Министерство просвещения РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Уральский государственный педагогический университет»
Институт математики, физики, информатики и технологий
Кафедра информатики, информационных технологий
и методики обучения информатике

ИНФОРМАЦИОННАЯ СИСТЕМА ДЛЯ ВЕДЕНИЯ СДЕЛОК НА БИРЖЕ

Выпускная квалификационная работа
бакалавра по направлению подготовки
09.03.03 – Прикладная информатика в сервисе

Исполнитель: студент группы ПИ-1601z
Института математики, физики,
информатики и технологий
Кожевников Н.С.

Работа допущена к защите
«____»__________ 2021 г.

Зав. кафедрой __________

Руководитель: доктор педагогических наук,
доцент кафедры ИИТ и МОИ
Лапенок М.В.

Екатеринбург – 2021

РЕФЕРАТ

Кожевников Н.С. ИНФОРМАЦИОННАЯ СИСТЕМА ДЛЯ ВЕДЕНИЯ СДЕЛОК НА БИРЖЕ, выпускная квалификационная работа: 48 стр., рис. 14, библ. 40 назв., приложений 3.

Ключевые слова: БИРЖА, ИНФОРМАЦИОННАЯ СИСТЕМА, КЛИЕНТ-СЕРВЕР, ВЕБ-ПРИЛОЖЕНИЕ.

Предмет разработки – информационная система для сохранения и анализа данных о сделках на финансовой бирже.

Цель работы – разработать веб-приложение на базе клиент-серверной архитектуры для автоматизации сбора данных о биржевых сделках и вывода статистики по уже сохранённым записям.

Данный проект был реализован в качестве двух взаимосвязанных приложений: клиента и сервера. Сервер представляет из себя программное обеспечение, которое принимает запросы от клиента, производит основные вычисления, взаимодействует с базой данных и обращается к стороннему сервису через API для сбора данных. Клиент реализован как интерфейс веб-сайта, через который пользователь взаимодействует с информационной системой. Для сервера были использованы СУБД PostgreSQL, язык программирования Scala и фреймворк Play Framework. Клиент был реализован с помощью языка программирования TypeScript и фреймворка Vue.js.

Итоговая информационная система позволяет пользователю просматривать список уже имеющихся биржевых сделок и вносить новые записи, в том числе с автоматизацией ручного труда по заполнению однотипных параметров сделки. Также имеется возможность просмотреть статистику результатов сделок в разрезе их успешности и прибыльности для пользователя.

Реализованная информационная система полностью соответствует техническому заданию на разработку.

ОГЛАВЛЕНИЕ

Введение 4

Глава I. Теоретические основы построения информационных систем в контексте биржевой торговли 5

1.1. Принципы и технологии современной биржевой торговли, API брокера .. 5

1.2. Анализ и выбор программных средств разработки 9

1.3. Формализованное описание технического задания 24

Глава II. Разработка клиента и сервера для информационной системы ведения сделок на бирже 31

2.1. Описание программной реализации клиентского приложения 31

2.2. Описание программной реализации серверного приложения 34

2.3. Описание принципов работы с разработанной информационной системой 37

Заключение 42

Список информационных источников 44

Приложения 49

Приложение 1. 49

Приложение 2. 51

Приложение 3. 52

Введение

Сфера биржевой торговли, ранее считавшаяся уделом исключительно профессионалов, переживает этап популяризации среди более широких слоёв населения. Многие крупные банки, такие как «Сбер», «Тинькофф», разработали и уже запустили услуги частного инвестирования для своих клиентов. Даже вне этих услуг можно отметить повышение интереса людей к сфере биржевой торговли, что выражается в повышении количества поисковых запросов на эту тему и множеству обращений новичков на тематических форумах в сети Интернет.

При этом биржевая торговля по своей природе является комплексным, крайне сложным родом деятельности. Одним из ключевых приёмов для того, чтобы преуспеть в долгосрочной перспективе, является запись биржевых сделок. Это позволяет проводить регулярное подведение итогов и корректировать торговую стратегию на основе полученной статистики.

На данный момент крайне востребовано программное обеспечение, которое одновременно решает задачи хранения и анализа биржевых данных, а также автоматизацию их заполнения, поэтому данный проект является **актуальным**.

Предмет разработки: информационная система для ведения сделок на бирже.

Цель разработки: спроектировать и разработать клиент-серверное веб-приложение с функционалом хранения и вывода статистики по записям (сделкам), а также автоматизацией в ходе создания новых записей.

Задачи:

1. Изучить процесс биржевой торговли и принципы ведения сделок.
2. Произвести анализ и выбор программных инструментов и средств.
3. Спроектировать информационную систему и разработать её на базе клиент-серверного веб-приложения в соответствии с техническим заданием.
4. Описать разработанное приложение и привести примеры его работы.

Глава I. Теоретические основы построения информационных систем в контексте биржевой торговли

1.1. Принципы и технологии современной биржевой торговли, API брокера

Согласно сайту Википедия, биржа — это «организатор торгов товарами, валютой, ценными бумагами, производными и другими рыночными инструментами» [23].

До эпохи компьютеризации биржи осуществляли свою деятельность в отдельных зданиях, при этом все участники торгов должны были договариваться в устной форме. Первые биржи появились ещё в XVI веке, в Бельгии и Франции. Там проходили торги с использованием векселей и государственных займов.

Современные биржи имеют куда больший список доступных рыночных инструментов и, как правило, уже не имеют торговых залов - они базируются на специализированных программно-аппаратных комплексах и организуют торги в полностью электронном виде. С помощью сложных информационных систем ведётся исполнение сделок, их учёт и реализуются все необходимые расчёты.

В торгах на современных биржах не может участвовать случайный человек – законодательство ограничивает биржевую торговлю для физических лиц. Так, на веб-сайте Московской биржи прямо указано, что «физические лица могут участвовать в торгах на рынках Группы Московская Биржа только в качестве клиентов участников торгов» [34]. При этом для того, чтобы стать участником торгов, недостаточно быть юридическим лицом - необходимо также получить лицензию профессионального участника рынка, приобрести права на торговлю и оплачивать ежемесячные комиссии биржи. Всё это возможно, но крайне затруднительно для рядового инвестора.

Поэтому наиболее целесообразным является торговля через брокера – профессионального биржевого посредника при заключении сделок между покупателями и продавцами на рынке. Как торговый представитель, брокер:

- хранит, использует и учитывает денежные средства клиентов, предназначенные для приобретения различных рыночных инструментов;
- удостоверяется в правомочии клиентов – юридических лиц и в дееспособности клиентов – физических лиц;
- оказывает консультационные услуги по вопросам торговли на бирже;
- удерживает комиссионные с каждой сделки, которые являются основным источником его дохода [40, с. 103].

Современные брокеры, как и биржи, тоже предлагают свои услуги в основном в электронной форме - через веб-платформы, на которых происходит регистрация пользователей, управление их сделками и хранение исторических данных. Таким образом, рядовой инвестор может начать торговлю на бирже, не обладая специализированным оборудованием, профессиональным программным обеспечением и правовыми лицензиями. Для начала достаточно только устройства с выходом в сеть Интернет, регистрации на платформе брокера и начальный капитал для открытия сделок.

Однако в долгосрочной перспективе нужно приумножать исходные денежные средства, так как в этом и состоит смысл биржевой торговли для любого инвестора. С учётом хаотичной природы рынка задача долгосрочного роста капитала считается крайне сложной, особенно для непрофессионального участника торгов. Поэтому одной из важнейших практик в биржевой торговле является систематическое ведение записей о своих сделках. В профессиональной литературе прошлых лет чаще всего подразумевается использование простого блокнота, в котором следует расчертить таблицу для сделок и затем вносить все необходимые данные вручную [39, с. 769].

На данный момент такой вариант уже не актуален, учитывая развитие веб-технологий и устройств для выхода в сеть Интернет. Ведение списка сделок в электронном виде - более удобный способ, учитывая возможности, которые предоставляют современные веб-приложения. Это не только более надёжное хранение информации в базах данных с применением облачных технологий, но и различные дополнительные услуги развитых информационных систем:

удобный вывод сохранённых записей в виде интерактивных таблиц, анализ и сбор статистики, а также частичная или полная автоматизация заполнения данных.

Такая автоматизация становится возможной благодаря взаимодействию веб-приложения для ведения сделок с API платформы брокера. Согласно сайту sendpulse.com, API (программный интерфейс приложения) (англ. Application Programming Interface) – это «интерфейс прикладного программирования, который представляет собой набор готовых классов, процедур, функций, структур и констант, которые предоставляются сервисом для использования во внешних программных продуктах» [37].

Многие современные брокеры разрабатывают и предоставляют API всем желающим, чтобы у разработчиков программного обеспечения была возможность обращаться к функционалу платформы брокера напрямую из своего приложения, без использования графического пользовательского интерфейса. Однако, чтобы начать пользоваться API, необходимо однократно произвести несколько подготовительных действий. В первую очередь зарегистрировать своё приложение в специальном разделе веб-сайта брокера, который предназначен для разработчиков ПО. Для этого достаточно указать его наименование и адрес в сети Интернет. После регистрации разработчик получает уникальный числовой идентификатор своего приложения на платформе брокера. Этот идентификатор используется для связи приложения и веб-сайта. Сам по себе он позволяет получить лишь информацию общего плана, например, поток данных по текущим котировкам валют.

Для того, чтобы запрашивать данные о конкретной клиентской учётной записи, нужно получить отдельный авторизационный токен. Это уникальная последовательность символов, связывающая приложение и определённого пользователя веб-сайта. При этом пользователь может указать права доступа для такого приложения относительно его учётной записи — это может быть только чтение, управление сделками или даже доступ к денежному счёту [1].

После регистрации и авторизации разработчик получает возможность обращаться к учётной записи пользователя из своего приложения, используя HTTP-запросы. Обмен информацией осуществляется в текстовом формате обмена данными «JSON», основанном на языке программирования JavaScript [24].

Таким образом можно получать сведения об истории сделок пользователя. Каждая сделка имеет уникальный номер, что позволяет найти её в общем списке и получить необходимые характеристики. Это множество характеристик формируется исходя из процесса торговли. Любая сделка на финансовой бирже — это процесс открытия контракта в какой-либо валюте на покупку или продажу при определённом уровне стоимости этой валюты, а затем закрытие этого контракта. Соответственно, из веб-платформы брокера с помощью API выгружаются следующие характеристики сделки:

1. Дата и время открытия сделки.
2. Актив (валютная пара), в рамках которого открывался контракт. Например: EUR/USD.
3. Экспирация — это время от момента открытия до момента закрытия сделки, указывается в минутах.
4. Прогноз – обозначение, была ли произведена покупка или продажа контракта. Покупка обозначается как «CALL», продажа как «PUT».
5. Вложение – сумма контракта, это те денежные средства, которыми рискует пользователь при открытии контракта. Указывается в долларах США.
6. Процент потенциального дохода в случае успешности контракта — это процент от вложения пользователя, который он получит, если его прогноз окажется верным по окончанию сделки. Процент дохода устанавливается брокером.
7. Итоговая прибыль - сумма, которую пользователь получает или теряет по итогу контракта, в зависимости от успешности своего прогноза.

1.2. Анализ и выбор программных средств разработки

Для проектирования и реализации проекта необходимо определить, с помощью каких программных инструментов и методов будет разработана информационная система. Начать стоит с определения самого понятия предмета разработки. Обратимся к стандарту по информационным технологиям ГОСТ 34.321-96. «Информационная система — это система, которая организует процессы сбора, хранения и обработки информации о проблемной области. Она может быть размещена на одной или нескольких компьютерных системах» [26].

