Бизнес-логика к приложению реализована. Все основные методы успешно работают. Тесты успешно выполняются.

Выполненная работа позволила многократно усовершенствовать навыки программирования на современных объектно-ориентированных языках, таких как Java. Также получены знания в работе с различными библиотеками и источниками данных.

В ходе работы удалось выявить некоторые минусы в работе приложений подобного типа и их “слабые места”.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Трейдер. URL: https://ru.wikipedia.org/wiki/Трейдер#cite_note-1 (Дата обращения: 31.05.2018г)

2. Марк Фридфертиг, Джордж Уэст — «Электронная внутридневная торговля»

3. Скальпинг. URL: https://ru.wikipedia.org/wiki/Скальпинг (дата обращения: 18.03.2018г).

4. Торговые приводы для скальпинга. URL: http://dctrading.ru/scalping-privody-m25/ (Дата обращения: 09.03.2018г)

5. Как пользоваться стаканом цен на Форекс. URL: http://vashbiznesplan.ru/investicii/forex/kak-polzovatsya-stakanom-tsen-na-foreks.html (Дата обращения: 10.03.2018г)

6. Торговый терминал EasyScalp. URL: http://easyscalp.ru/ (Дата обращения: 11.03.2018г)

7. Скачать Easy Scalp. URL: http://easyscalp.ru/download/ (Дата обращения: 17.03.2018г)

8. Торговый привод QScalp. URL: https://www.qscalp.ru/ (Дата обращения: 11.03.2018г)

9. Торговый прйвод QScalp 3.6. Руководство пользователя.

10. TradingView. URL: https://ru.wikipedia.org/wiki/TradingView (Дата обращения: 11.03.2018г)

11. UML. URL: https://ru.wikipedia.org/wiki/UML (дата обращения: 24.12.2017).

12. Spark. URL: http://sparkjava.com/ (дата обращения: 29.05.2018).

13. OpenCsv. URL: http://opencsv.sourceforge.net (дата обращения: 12.11.2017).

14. Simple Xml Serialization. URL: http://simple.sourceforge.net (дата обращения: 15.11.2017).

15. PostgreSQL JDBC Driver. URL: https://jdbc.postgresql.org (дата обращения: 30.11.2017).

16. Тестирование в Java. JUnit. URL: https://habrahabr.ru/post/120101 (дата обращения: 15.12.2017).

17. Java EE Tutorial. URL: https://javaee.github.io/tutorial/ (дата обращения: 29.05.2018).

18. JDK 10 Documentation. URL: https://docs.oracle.com/javase/10/ (дата обращения: 29.05.2018).

19.

Приложение А ТЕХНИЧЕСКОЕ ЗАДАНИЕ

А1. Введение

Необходимо разработать приложение на Java платформе, реализующее бизнес-логику для работы с данными торгового привода, являющегося составной частью полноценного торгового привода с графическим представлением активов.

Приложение будет представлять собой торговый привод с графическим представлением активов в виде графика японских свечей и стакана заявок.

А2. Основание для разработки

Задание на выпускную квалификационную работу

А3. Назначение разработки

Приложение предназначено для использования в учебных и коммерческих целях в качестве торгового привода. Привод должен предоставлять информацию относительно следующих видов объектов:

пользователь — текущий пользователь;

история тренда — история текущего тренда в минутном таймфрейме;

история торговли — история торговли пользователей;

заявки — текущие открытые заявки;

счет — счета пользователей пользователя;

тариф — тарифы пользователей.

Также пользовательская часть приложения должна предоставлять информацию в графическом виде для следующих объектов:

история тренда — история тренда;

заявки — текущие открытые заявки.

Пользователь будет иметь возможности по просмотру движения тренда и ознакомлением с торговой платформой. Также должна быть возможность просмотра текущих заявок.

А4. Требования к модулю для работы с базой данных

Модуль для работы с базой данных должен обеспечивать выполнение операций создания, чтения, редактирования и удаления любого объекта. Операции должны корректно выполняться.

Вся информация хранится в реляционной базе данных. Также имеется возможность работы приложения с такими источниками данных, как CSV и XML.

А5. Требования к платформе

Программа должна запускаться на всех платформах операционных систем, в которых предполагается поддержка JVM.

Приложение Б ИНСТРУКЦИЯ СИСТЕМНОГО АДМИНИСТРАТОРА

Приложение “Обучающий торговый привод” разделён на 2 части, которые взаимодействуют друг с другом по HTTP-протоколу. Все запросы проходят в формате JSON.

Пользовательская часть приложения предполагает работу в 2х режимах:

В локальном режиме, где данные для графика и стакана будут считываться из локального csv-файла, находящегося в структуре проекта.

В глобальном режиме, где данные для графика и стакана будут запрашиваться из серверной части приложения по HTTP-протоколу.

В случае необходимости провести тестирование системный администратор может запустить CLI в серверной части приложение и просмотреть, удалить, обновить, добавить необходимые данные в базу данных.

Для тестирования клиентской части приложения администратор должен запустить его через подгрузку csv, где записано множество объектов заявок, на основе которых будет заполняться и изменяться стакан заявок и график цен.

Если необходимо будет запустить приложение в штатное режиме, то для этого придётся запустить серверную часть в режиме REST API и клиентскую часть в режиме UI, который предполагает работоспособность клиентского интерфейса, взаимодействуя с серверной частью.

Приложение В ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

Приложение должно функционировать на рабочем месте которое полностью соответствует требованиям пожарной безопасности.

Запуская приложение в тестовом или штатном режиме пользователь будет иметь возможность просматривать движение активов и изменение цен и объёмов.

Также для работы с графиком доступно контекстное меню, которое позволит пользователю изменить настройки(изменить палитру, шрифт, надписи, видимость подписей осей), копировать, сохранять и печатать график, а также работать с масштабом(приближать, отдалять, автомасштабировать).

Рисунок 9. Контекстное меню

На рисунке 9 виден пример контекстного меню, который позволяет работать пользователю с графиком цен.