Любая информационная система состоит из трёх ключевых компонентов:

- программное обеспечение;
- аппаратное обеспечение;
- персонал, обслуживающий ИС.

В данном проекте аппаратное обеспечение представляет из себя серверную ЭВМ, а программное - два приложения, клиент и сервер, которые в совокупности предоставляют пользователю функционал ИС. Наконец, персонал в данном случае — это разработчик, занимающийся разработкой и последующим сопровождением программного кода проекта.

Программное обеспечение и методы его разработки являются наиболее сложным аспектом, поэтому будет целесообразно рассмотреть именно его. Согласно цели разработки, веб-приложение должно быть реализовано на базе клиент-серверной архитектуры. Это такая вычислительная архитектура, в которой задания распределены между сервером, представляющим из себя одну или несколько выделенных ЭВМ с повышенной производительностью, и клиентом, посылающим на сервер задания для их обработки [20].

На практике, клиент и сервер — это совокупность программного и аппаратного обеспечения. Обычно они располагаются на разных ЭВМ и обмениваются данными через вычислительную сеть - локальную или сеть Интернет. Сутью данной архитектуры является предоставление одним сервером различных сервисных услуг или ресурсов множеству клиентов. Она позволяет

выполнять трудоёмкие операции и задания, требующие повышенной производительности, даже на тех ЭВМ, которые не обладают достаточными характеристиками. Если у этих ЭВМ есть программное и аппаратное обеспечение, позволяющее им быть клиентами, то они могут обращаться на сервер, который произведёт все вычисления и отправит им только конечный результат.

С точки зрения обычного пользователя, сервер — это любой веб-сайт в сети Интернет, обслуживаемый на отдельной ЭВМ, а клиент — это любая другая ЭВМ с установленным на ней браузером и возможностью выхода в сеть. Однако в контексте современной разработки клиент и сервер — это два слабосвязанных компонента одного веб-приложения. Обычно они представлены двумя отдельными приложениями, часто пишутся на разных языках программирования, используют различные технологии и подходы.

При такой архитектуре в разработке различают фронтенд (англ. front-end) — это клиентская сторона пользовательского интерфейса к программно-аппаратной части веб-сервиса, и бэкенд (англ. back-end) — это непосредственно сервер, программно-аппаратная часть веб-сервиса. Фронтенд задействует мощности ЭВМ конечного пользователя с помощью браузера, а программный код бэкенда всегда исполняется на выделенной ЭВМ. Чаще всего они обмениваются информацией с помощью HTTP-запросов [3].

В категорию программных средств разработки входят: язык программирования, вспомогательные программные средства, такие как библиотеки, а также набор компонентов, которые позволяют набирать исходный код, компилировать и выполнять отладку. Программист может выполнять это всё вручную и по отдельности – например, набирать исходный код в обычном текстовом редакторе, а компилировать и отлаживать проект в командной строке. Однако такой подход сильно снижает скорость работы, так как необходимо выполнять множество повторяющихся действий вручную. К тому же, это неудобно, особенно для начинающих разработчиков – даже о незначительной

синтаксической ошибке в коде можно узнать лишь на этапе компиляции, уже потратив некоторое время на сборку проекта.

Поэтому на данный момент очень распространены среды разработки – системы, уже объединяющие в себе все вышеуказанные компоненты и выполняющие большую часть действий автоматически. Далее будет приведено более строгое определение.

Интегрированная среда разработки, ИСР (англ. Integrated Development Environment – IDE) — это совокупность программных средств, поддерживающая все этапы разработки программного обеспечения от написания исходного текста программы до ее компиляции и отладки, и обеспечивающая простое и быстрое взаимодействие с другими инструментальными средствами [29].

Среда разработки включает в себя:

- текстовый редактор;
- компилятор и/или интерпретатор;
- средства автоматизации сборки;
- отладчик.

Иногда содержит также средства для интеграции с системами управления версиями и разнообразные инструменты для упрощения конструирования графического интерфейса пользователя. Многие современные среды разработки также включают браузер классов, инспектор объектов и диаграмму иерархии классов — для использования при объектно-ориентированной разработке ПО. ИСР обычно предназначены для нескольких языков программирования.

Использование ИСР для разработки программного обеспечения является прямой противоположностью способу, в котором используются несвязанные инструменты, такие как текстовый редактор, компилятор, и т.п.

Интегрированные среды разработки были созданы для того, чтобы максимизировать производительность программиста благодаря тесно связанным компонентам с простыми пользовательскими интерфейсами. Это позволяет разработчику выполнять меньше действий для переключения различных режимов, в отличие от дискретных программ разработки. Однако, так как ИСР

является сложным программным комплексом, то среда разработки сможет качественно ускорить процесс разработки ПО лишь после специального обучения.

ИСР обычно представляет собой единственную программу, в которой проводится вся разработка. Она, как правило, содержит много функций для создания, изменения, компилирования, развертывания и отладки программного обеспечения. Цель интегрированной среды заключается в том, чтобы объединить различные утилиты в одном модуле, который позволит абстрагироваться от выполнения вспомогательных задач, тем самым позволяя программисту сосредоточиться на решении собственно алгоритмической задачи и избежать потерь времени при выполнении типичных технических действий (например, вызове компилятора). Таким образом повышается производительность труда разработчика. Также считается, что тесная интеграция задач разработки может далее повысить производительность за счёт возможности введения дополнительных функций на промежуточных этапах работы. Например, ИСР позволяет проанализировать код и тем самым обеспечить мгновенную обратную связь и уведомить о синтаксических ошибках.

Большинство современных ИСР являются графическими. Но первые ИСР использовались ещё до того, как стали широко применяться операционные системы с графическим интерфейсом — они были основаны на текстовом интерфейсе с использованием функциональных и горячих клавиш для вызова различных функций (например, Turbo Pascal, созданный фирмой Borland) [30].

В данной работе при разработке использовалась ИСР JetBrains IntelliJ IDEA. Определение из Википедии: «IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Scala, разработанная компанией JetBrains.

Первая версия появилась в январе 2001 года и быстро приобрела популярность как первая среда для Java с широким набором интегрированных инструментов для рефакторинга, которые позволяли программистам быстро реорганизовывать исходные тексты программ. Дизайн среды ориентирован на

продуктивность работы программистов, позволяя сконцентрироваться на функциональных задачах, в то время как IntelliJ IDEA выполняет рутинные операции» [5, 6].

Данная среда разработки была выбрана как одна из лучших среди существующих, в первую очередь за её многофункциональность и ускорение процесса разработки.

К главным достоинствам данной ИСР можно отнести следующие:

- Поддержка множества языков программирования, как серверных, так и клиентских. Под «поддержкой» понимается полный учёт всех особенностей языка, включая сложные и неочевидные конструкции, а также помощь в преобразовании таких конструкций. В некоторых сложных языках одну и ту же задачу можно решить несколькими разными синтаксическими элементами. IntelliJ IDEA в большинстве случаев умеет определять наиболее оптимальное решение и подсказывает его.
- Интеграция с Git и другими системами контроля версий, встроенный отладчик, встроенное определение степени покрытия кода тестами, построение диаграмм классов и переходы по иерархиям вверх и вниз, встроенный декомпилятор байткода Java.
- Автоматическое форматирование кода в пределах всего проекта по заданной конфигурации. При разработке ПО особенно важно придерживаться единого стиля оформления программного кода, поэтому возможность задать этот стиль в ИСР и обеспечить его автоматическое соблюдение очень важна для продуктивной разработки.
- Рефакторинг кода. Рефакторинг – процесс переписывания исходного кода в пользу большей читабельности, производительности и дальнейшей расширяемости без изменения функционала [33]. Простой пример – программа может абсолютно верно выполнять поставленную задачу, однако, если она написана плохо, её производительность будет ниже ожидаемой, или расширить её будет крайне сложно либо невозможно. В таком случае проводится рефакторинг кода, который может быть

достаточно сложным процессом, особенно в больших проектах. IntelliJ IDEA может помочь с этим – в ИСР встроена подсистема рефакторинга, которая при запуске показывает программисту, как можно перестроить условия, методы и даже классы для того, чтобы повысить читабельность и производительность кода.

Проект использует и другие программные инструменты, однако они разнятся для клиента и сервера, поэтому их имеет смысл рассмотреть отдельно. Для реализации сервера был выбран язык программирования Scala. Согласно официальной документации, Scala — это современный мультипарадигмальный язык программирования, разработанный для выражения общих концепций программирования в простой, удобной и типобезопасной манере [14].

Язык был создан в 2001—2004 годах в Лаборатории методов программирования EPFL [13]. Он стал результатом исследований, направленных на разработку улучшенной языковой поддержки компонентного программного обеспечения. За основу при разработке языка были взяты две идеи:

1. Язык программирования компонентного ПО должен быть масштабируемым в том смысле, что должна быть возможность с помощью одних и тех же концепций описать как маленькие, так и большие части. Поэтому внимание было сконцентрировано на механизмах абстракции, композиции и декомпозиции вместо введения большого количества примитивов, которые могут быть полезными только на каком-то одном уровне масштабирования.
2. Масштабируемая поддержка компонентов может быть предоставлена языком программирования, унифицирующим и обобщающим объектно-ориентированное и функциональное программирование. Некоторые из основных технических новшеств Scala — это концепции, представляющие собой сплав этих парадигм программирования. В статически типизированных языках, к которым относится Scala, эти парадигмы до сих пор были почти полностью разделены [12].

Scala-программы во многом похожи на Java-программы, и могут свободно взаимодействовать с Java-кодом. Язык включает единообразную объектную модель — в том смысле, что любое значение является объектом, а любая операция — вызовом метода. При этом является также функциональным языком в том смысле, что функции — это полноправные значения.

В Scala включены мощные и единообразные концепции абстракций как для типов, так и для значений. В частности, язык содержит гибкие симметричные конструкции примесей для композиции классов и типажей. Позволяет производить декомпозицию объектов путём сравнения с образцом; образцы и выражения при этом были обобщены для поддержки естественной обработки XML-документов. В целом, эти конструкции позволяют легко выражать самостоятельные компоненты, использующие библиотеки Scala, не пользуясь специальными языковыми конструкциями.

Язык допускает внешние расширения компонентов с использованием представлений (views). Возможности обобщённого программирования реализуются за счёт поддержки обобщённых типов (generics), в том числе высшего порядка (generics of a higher kind). Кроме различных классических структурных типов данных, в язык включена поддержка экзистенциальных типов [12].

Таким образом, Scala является современным языком, созданным для быстрой разработки компонентных приложений – а именно такими обычно являются веб-приложения. Скорость разработки связывают с множеством удобных встроенных инструментов языка, а также с необходимостью писать меньше программного кода за счёт сложного компилятора, который не требует такого подробного описания синтаксических конструкций, как, например, в языках C++ или Java.

Поэтому, в конечном итоге, этот язык был выбран в данной работе именно за свои преимущества: скорость разработки, надёжность, гибкость и расширяемость.

Однако, располагая одним лишь языком программирования, написать достаточно сложное и полнофункциональное приложение за разумные сроки крайне трудно. В программном обеспечении всегда есть определённые части кода, повторяющиеся от проекта к проекту, а если говорить именно о веб-приложениях, то для них уже давно существует несколько устоявшихся стандартов, начиная от архитектуры и заканчивая деталями реализации конкретных задач.

Если в проекте стоит задача получить конечный продукт, а не только приобрести знания, то обычно бессмысленно писать все элементы программы самостоятельно, начиная с самых низких уровней разработки. Как правило, основа для любого проекта уже написана и выложена в открытый бесплатный доступ – расширяя её под свои нужды, можно в короткие сроки написать готовое полнофункциональное приложение. Такой основой называется фреймворк.

Фреймворки — это программные продукты, которые упрощают создание и поддержку технически сложных или нагруженных проектов. Фреймворк, как правило, содержит только базовые программные модули, а все специфичные для проекта компоненты реализуются разработчиком на их основе. Тем самым достигается не только высокая скорость разработки, но и большая производительность и надёжность решений [34].

В данной работе в качестве фреймворка для серверного приложения был выбран Play Framework. Play — каркас разработки с открытым кодом, написанный на Scala и Java, использует паттерн проектирования Model-View-Controller (MVC). Нацелен на повышение производительности, используя договорённости перед конфигурацией, горячую перегрузку кода и отображения ошибок в браузере [8, 9].

Если необходимо написать веб-приложение, используя язык программирования Scala, то Play Framework является де-факто стандартом. Он обладает широким функционалом и позволяет автоматизировать и упростить множество задач, возникающих в ходе разработки подобных приложений. Краткий список возможностей фреймворка:

- асинхронное выполнение;
- тестирование;
- работа с базами данных;
- обработка JSON;
- обработка XML;
- сессии/cookies;
- гибкий роутинг;
- композиция actions;
- система модулей;
- встроенный шаблонизатор HTML [10];
- встроенные расширяемые формы;
- поддержка ORM;
- кэширование;
- логгеры.

Дополнительным преимуществом фреймворка является возможность очень быстро написать на нём простое веб-приложение. Благодаря множеству встроенных инструментов, можно сосредоточиться на деталях реализации и описании бизнес-логики, в качестве каркаса используя уже написанный общий функционал.

Таким образом, было дано описание ключевых программных средств для разработки серверной части веб-приложения. Рассмотрим также клиентское приложение. На данный момент единственным клиентским языком программирования, получившим повсеместное распространение, является JavaScript - он исполняется в браузере клиента, используя мощности персонального компьютера конечного пользователя. JavaScript существует с 1995 года, имеет крайне высокую поддержку среди разработчиков и очень развитую экосистему библиотек и других готовых решений [7, 31]. Однако этот язык программирования имеет ряд особенностей, которые могут затруднять написание сложных многокомпонентных клиентских приложений. Одна из них

— это динамическая типизация, означающая, что интерпретатор не владеет информацией о типах переменных. В небольших проектах при малом количестве скриптов это приемлемо, однако в системах со сложной клиентской логикой динамическая типизация порождает ошибки и трудности в процессе программирования.

Именно поэтому в данном проекте для реализации клиента был выбран другой язык программирования, созданный на основе JavaScirpt — это TypeScript. Его развитие началось в конце 2012 года в компании Microsoft, и он с самого начала стал быстро распространяться среди разработчиков в силу своей гибкости и производительности [15, 25]. Стоит отметить несколько ключевых особенностей языка:

1. TypeScript — это строго типизированный и компилируемый язык. Результатом работы компилятора является исходный код на языке JavaScript, который затем интерпретируется браузером как обычно. Однако за счёт предварительной компиляции с полной проверкой типов, вводимых языком TypeScript, в итоговом коде уменьшается количество потенциальных ошибок.
2. TypeScript вводит полную поддержку концепций объектно-ориентированного программирования, таких как инкапсуляция, наследование, полиморфизм и так далее. За счёт этого становится проще писать компонентные клиентские программы и реализовывать прикладную область конкретными сущностями, описанными в классах.
3. Лучшая поддержка в ИСР. Из-за динамической типизации в исходном коде JavaScript повсеместно встречаются синтаксические конструкции, которые нельзя интерпретировать однозначно. Это затрудняет построение и разбор синтаксического дерева современными ИСР и не позволяет максимально раскрыть их возможности по подсветке синтаксиса, переходу к определению и интерактивной справке об элементах кода. TypeScript решает проблему однозначности конструкций за счёт строгой типизации,

обеспечивая ИСР всеми необходимыми данными для помощи разработчику.

В конечном итоге для реализации клиента был выбран именно TypeScript, в силу всех вышеперечисленных преимуществ. Однако в данной работе клиентское приложение является достаточно сложным, поэтому для его реализации потребовались дополнительные инструменты. Главнейшим таким инструментом стал фреймворк Vue.js.

Vue.js — это JavaScript-фреймворк с открытым исходным кодом для создания пользовательских интерфейсов. Он был создан бывшим сотрудником Google Эваном Ю, его первый выпуск состоялся в 2014 году [17, 18]. Основной концепцией фреймворка является прогрессивная и гибкая разработка компонентных клиентских приложений. Среди достоинств и возможностей Vue.js можно перечислить следующие:

1. Компонентный подход. Фреймворк Vue.js имеет компонентную структуру. В данном контексте компонент - это файл с расширением «.vue». Такой файл содержит в себе три блока: шаблон на языке гипертекстовой разметки HTML, логику на языке программирования JavaScript или TypeScript, и описание стилей на CSS [2, 4]. Таким образом, каждый компонент инкапсулирует всё, что необходимо для его отображения и работы. Это противопоставляется классическому подходу разработки сайтов, при котором стили, логика и разметка страницы размещаются в отдельных файлах. Компонентная структура удобна тем, что пользовательский интерфейс разделяется на элементы, которые легко переиспользовать. Это уменьшает дублирование кода, упрощает вёрстку и помогает в тестировании логики.
2. Виртуальный DOM. DOM (англ. Document Object Model) — это способ структурного представления данных с помощью объектов. Все веб-сайты сети Интернет, использующие HTML, построены согласно структуре DOM. Однако этот способ никогда не был рассчитан для создания динамического пользовательского интерфейса, и поэтому в современных

веб-приложениях, состоящих из большого количества интерактивных, постоянно меняющихся элементов, оригинальный DOM неприменим из-за проблем с производительностью. Виртуальный DOM — это легковесная копия оригинального дерева объектов, которая позволяет вносить множество изменений и лишь в определённые моменты применять изменения к реальному DOM. Vue.js полностью поддерживает такой подход и благодаря этому является одним из самых быстрых клиентских фреймворков на рынке [16].

3. Низкий порог вхождения и обширная документация. Из всех фреймворков, получивших мировое распространение, Vue.js является самым молодым. Его создатели учли многие проблемы предыдущих решений и в том числе это отразилось на упрощении основных концепций, а также создании подробной и понятной документации [19]. Благодаря этому разработчик может приступить к написанию своего первого приложения уже после одного дня ознакомления с Vue.js, в то время как с другими фреймворками на это могут уйти недели.
4. Поддержка TypeScript. Vue.js, изначально созданный только для JavaScript, имеет также полную совместимость с его надмножеством TypeScript. Это позволяет организовывать сложную логику клиентского приложения, пользуясь всеми преимуществами статической типизации.

Для логики клиента в данной работе также использовались приёмы из парадигмы функционального программирования и специализированные библиотеки TypeScript, которые эти приёмы реализуют [21, 22]. До введения определения функционального стиля стоит сначала рассмотреть классическую парадигму, которая противопоставляется первой — это императивное программирование. Так можно будет произвести сравнительный анализ двух подходов и сделать вывод, какой из них наиболее предпочтителен в данном проекте.

Императивное программирование — это парадигма программирования (стиль написания исходного кода компьютерной программы), для которой характерно следующее:

- в исходном коде программы записываются инструкции (команды);
- инструкции должны выполняться последовательно;
- при выполнении инструкции данные, полученные при выполнении предыдущих инструкций, могут читаться из памяти;
- данные, полученные при выполнении инструкции, могут записываться в память.

Императивная программа похожа на приказы (англ. imperative — приказ, повелительное наклонение), выражаемые повелительным наклонением в естественных языках, то есть представляют собой последовательность команд, которые должен выполнить компьютер [28].

Большая часть современных языков, таких как C, C++, Java, используют именно такую парадигму. При подобном подходе очень активно используются операции присваивания, именованные переменные, составные выражения, подпрограммы (модули) и так далее. Эти элементы относительно просты для понимания, однако они увеличивают сложность модели вычислений, из-за чего императивные программы подвержены ошибкам, специфичным для этой парадигмы.

При этом, хоть функциональный стиль написания ПО и более сложен для понимания и восприятия, он во многих аспектах позволяет выстраивать более надёжные, оптимизированные программы, а также писать их быстрее.

Определение из Википедии: «Функциональное программирование – раздел дискретной математики и парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм в процедурном программировании).

Противопоставляется парадигме императивного программирования, которая описывает процесс вычислений как последовательное изменение

состояний (в значении, подобном таковому в теории автоматов). При необходимости, в функциональном программировании вся совокупность последовательных состояний вычислительного процесса представляется явным образом, например, как список» [36].

Среди сильных сторон функциональной парадигмы называют следующие:

- Повышение надёжности кода. Привлекательная сторона вычислений без состояний — повышение надёжности кода за счёт чёткой структуризации и отсутствия необходимости отслеживания побочных эффектов. Любая функция работает только с локальными данными и работает с ними всегда одинаково, независимо от того, где, как и при каких обстоятельствах она вызывается. Невозможность мутации данных при пользовании ими в разных местах программы исключает появление труднообнаруживаемых ошибок (таких, например, как случайное присваивание неверного значения глобальной переменной в императивной программе).
- Удобство организации модульного тестирования. Поскольку функция в функциональном программировании не может порождать побочные эффекты, менять объекты нельзя как внутри области видимости, так и снаружи (в отличие от императивных программ, где одна функция может установить какую-нибудь внешнюю переменную, считываемую второй функцией). Единственным эффектом от вычисления функции является возвращаемый ей результат, и единственный фактор, оказывающий влияние на результат — это значения аргументов. Таким образом, имеется возможность протестировать каждую функцию в программе, просто вычислив её от различных наборов значений аргументов. При этом можно не беспокоиться ни о вызове функций в правильном порядке, ни о правильном формировании внешнего состояния. Если любая функция в программе проходит модульные тесты, то можно быть уверенным в качестве всей программы. В императивных программах проверка возвращаемого значения функции недостаточна: функция может

модифицировать внешнее состояние, которое тоже нужно проверять, чего не нужно делать в функциональных программах.

- Возможности оптимизации при компиляции. Традиционно упоминаемой положительной особенностью функционального программирования является то, что оно позволяет описывать программу в так называемом «декларативном» виде, когда строгая последовательность выполнения многих операций, необходимых для вычисления результата, в явном виде не задаётся, а формируется автоматически в процессе вычисления функций. Это обстоятельство, а также отсутствие состояний даёт возможность применять к функциональным программам достаточно сложные методы автоматической оптимизации.
- Возможности параллелизма. Ещё одним преимуществом функциональных программ является то, что они предоставляют широчайшие возможности для автоматического распараллеливания вычислений. Поскольку отсутствие побочных эффектов гарантировано, в любом вызове функции всегда допустимо параллельное вычисление двух различных параметров — порядок их вычисления не может оказать влияния на результат вызова [32].

Также в качестве отдельного преимущества можно отметить, для некоторых языков, ускорение разработки за счёт использования более кратких конструкций, нежели при императивном подходе. Так, например, одна из самых распространённых в программировании задач – перебор значений коллекции – обычно решается в императивных языках описыванием цикла с объявлением дополнительных переменных-счётчиков. В то время как в функциональных языках обычно предусмотрены крайне мощные и разнообразные методы встроенной библиотеки для работы с коллекциями, и операция перебора зачастую решается в одну строчку кода.

Таким образом, для написания клиентского приложения целесообразнее использовать именно функциональный стиль программирования, так как в сочетании со статической типизацией языка TypeScript это даёт возможность

разработать крайне надёжную, хорошо протестированную и быструю программу.

1.3. Формализованное описание технического задания

Техническое задание на разработку информационной системы для ведения сделок на бирже. Составлено на основе ГОСТ 34.602-89 [27].

1. Общие сведения.
   1.1. Название организации-заказчика.
   ФГБОУ ВО «УрГПУ»
   1.2. Название продукта разработки (проектирования).
   «Информационная система для ведения сделок на бирже».
   1.3. Назначение продукта.
   Система предназначена для хранения истории биржевых сделок, автоматизации их создания и вывода статистики.
   1.4. Плановые сроки начала и окончания работ.
   В соответствии с планом выполнения ВКР.
2. Характеристика области применения продукта.
   2.1. Процессы и структуры, в которых предполагается использование продукта разработки.
   Использование физическими лицами (частные инвесторы), трейдерскими организациями, банками и так далее.
   2.2. Характеристика персонала (количество, квалификация, степень готовности).
   - Разработчик: владение навыками проектирования и разработки клиент-серверных веб-приложений (Scala, Play Framework, СУБД PostgreSQL, TypeScript, Vue.js, HTML5, CSS3).
   - Системный администратор: навыки владения серверными ОС семейства Unix, опыт доставки и развёртывания выпусков ПО на

серверную ЭВМ, владение инструментами удалённого администрирования (SSH, RDP).

- Пользователь: базовые навыки в биржевой торговле, владение ПК на базовом уровне, базовые навыки работы в сети Интернет.

3. Требования к продукту разработки.

3.1. Требования к продукту в целом.

Требуется спроектировать и разработать информационную систему для ведения сделок на финансовой бирже. Система должна быть реализована на базе клиент-серверного приложения и должна предоставлять конечному пользователю следующий функционал:

- Возможность создать новую биржевую сделку двумя способами: заполнением обязательных полей вручную или с помощью автозаполнения информации согласно данным от API веб-сайта брокера.
- Редактирование и удаление уже существующих сделок.
- Вывод существующих сделок в виде интерактивной таблицы, с возможностью сортировки столбцов.
- Вывод статистики на основе характеристик уже созданных сделок.

3.2. Аппаратные требования.

3.2.1. Серверная ЭВМ.

- 64- или 32-разрядный процессор, 4 ядра и 8 потоков или более, тактовая частота 3.2 Ггц или более.
- 4 Гб ОЗУ и более.
- Не менее 100 Гб дискового пространства и последующее расширение исходя из роста числа пользователей.
- Выход в сеть Интернет на скорости минимум 10 Мбит\сек. и последующее увеличение исходя из роста числа пользователей.

3.2.2. Рабочая станция (клиент).

- Выход в сеть Интернет на скорости минимум 1 Мбит\сек.

3.3. Указание системного программного обеспечения.

3.3.1. Серверная ЭВМ.

- Операционная система семейства Unix (любой серверной дистрибутив Linux).

3.3.2. Рабочая станция (клиент).

- Любой интернет-браузер с поддержкой HTML5, CSS3, JavaScript.

3.4. Указание программного обеспечения, используемого для реализации.

- Heroku - облачный хостинг для развёртки и запуска приложения.
- IDE JetBrains IntelliJ IDEA - интегрированная среда разработки.
- Для сервера: язык программирования Scala, фреймворк Play, СУБД PostgreSQL.
- Для клиента: язык программирования TypeScript, фреймворк Vue.js, HTML5, CSS3.
- Формат данных JSON для обмена информацией между клиентом и сервером.

3.5. Особенности реализации серверной и клиентской частей.

Серверная часть располагается на выделенной серверной ЭВМ и принимает запросы от клиентской части по протоколу HTTP. Не имеет пользовательского интерфейса. Задачи сервера: взаимодействие с базой данных, обработка сложных запросов, взаимодействие со сторонним сервисом через API брокера.

Клиентская часть также является отдельным приложением и располагается на выделенной ЭВМ (как правило, на той же серверной). В отличие от сервера имеет графический пользовательский интерфейс в виде веб-сайта. Задачами клиентской части является обработка простых запросов, не требующих обращения к базе данных или сторонним сервисам, а также отображение пользовательского интерфейса.

3.6. Форматы входных и выходных данных.

Входные данные представляют из себя записи о новых сделках, характеристики которых заполняются пользователем вручную или загружаются из веб-платформы брокера через API, а также запросы пользователя на

различные операции с этими данными: просмотр, изменение, удаление, вывод статистики.

Выходными данными является содержимое интерфейса веб-сайта, отвечающее запросам пользователя.

3.7. Источники данных и порядок и ввода в систему, порядок вывода, хранения.

Для постоянного хранения данных используется реляционная база данных. Доступ к ней реализуется через СУБД PostgreSQL. На уровне серверного приложения для более простого сопоставления типов данных и обращения к БД используется сторонняя библиотека Slick. Ввод и вывод данных происходит посредством SQL-запросов к БД.

3.8. Порядок взаимодействия с другими системами, возможности обмена информацией.

Для автоматического заполнения данных о сделках пользователя осуществляется взаимодействие со сторонним веб-приложением брокера, которое хранит информацию об этих сделках. Доступ организуется через API брокера, для обмена информацией используется протокол HTTP. В ходе взаимодействия брокеру передаются авторизационные параметры в виде уникального токена авторизации для доступа к приватным данным пользователя о его сделках.

3.9. Меры защиты информации.

Не предусмотрено.

4. Требования к пользовательскому интерфейсу.

4.1. Общая характеристика пользовательского интерфейса.

- Серверная часть: не имеет пользовательского интерфейса.
- Клиентская часть: имеет графический пользовательский интерфейс, отображаемый в браузере в виде веб-сайта.

4.2. Размещение информации на экране, дизайн экрана.

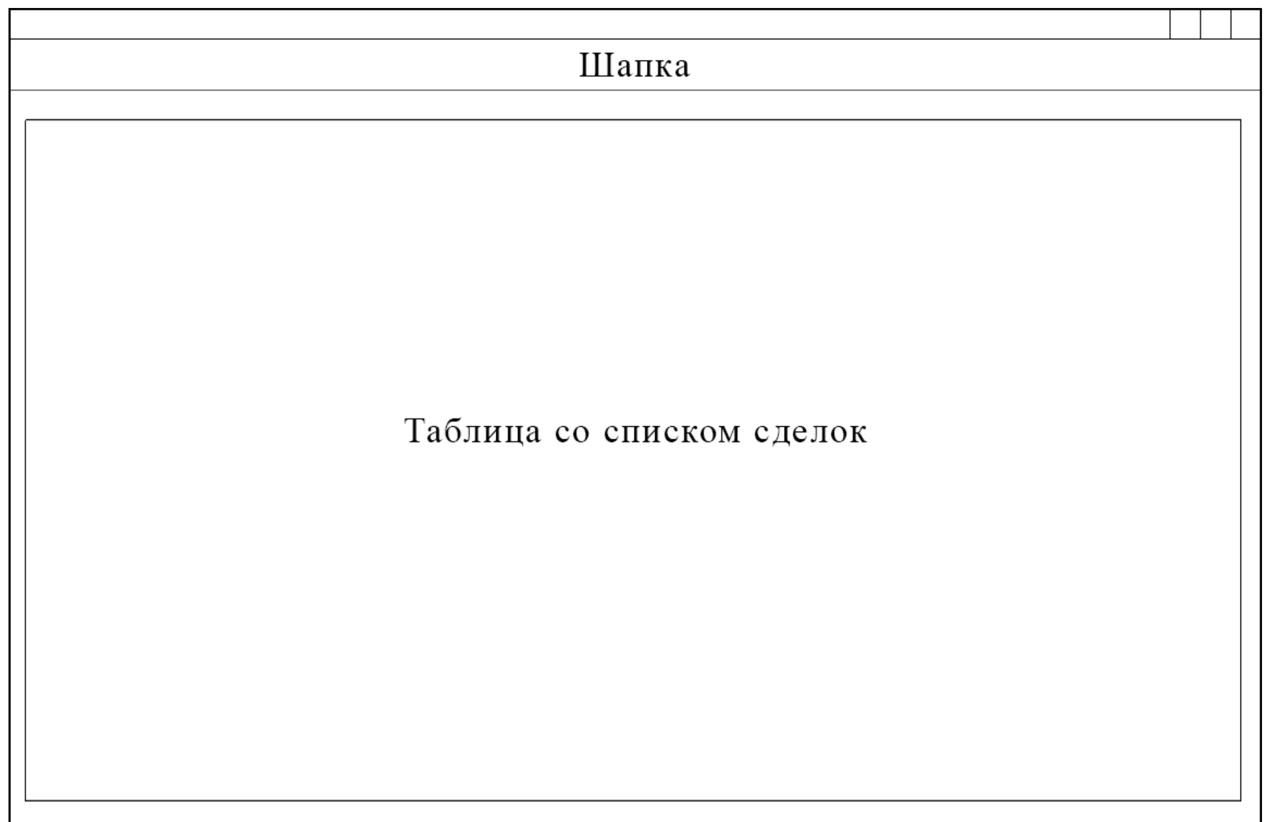


Рис. 1. Интерфейс главной страницы веб-приложения

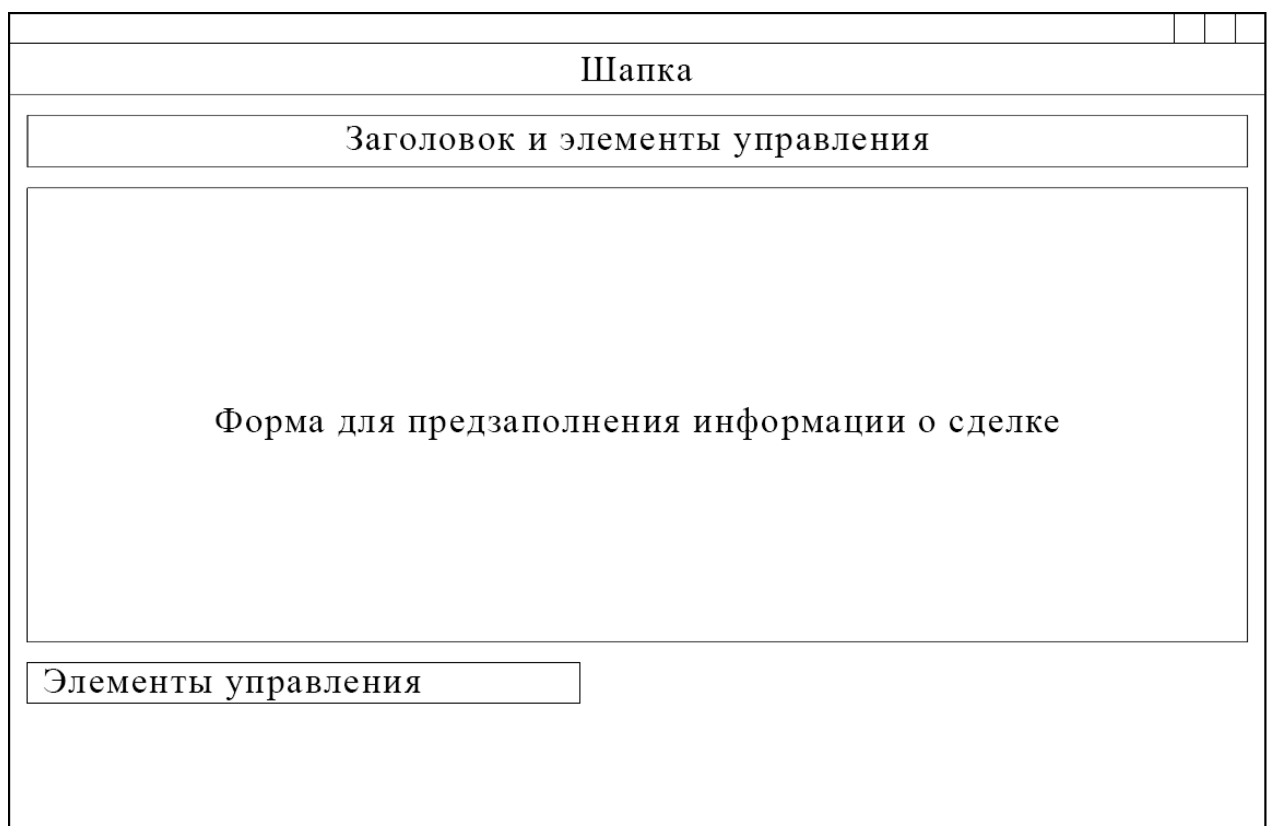


Рис. 2. Интерфейс для предзаполнения информации о сделке

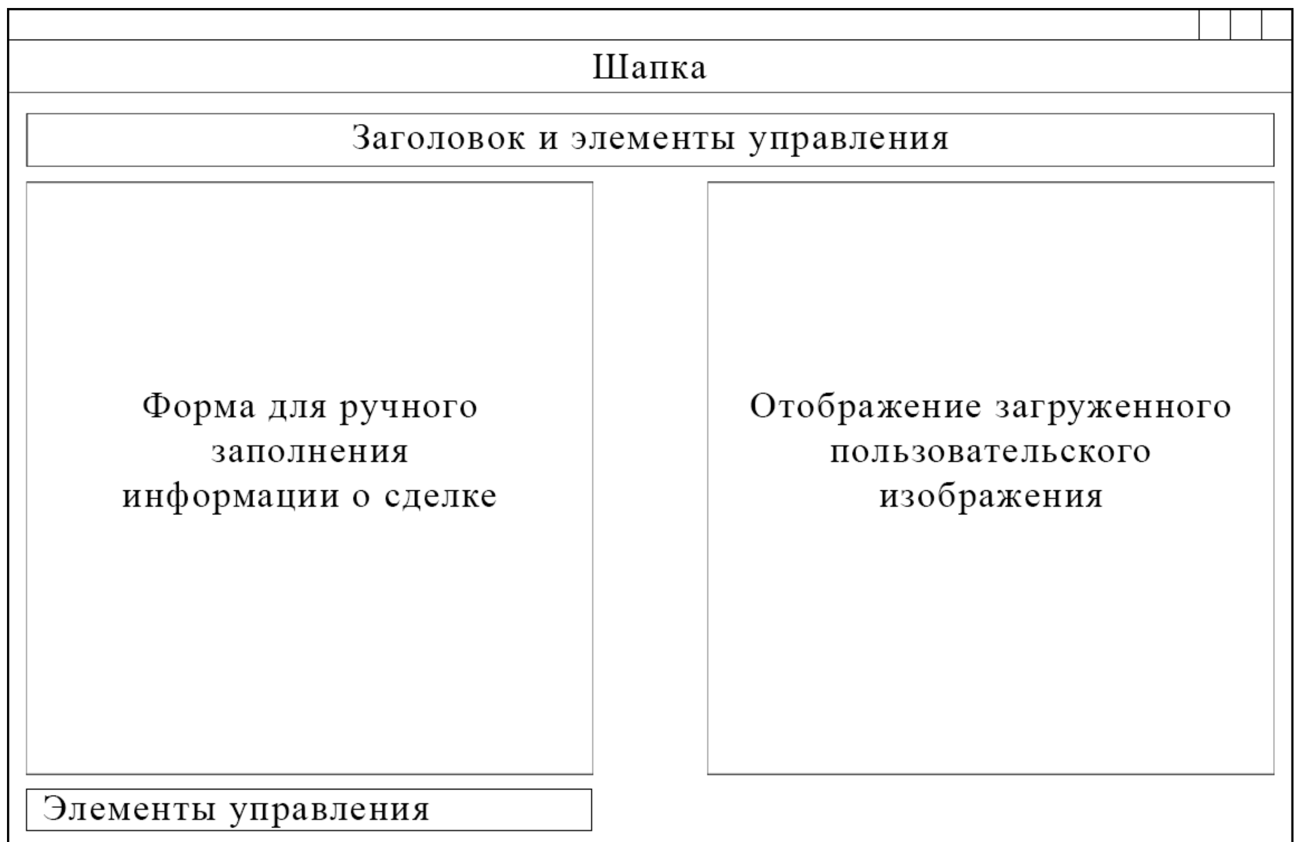


Рис. 3. Интерфейс для ручного заполнения информации о сделке

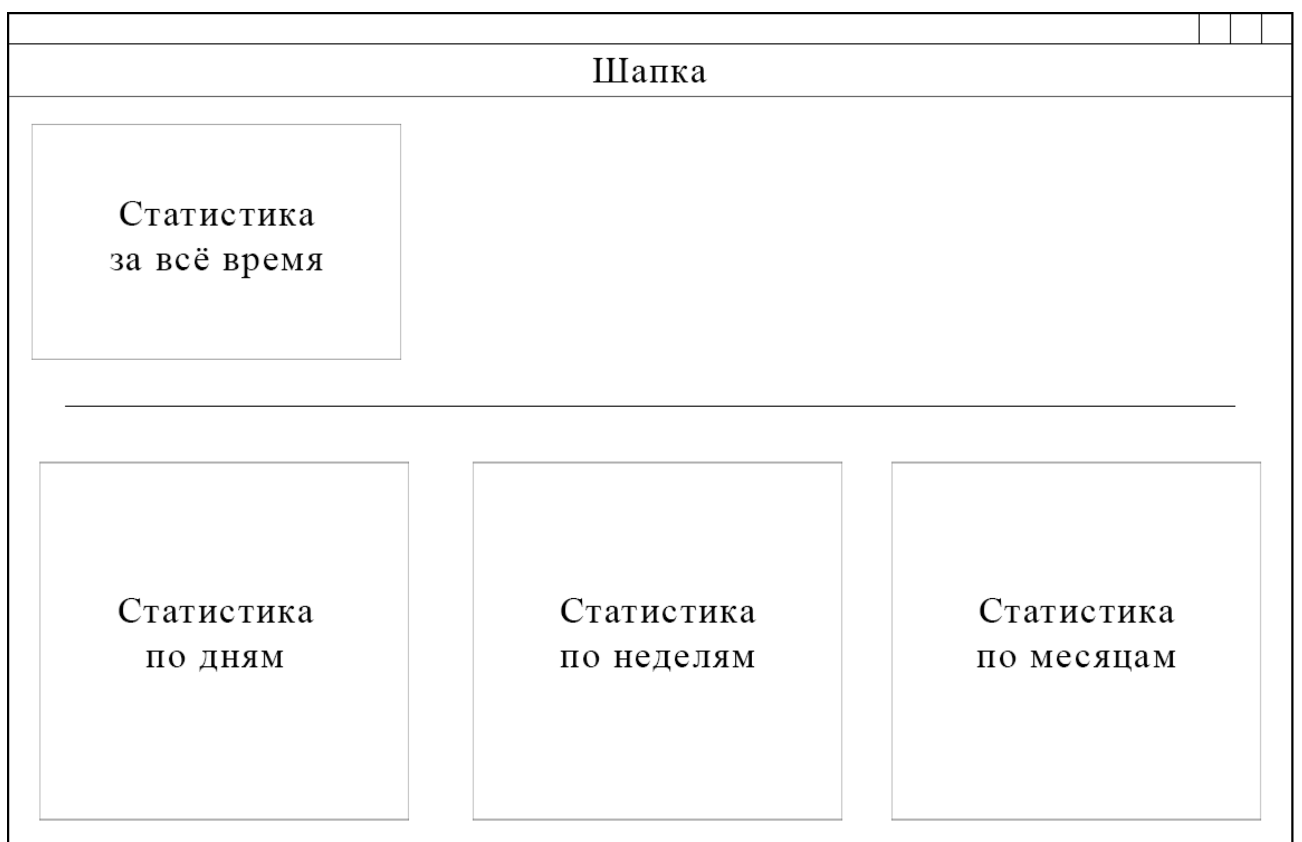


Рис. 4. Интерфейс вывода статистики по сделкам

5. Требования к документированию.

   5.1. Перечень сопроводительной документации.

   Не предусмотрено.

   5.2. Требования к содержанию отдельных документов.

   Не предусмотрено.

6. Порядок сдачи-приёмки продукта.

   В соответствии с планом выполнения ВКР.

Глава II. Разработка клиента и сервера для информационной системы ведения сделок на бирже

2.1. Описание программной реализации клиентского приложения

Опишем структуру клиентской части приложения и её ключевые компоненты. Компонентный подход клиентского фреймворка диктует иерархическую структуру построения проекта. В каждом приложении на Vue.js должен быть главный компонент, который содержит в себе все остальные. Обычно это шаблон с вёрсткой, содержащей самые общие части веб-сайта – «шапку» (англ. header), главное меню, «подвал» (англ. footer). Всё остальное содержимое между ними зависит от текущей страницы и регулируется специальной встроенной библиотекой Vue.js – маршрутизатором vue-router. Маршрутизатор отвечает за связь между компонентами и их относительными адресами. Сопоставляя содержимое адресной строки браузера, маршрутизатор согласно конфигурационному файлу подставляет в текущую страницу необходимый компонент. В данном проекте компоненты, которые содержат вёрстку и логику отдельных страниц, называются «представлениями» (англ. views).

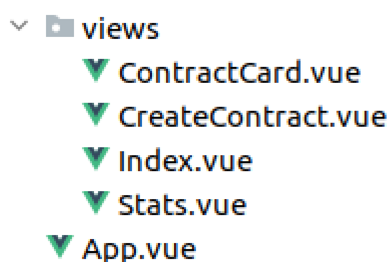


Рис. 5. Файловая структура веб-страниц проекта и главный компонент «App.vue»

Страницы также не содержат большого количества логики и HTML-тегов. Вместо этого они объединяют несколько компонентов меньшего размера, которые представляют из себя отдельные элементы страницы: формы для заполнения данных, таблицы, меню и прочие. На этих базовых, независящих

друг от друга элементов, строится весь веб-сайт, что позволяет легко читать и изменять существующий код.

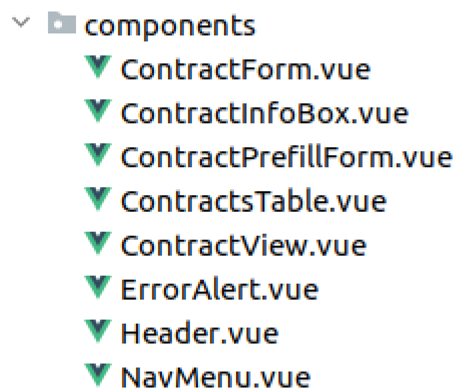


Рис. 6. Файловая структура базовых компонентов

Одним из главных компонентов является «ContractPrefillForm» – он описывает форму для предзаполнения параметров сделки информацией с веб-сайта брокера через API. Сама форма простая и имеет только одно поле, однако производит нетривиальный разбор исходных данных от пользователя и запускает крайне сложный алгоритм на сервере. Программный код этого компонента приведён в [Приложение 1].

Логика всех компонентов оперирует данными, такими как параметры сделки или содержимое сообщения об ошибке. Эти данные объединяются в объекты, чтобы с ними было проще работать. Их отличие от объектов парадигмы объектно-ориентированного программирования в том, что они не содержат логики. Фактически это структуры, которые просто объединяют данные в рамках одной сущности. Данные структуры называются моделями.

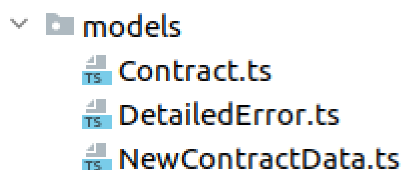


Рис. 7. Файловая структура моделей

Большая часть логики находится в базовых компонентах, однако есть некоторые универсальные для проекта алгоритмы, которые используется в нескольких местах сразу. Они представлены множеством вспомогательных

функций, которые помещены в отдельные разделы структуры исходного кода. Один такой раздел — это «router», который содержит функции, обобщающие перемещение пользователя по веб-страницам проекта. Второй раздел называется «utils», в нём находятся универсальные алгоритмы, упрощающие все остальные аспекты приложения: систему событий, форматирование вывода, распознавание ответов от сервера и другое.

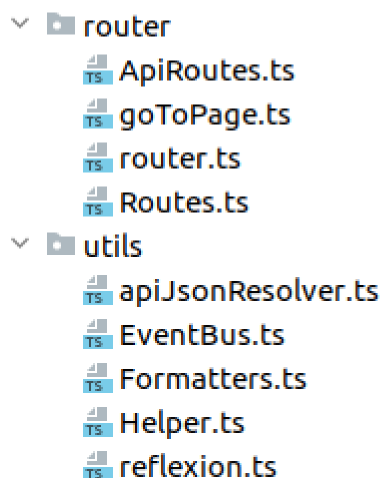


Рис. 8. Файловая структура разделов «router» и «utils»

Таким образом, главные веб-страницы приложения содержат ряд базовых компонентов, которые оперируют моделями и при необходимости обращаются к дополнительной обобщённой логике универсальных разделов. Например, страница создания новой сделки «ContractCard.vue» содержит минимальную логику загрузки страницы и базовый компонент «ContractForm.vue», который представляет из себя форму для заполнения данных. Этот элемент отвечает за проверку корректности ввода пользователя и отправляет данные на сервер. Для отправки он использует модель «Contract.ts» и вспомогательные функции из «apiJsonResolver.ts», которые распознают ответ сервера. Благодаря такой архитектуре достигается разделение ответственности между разными компонентами, что позволяет легко расширять и сопровождать программный код.

2.2. Описание программной реализации серверного приложения

Подобно тому, как клиентский фреймворк «Vue.js» предлагает компонентную структуру проекта, на сервере «Play Framework» также обязывает следовать своему архитектурному дизайну. Он использует шаблон проектирования «Model-View-Controller». Согласно сайту ru.hexlet.io, «MVC расшифровывается как модель-представление-контроллер (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения» [38].

Строгого стандарта реализации этого шаблона проектирования нет, поэтому достаточно распространены его модификации. В данной работе полностью опущена работа с «Представлением», то есть с шаблонами, которые предоставляют пользователю интерфейс. Так как это сервер, он не взаимодействует с пользователем напрямую и в графическом интерфейсе не нуждается.

Компонент «Модель» в веб-приложении представлена как набор сущностей, отражающих прикладную область и также называющимися моделями, и как отдельный класс «Сервис», который обеспечивает работу с этими моделями на уровне базы данных. Задачи «Сервиса»:

- обеспечивать связь программы с базой данных;
- сопоставлять типы данных приложения и базы;
- производить проверку и преобразования данных перед работой с БД там, где это необходимо;
- производить непосредственно сами операции с базой данных: сохранение, удаление, редактирование и выгрузка информации.

В данном проекте используется база данных под управлением СУБД PostgreSQL [11]. На уровне «Контроллера», то есть основной управляющей логики, предусмотрена отдельная обработка ошибок, связанных с базой данных.

Так, если запрашиваемые данные отсутствуют, конфликтуют или появляется любая другая проблема – будет задействован специальный механизм, который сформулирует ошибку и отправит её назад клиенту в том виде, который он сможет распознать. Поэтому при любых ошибках сервера, кроме его полной остановки, конечный пользователь всегда увидит в графическом интерфейсе клиента описание возникшей проблемы.

Рассмотрим основные классы и методы приложения. Так как в архитектуре «MVC» основным управляющим компонентом является «Контроллер», в данном приложении он является главным связующим звеном клиент-серверной архитектуры. «Контроллер» принимает HTTP-запросы от клиента, обрабатывает их и отправляет ответы. Каждый публичный метод контроллера является входной точкой для одного HTTP-запроса.

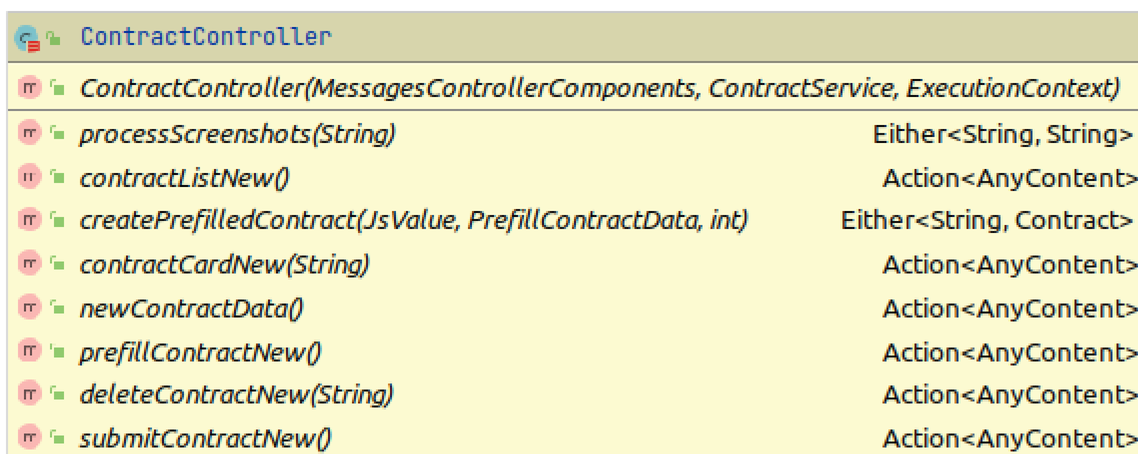


Рис. 9. Класс-контроллер

После обращения к любому из методов производится обращение к базе данных через компонент «Сервиса», на который контроллер имеет ссылку в конструкторе класса. После получения объектной модели нужной сущности, производятся необходимые преобразования данных, обработка возможных ошибок, и ответ на исходный запрос отправляется клиенту.

Самым технически сложным методом является «prefillContractNew» - он связывается с веб-сайтом брокера через API, интерпретирует результат и создаёт новую сделку на основе полученных параметров, при этом контролируя каждый

этап процесса и перехватывая ошибки. Программный код этого метода приведён в [Приложение 2].

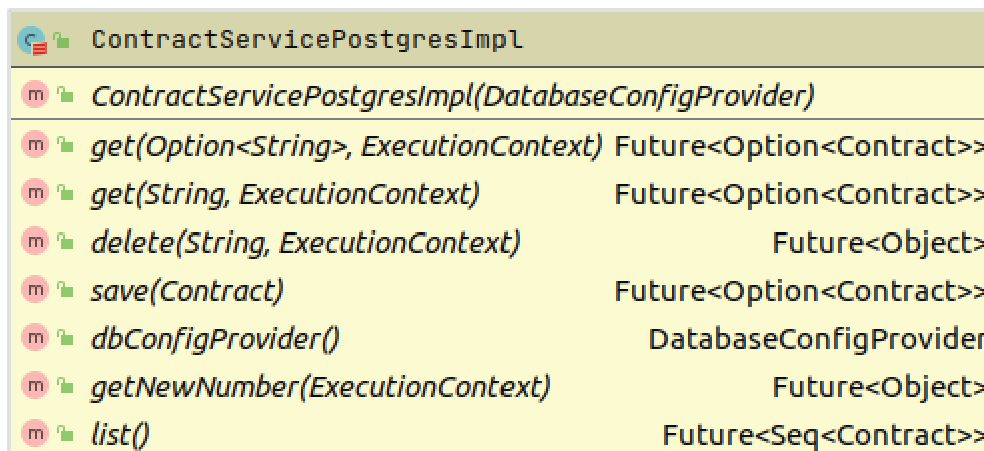


Рис. 10. Класс-сервис

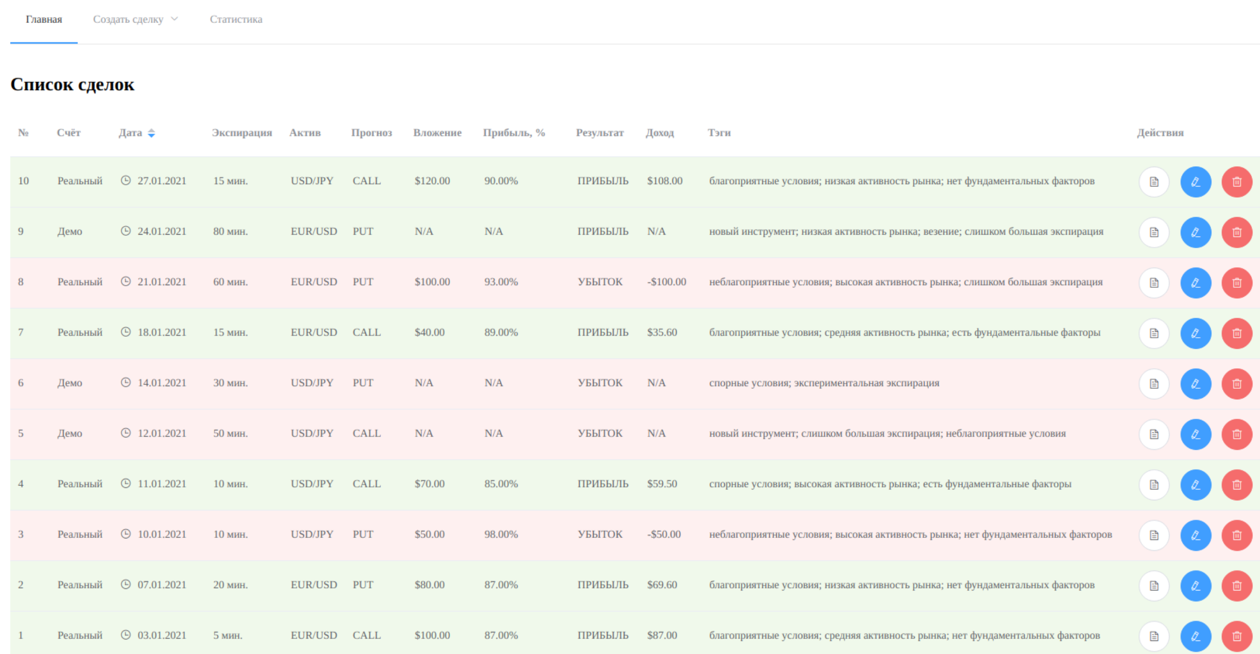
Сервис, к которому контроллер обращается для связи с базой данных, обладает не только базовыми методами для основных операций чтения, сохранения и удаления. Он также имеет вспомогательный метод «getNewNumer», который используется для вычисления, какой следующий номер сделки свободен для создания новой. Помимо этого, сервис содержит дополнительный метод «list», который запрашивает у базы данных весь список сделок целиком. Этот метод используется для того, чтобы отобразить таблицу сделок для пользователя на главной странице клиента.

Почти все методы Сервиса имеют возвращаемый тип вида «Future[Type]». Это позволяет достичь асинхронности при обращении к базе данных. Если веб-приложением будут пользоваться несколько человек и обращения к базе данных будут синхронными, то запрос любого пользователя будет блокировать работу для всех остальных. Благодаря типу-обёртке Future, предоставляемому языком Scala, каждое обращение будет выполняться в отдельном потоке исполнения и не будет влиять на работу других пользователей.

Программный код класса-сервиса приведён в [Приложение 3].

2.3. Описание принципов работы с разработанной информационной системой

Разберём взаимодействие пользователя с веб-приложением. Главная страница веб-приложения выглядит следующим образом:



№	Счёт	Дата	Экспирация	Актив	Прогноз	Вложение	Прибыль, %	Результат	Доход	Тэги	Действия
10	Реальный	27.01.2021	15 мин.	USD/JPY	CALL	$120.00	90.00%	ПРИБЫЛЬ	$108.00	благоприятные условия; низкая активность рынка; нет фундаментальных факторов	
9	Демо	24.01.2021	80 мин.	EUR/USD	PUT	N/A	N/A	ПРИБЫЛЬ	N/A	новый инструмент; низкая активность рынка; везение; слишком большая экспирация	
8	Реальный	21.01.2021	60 мин.	EUR/USD	PUT	$100.00	93.00%	УБЫТОК	-$100.00	неблагоприятные условия; высокая активность рынка; слишком большая экспирация	
7	Реальный	18.01.2021	15 мин.	EUR/USD	CALL	$40.00	89.00%	ПРИБЫЛЬ	$35.60	благоприятные условия; средняя активность рынка; есть фундаментальные факторы	
6	Демо	14.01.2021	30 мин.	USD/JPY	PUT	N/A	N/A	УБЫТОК	N/A	спорные условия; экспериментальная экспирация	
5	Демо	12.01.2021	50 мин.	USD/JPY	CALL	N/A	N/A	УБЫТОК	N/A	новый инструмент; слишком большая экспирация; неблагоприятные условия	
4	Реальный	11.01.2021	10 мин.	USD/JPY	CALL	$70.00	85.00%	ПРИБЫЛЬ	$59.50	спорные условия; высокая активность рынка; есть фундаментальные факторы	
3	Реальный	10.01.2021	10 мин.	USD/JPY	PUT	$50.00	98.00%	УБЫТОК	-$50.00	неблагоприятные условия; высокая активность рынка; нет фундаментальных факторов	
2	Реальный	07.01.2021	20 мин.	EUR/USD	PUT	$80.00	87.00%	ПРИБЫЛЬ	$69.60	благоприятные условия; низкая активность рынка; нет фундаментальных факторов	
1	Реальный	03.01.2021	5 мин.	EUR/USD	CALL	$100.00	87.00%	ПРИБЫЛЬ	$87.00	благоприятные условия; средняя активность рынка; нет фундаментальных факторов	

Рисунок 10. Главная страница

На данной странице пользователь видит весь список своих сделок с их параметрами. Каждая строка со сделкой выделена красным или зелёным цветом в зависимости от того, был ли верным прогноз. Сделки, проводившиеся на демонстрационном счёте брокера, не имеют параметров «Вложение», «Прибыль, %» и «Доход», поэтому в этих столбцах записывается значение «N/A».

В последнем столбце расположены кнопки действий, которые позволяют просмотреть, отредактировать или удалить каждую из сделок. В «шапке» веб-сайта находится навигационное меню, позволяющее перейти к созданию новой сделки или просмотру статистики по текущим.

При создании сделки доступны два варианта: заполнение вручную и предзаполнение. В первом случае пользователю предлагается ввести все параметры сделки самостоятельно:

Номер сделки — 10 + | Вложение, $ — 120.00 +

* Дата сделки 27.01.2021 | Прибыль, % — 90.00 +

* Тип счёта Реальный | Удачно?

* Актив USD/JPY | * Прогноз CALL

* Экспирация, мин. — 15 +

Скриншоты https://sun9-70.userapi.com/impg/dcz5rCCcxOn28wE39un53QD0dSMG_dOo2K0zjA/Pxvlfhrwhl

Тэги благоприятные условия; средняя активность рынка; нет фундаментальных факторов

Описание Тестовая сделка 10

Сохранить Отмена

Рис. 12. Пример заполненной формы ручного создания сделки

В данной форме автоматически проставляется лишь два поля: номер сделки как ближайший свободный и текущая дата в качестве даты сделки. Однако приложение также обладает возможностью запрашивать большую часть данных о сделке напрямую у брокера, поэтому предлагает второй вариант: предзаполнение сделки.

Главная   Создать сделку   Статистика

← Назад | **Предзаполнить сделку**

* Первая строчка: номер транзакции; вторая и последующие: ссылки на скриншоты

Отправить   Отмена

Рис. 13. Форма для предзаполнения сделки

Данная форма имеет всего одно поле многостраничного ввода текста. Для упрощения ввода в проекте принято следующее соглашение: первая строка этого поля должна быть уникальным номером сделки, который можно получить на веб-сайте брокера, вторая и последующие строки должны быть прямыми ссылками на изображения в сети Интернет. Как правило, это графики изменения котировок валют на момент совершения сделки.

После отправки содержимое данной формы распознаётся по вышеуказанному принципу и отправляется на сервер. Далее сервер производит необходимую обработку:

1. Запрашивает у веб-сайта брокера параметры указанной сделки с помощью API.
2. Производит скачивание указанных пользователем изображений.
3. Кодирует изображения в формате Base64.

4. Создаёт новую запись сделки в базе данных с полученными параметрами и преобразованным изображением.

В результате работы данного алгоритма заполняются все поля сделки, кроме тегов и описания. Эти два поля заполняются пользователем самостоятельно.

Такой подход позволяет максимально освободить пользователя от ручной работы и является главным преимуществом данной информационной системы. Подобная автоматизация существенно ускоряет работу и улучшает пользовательский опыт.

В то же время, ещё одной отличительной особенностью проекта является вывод статистики на основе данных, введённых пользователем. Для успешного долгосрочного ведения торговли важно возвращаться к уже совершённым сделкам и анализировать их в разрезе различных временных периодов. Приложение может помочь в этом, агрегируя данные о сделках и выводя их в удобной табличной форме:

Главная Создать сделку Статистика

За всё время

Сумма: $209.70
Всего сделок: 10
Сделки +: 6
Сделки -: 4
% успешных: 60%

По дням

Дата	Сумма за день	Всего сделок	Сделки +	Сделки -	% успешных
27.01.2021	$108.00	1	1	0	100%
23.01.2021	$0.00	1	1	0	100%
20.01.2021	-$100.00	1	0	1	0%
17.01.2021	$35.60	1	1	0	100%
13.01.2021	$0.00	1	0	1	0%
11.01.2021	$0.00	1	0	1	0%
10.01.2021	$59.50	1	1	0	100%
09.01.2021	-$50.00	1	0	1	0%
06.01.2021	$69.60	1	1	0	100%
02.01.2021	$87.00	1	1	0	100%

По неделям

Диапазон дней	Сумма за неделю	Всего сделок	Сделки +	Сделки -	% успешных
25.01.2021 - 31.01.2021	$108.00	1	1	0	100%
18.01.2021 - 24.01.2021	-$100.00	2	1	1	50%
11.01.2021 - 17.01.2021	$35.60	3	1	2	33.33%
04.01.2021 - 10.01.2021	$79.10	3	2	1	66.67%
28.12.2020 - 03.01.2021	$87.00	1	1	0	100%

По месяцам

Месяц и год	Сумма за месяц	Всего сделок	Сделки +	Сделки -	% успешных
Январь 2021	$209.70	10	6	4	60%

Рис. 14. Страница статистики

На данной странице пользователю предоставляется информация не по каждой сделке, а объединённые данные по нескольким, согласно различным временным интервалам. В первую очередь подводится итог за всё время – это

данные всех сделок, которые хранятся в системе на момент загрузки веб-страницы. Суммируются только ключевые характеристики: итоговая сумма полученных и потерянных денежных средств, общее количество сделок, количество удачных и неудачных сделок, а также подведение процента успешности.

Далее пользователю предоставляется три таблицы – с теми же суммированными характеристиками, но по итогам каждого дня, недели и месяца. Как и в общем списке на главной странице, для наглядности прибыльные временные отрезки имеют зелёное цветовое обозначение, а убыточные – красное. Это позволяет быстро выявить неудачные интервалы, найти соответствующие сделки в общей таблице и проанализировать причины убыточности.

Заключение

Таким образом, цель работы достигнута – разработана информационная система для ведения сделок на бирже. В её технологической основе лежит веб-приложение с клиент-серверной архитектурой.

Структура проекта предусматривает существенное расширение как клиента, так и сервера. Клиент построен на компонентной архитектуре, в его иерархическую структуру легко добавлять новые компоненты с минимальными изменениями старых. Таким образом можно быстро расширять приложение новыми веб-страницами.

Сервер использует шаблон проектирования MVC, который также облегчает добавление нового функционала - при необходимости базовый функционал проекта можно расширить новыми классами-контроллерами и моделями. За счёт этого к новым веб-страницам клиента можно также быстро добавлять и новую логику.

Задачи можно считать выполненными:

1. Выполнен анализ прикладной области биржевой торговли, который показал актуальность программного обеспечения для хранения результатов торгов и необходимость автоматизации процессов, таких как сбор данных о сделках и вывод статистики.
2. Произведён разбор программных инструментов для реализации проекта, который позволил обосновать использование языков программирования Scala и TypeScript, а также фреймворков Play и Vue.js, для создания информационной системы ведения сделок на бирже.
3. Разработано веб-приложение с использованием архитектуры «клиент-сервер» в соответствии с техническим заданием. Проект расположен на бесплатном облачном хостинге Heroku и доступен по следующему адресу в сети Интернет: https://bo-trading-journal.herokuapp.com/
4. Описаны основные действия при работе с приложением и приведены примеры его работы: вывод списка сделок; создание новой сделки методом

ручного ввода параметров; создание сделки методом автозаполнения параметров согласно данным от брокера; вывод статистики.

Таким образом, поставленная цель и задачи достигнуты, разработанный проект соответствует всем требованиям технического задания. Работа носит законченный характер.

Список информационных источников

1. Binary.com API. – Текст : электронный // Binary.com Developers : [сайт]. – URL: https://developers.binary.com/ (дата обращения: 20.01.2021).
2. CSS. – Текст : электронный // developer.mozilla.org : [сайт]. – URL: https://developer.mozilla.org/ru/docs/Web/CSS (дата обращения: 20.01.2021).
3. Frontend и backend. – Текст : электронный // proglib.io : [сайт]. – URL: https://proglib.io/p/frontend-backend (дата обращения: 20.01.2021).
4. HTML. – Текст : электронный // developer.mozilla.org : [сайт]. – URL: https://developer.mozilla.org/ru/docs/Web/HTML (дата обращения: 20.01.2021).
5. IntelliJ IDEA. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/IntelliJ_IDEA (дата обращения: 20.01.2021).
6. IntelliJ IDEA. Официальный сайт. – Текст : электронный // jetbrains.com : [сайт]. – URL: https://www.jetbrains.com/ru-ru/idea/ (дата обращения: 20.01.2021).
7. JavaScript. Официальный сайт. – Текст : электронный // javascript.com : [сайт]. – URL: https://www.javascript.com/ (дата обращения: 20.01.2021).
8. Play (фреймворк). – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Play_(фреймворк) (дата обращения: 20.01.2021).
9. Play Framework. Официальная документация. – Текст : электронный // playframework.com : [сайт]. – URL: https://www.playframework.com/documentation/2.8.x/ScalaHome (дата обращения: 20.01.2021).
10. Play Framework. Шаблонизатор HTML. – Текст : электронный // playframework.com : [сайт]. – URL: https://www.playframework.com/documentation/2.8.x/ScalaTemplates (дата обращения: 20.01.2021).

11. PostgreSQL. Официальный сайт. – Текст : электронный // postgresql.org : [сайт]. – URL: https://www.postgresql.org/ (дата обращения: 20.01.2021).
12. Scala (язык программирования). – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Scala_(язык_программирования) (дата обращения: 20.01.2021).
13. Scala. Официальный сайт. – Текст : электронный // scala.epfl.ch : [сайт]. – URL: https://scala.epfl.ch/ (дата обращения: 20.01.2021).
14. Tour of Scala. – Текст : электронный // docs.scala-lang.org : [сайт]. – URL: https://docs.scala-lang.org/ru/tour/tour-of-scala.html (дата обращения: 20.01.2021).
15. TypeScript. Официальный сайт. – Текст : электронный // typescriptlang.org : [сайт]. – URL: https://www.typescriptlang.org/ (дата обращения: 20.01.2021).
16. Virtual DOM. – Текст : электронный // Хабр: [сайт]. – URL: https://habr.com/ru/post/256965/ (дата обращения: 20.01.2021).
17. Vue.js. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Vue.js (дата обращения: 20.01.2021).
18. Vue.js. Официальный сайт. – Текст : электронный // ru.vuejs.org : [сайт]. – URL: https://ru.vuejs.org/index.html (дата обращения: 20.01.2021).
19. Vue.js. Руководство. – Текст : электронный // ru.vuejs.org : [сайт]. – URL: https://ru.vuejs.org/v2/guide/ (дата обращения: 20.01.2021).
20. Архитектура «клиент-сервер». – Текст : электронный // mstu.edu.ru : [сайт]. – URL: http://www.mstu.edu.ru/study/materials/zelenkov/ch_7_1.html (дата обращения: 20.01.2021).
21. Библиотека TypeScript для расширения системы типов. – Текст : электронный // gcanti.github.io : [сайт]. – URL: https://gcanti.github.io/io-ts/ (дата обращения: 20.01.2021).

22. Библиотека TypeScript для функционального программирования. – Текст : электронный // gcanti.github.io : [сайт]. – URL: https://gcanti.github.io/fp-ts/ (дата обращения: 20.01.2021).
23. Биржа. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Биржа (дата обращения: 20.01.2021).
24. Введение в JSON. – Текст : электронный // json.org : [сайт]. – URL: http://www.json.org/json-ru.html (дата обращения: 20.01.2021).
25. Введение в TypeScript. – Текст : электронный // metanit.com : [сайт]. – URL: https://metanit.com/web/typescript/1.1.php (дата обращения: 20.01.2021).
26. ГОСТ 34.321-96. Информационные технологии. Система стандартов по базам данных. Эталонная модель управления данными. – Текст : электронный // Электронный фонд правовой и нормативно-технической документации : [сайт]. – URL: http://docs.cntd.ru/document/gost-34-321-96 (дата обращения: 20.01.2021).
27. ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы – Текст : электронный // Электронный фонд правовой и нормативно-технической документации : [сайт]. – URL: http://docs.cntd.ru/document/gost-34-602-89 (дата обращения: 20.01.2021).
28. Императивное программирование. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Императивное_программирование (дата обращения: 20.01.2021).
29. Интегрированная среда разработки. – Текст : электронный // de.ifmo.ru : [сайт]. – URL: https://de.ifmo.ru/bk_netra/page.php?index=82&layer=1&tutindex=25 (дата обращения: 20.01.2021).
30. Интегрированная среда разработки. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL:

https://ru.wikipedia.org/wiki/Интегрированная_среда_разработки (дата обращения: 20.01.2021).

31. История создания языка программирования JavaScript. – Текст : электронный // web.informatics.ru : [сайт]. – URL: https://web.informatics.ru/works/17-18/web_online/barabanov_n_v/language_js.html (дата обращения: 20.01.2021).
32. Основные принципы функционального программирования. – Текст : электронный // tproger.ru : [сайт]. – URL: https://tproger.ru/translations/functional-programming-concepts/ (дата обращения: 20.01.2021).
33. Рефакторинг. – Текст : электронный // refactoring.guru : [сайт]. – URL: https://refactoring.guru/ru/refactoring (дата обращения: 20.01.2021).
34. Участникам Торгов Московской биржи. – Текст : электронный // Московская биржа : [сайт]. – URL: https://www.moex.com/s1480 (дата обращения: 20.01.2021).
35. Фреймворки в веб-разработке. – Текст : электронный // web-creator.ru : [сайт]. – URL: https://web-creator.ru/articles/about_frameworks (дата обращения: 20.01.2021)
36. Функциональное программирование. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Функциональное_программирование (дата обращения: 20.01.2021).
37. Что такое API. – Текст : электронный // sendpulse.com : [сайт]. – URL: https://sendpulse.com/ru/support/glossary/api-code (дата обращения: 20.01.2021).
38. Что такое MVC. – Текст : электронный // ru.hexlet.io : [сайт]. – URL: https://ru.hexlet.io/blog/posts/chto-takoe-mvc-rasskazyvaem-prostymi-slovami (дата обращения: 20.01.2021).
39. Швагер, Д. Д. Технический анализ. Полный курс / Швагер Д. Д. – Москва : Альпина Паблишер, 2020. – 880 с. – Текст: непосредственный.

40. Щербина, Л. В. Рынок ценных бумаг. Шпаргалка / Л. В. Щербина, Е. В. Гладышева. – Ижевск : Научная Книга, 2009. – 160 с. – Текст : непосредственный.

Приложения

Приложение 1.

ContractPrefillForm.vue

```
@Component
export default class ContractInfoBox extends Vue {
 form = { prefillData: "" }

 rules = {
   prefillData: [
     { required: true, message: 'Это поле обязательно для заполнения', trigger:
'change' }
   ]
 }

 $refs! = {
   form: Form
 }

 loading: boolean = false

 submitForm() {
   (this.$refs.form as Form).validate((valid: boolean) => {
     if (valid) {
       this.loading = true

       const lines = this.form.prefillData.split("\n")
       const transactionId = lines[0]
       const screenshotUrls = O.fold(
         () => "",
         (urls: string[]) => urls.join(";")
       )(A.tail(lines))

       simpleRequest(
         ApiRoutes.prefillContract,
         goToPage.contractForm,
         defaultActionOnError(_ => this.loading = false),
         {
           method: "POST",
           headers: {
```

```
            'Content-Type': 'application/json'
          },
          body: JSON.stringify({
            transactionId: transactionId,
            screenshotUrls: screenshotUrls
          })
        }
      )
    } else {
      return false
    }
  })
}

handleBack() {
  this.$router.back()
}
}
```

Приложение 2.

ContractController.scala#prefillContractNew

```
def prefillContractNew: Action[AnyContent] = Action.async { implicit request =>
 readAndParseJsonWithErrorHandling[PrefillContractData] { prefillContractData =>
   BinaryHelper.getProfitTable.flatMap { profitTableJson =>
     contractService.getNewNumber.flatMap { contractNumber =>
       createPrefilledContract(profitTableJson, prefillContractData,
contractNumber)
         .fold(
           error => ApiError.asAsyncResult(
             caption = "PREFILL CONTRACT PROBLEM",
             cause = "Что-то пошло не так при попытке создать предзаполненную
сделку",
             details = Some(error)
           ),
           prefilledContract =>
             contractService
               .save(prefilledContract)
               .map(_ => Ok(Json.toJson(prefilledContract.id)))
               .recover { case e =>
                 databaseErrorResponse(
                   s"Что-то пошло не так при попытке сохранить " +
                     s"предзаполненную сделку ${prefilledContract.id}", e
                 )
               }
         )
     }.recover { case e =>
       databaseErrorResponse("Что-то пошло не так при попытке получить номер для
новой сделки", e)
     }
   }.recover { e =>
     ApiError.asResult(
       caption = "PROFIT TABLE REQUEST PROBLEM",
       cause = "Что-то пошло не так при попытке запросить список сделок у
брокера",
       details = Some(e.getMessage)
     )
   }
 }
}
```

Приложение 3.

ContractService.scala

```
@Singleton
class ContractServicePostgresImpl @Inject()(protected val dbConfigProvider:
DatabaseConfigProvider) extends ContractService {
 private val dbConfig = dbConfigProvider.get[JdbcProfile]

 import dbConfig._
 import profile.api._

 private class ContractTable(tag: Tag) extends Table[Contract](tag, "Contract")
{
   def id = column[String]("id", O.PrimaryKey)
   def number = column[Int]("number")
   def contractType = column[String]("contractType")
   def created = column[Timestamp]("created")
   def expiration = column[Int]("expiration")
   def fxSymbol = column[String]("fxSymbol")
   def direction = column[String]("direction")
   def buyPrice = column[Option[Double]]("buyPrice")
   def profitPercent = column[Option[Double]]("profitPercent")
   def isWin = column[Boolean]("isWin")
   def screenshotPaths = column[String]("screenshotPaths")
   def tags = column[String]("tags")
   def isCorrect = column[Boolean]("isCorrect")
   def description = column[String]("description")
 }

 private val contracts = TableQuery[ContractTable]

 override def save(contract: Contract): Future[Option[Contract]] = db.run {
   (contracts returning contracts).insertOrUpdate(contract)
 }

 override def delete(id: String)(implicit ec: ExecutionContext): Future[Boolean]
= db.run {
   contracts.filter(_.id === id).delete.map( deletedRows =>
     if (deletedRows > 0) true else false
   )
 }
```

```
 override def get(id: String)(implicit ec: ExecutionContext):
Future[Option[Contract]] = db.run {
   contracts.filter(_.id === id).result.headOption
 }

 override def get(idOpt: Option[String])(implicit ec: ExecutionContext):
Future[Option[Contract]] =
   idOpt.map(id => db.run { contracts.filter(_.id === id).result.headOption
}).getOrElse(Future(None))

 override def list: Future[Seq[Contract]] = db.run {
   contracts.result
 }
}
```