МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

09.03.03 «Прикладная информатика»
(код и наименование направления подготовки / специальности)

Бизнес-информатика
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка приложения для технического анализа инструментов финансового рынка

Обучающийся А.О. Танатаров
(Инициалы Фамилия) (личная подпись)

Руководитель Н.Н. Рогова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

По подсчетам «РБК Инвестиций», капитализация российского рынка акций по состоянию на 28 февраля 2024 года составила ₽61,96 трлн., перспектива роста капитализации российского фондового рынка к 2030 году должна составить 66% от ВВП страны. Об этом президент России Владимир Владимирович Путин сообщил в ходе обращения к Федеральному собранию [5].

В связи с перспективой роста капитилизации, высокой волатильностью и специфичностью Российского фондового рынка, особенно актуальна проблема качественного подбора торговых инструментов и последующего анализа для совершения сделок на российском фондовом рынке.

Целью выпускной квалификационной работы - разработка приложения для технического анализа инструментов финансового рынка. Разрабатываемое программное обеспечение предназначено для повышения эффективности торговли и точности трейдинга посредством качественного отбора торговых инструментов.

Практическая значимость разработки заключается в создании приложения «Скринер», которое позволяет трейдерам и инвесторам автоматизировать процесс отбора, фильтрации и технического анализа торговых инструментов. Это повышает эффективность и точность принятия решений, сокращает временные затраты на анализ и уменьшает риски, связанные с человеческим фактором.

В результате выполнения выпускной квалификационной работы разработанно приложение «Скринер», приложение является эффективным инструментом для трейдеров, оно позволяет улучшить процесс принятия решений при выборе торгового инструмента.

Выпускная квалификационная работа состоит из 74 страниц текста и содержит 14 рисунков, 11 таблиц и 20 источников.

Оглавление

Введение

Современный финансовый рынок характеризуется высокой волатильностью и огромным объемом информации, которую необходимо анализировать для принятия обоснованных инвестиционных решений.

Технический анализ является одним из ключевых методов, используемых трейдерами для прогнозирования будущих движений цен на основе исторических данных. В условиях российского фондового рынка, особую актуальность приобретает разработка специализированных инструментов для анализа инструментов Московской и Санкт-Петербуржской бирж [8]. Разработка приложения для отбора, фильтрации и технического анализа торговых инструментов позволит существенно облегчить и автоматизировать процесс принятия решений, повышая эффективность и точность трейдинга.

В условиях российской экономики и особенностей российского фондового рынка, создание специализированных инструментов для анализа акций становится особенно актуальным по нескольким причинам:

– увеличение количества частный инвесторов в России объема торгов на Московской бирже. Московская биржа (MOEX) является крупнейшей фондовой площадкой России и одной из ведущих бирж в мире. Объемы торгов на бирже продолжают расти, что связано с увеличением числа участников рынка и расширением спектра торгуемых инструментов. В таких условиях трейдерам и инвесторам необходимо эффективное программное обеспечение для обработки больших объемов данных и быстрого анализа рыночных трендов. По данным торговой площадки количество физ. лиц, имеющих брокерские счета на "Московской бирже", по итогам января 2024 года достигло 30,2 млн человек. Суммарный объем вложений частных инвесторов на фондовом рынке Московской биржи в январе 2024 года

вырос на 18% по сравнению с январем 2023 года и составил 63,4 млрд рублей [1].

- высокая волатильность российского рынка. Российский фондовый рынок известен своей высокой волатильностью, что связано с экономическими и политическими факторами, влияющими на стабильность. Высокая волатильность делает прогнозирование движений цен более сложным, но и более необходимым. Технический анализ, используя индикаторы и модели, позволяет лучше понимать рыночные тренды и принимать обоснованные решения в условиях неопределенности.
- недостаток локализованных инструментов. Существующие на рынке инструменты для технического анализа часто ориентированы на международные рынки и не всегда учитывают специфику российского фондового рынка. Например, популярные платформы, такие как MetaTrader и TradingView, предлагают мощные аналитические возможности, но не всегда адаптированы для работы с данными и специфическими требованиями российского рынка. Разработка специализированного программного обеспечения «скринера», будет учитывать особенности российского фондового рынка, позволит инвесторам и трейдерам получать более точные и релевантные данные.
- автоматизация и повышение эффективности анализа. Ручной анализ финансовых данных требует значительных временных и интеллектуальных ресурсов. Автоматизация процессов анализа с помощью специализированных приложений позволяет значительно повысить эффективность работы трейдеров. Программные решения, интегрированные с данными биржи, могут автоматически собирать, обрабатывать и анализировать информацию, предоставляя пользователям готовые результаты и рекомендации.

Таким образом, разработка приложения для технического анализа финансовых инструментов (акций) является актуальной задачей, которая отвечает потребностям современного финансового рынка.

Скринер (в терминалогии трейдеров) — это инструмент, который помогает трейдерам и инвесторам находить финансовые инструменты, соответствующие определённым критериям и фильтрам [7]. Основное предназначение скринера заключается в следующем:

- фильтрация большого количества акций. На фондовых рынках торгуются тысячи акций, и скринер позволяет быстро отсортировать их по заданным параметрам, таким как цена, объём торгов, рыночная капитализация, показатели финансовой отчётности и другие.
- поиск инвестиционных возможностей. Скринер помогает выявлять акции, которые соответствуют инвестиционной стратегии пользователя. Например, он может помочь найти недооценённые акции, акции с высоким потенциалом роста или акции, соответствующие определённым техническим индикаторам.
- оптимизация времени. Вместо того чтобы вручную анализировать каждую акцию, скринер автоматизирует процесс, экономя время и усилия инвесторов.
- анализ рынка. Скринеры часто используются для анализа состояния рынка, определения трендов и поиска сигналов для входа или выхода из позиций.
- поддержка принятия решений. Скринер предоставляет данные и аналитические инструменты, которые помогают трейдерам и инвесторам принимать обоснованные решения на основе текущих рыночных условий и собственной стратегии.

Целью данного исследования является разработка приложения для технического анализа инструментов финансового рынка.

Объект исследования - процесс технического анализа инструментов финансового рынка.

Предмет исследования – программное обеспечение для обработки и формирования списка финансовых инструментов с применением алгоритмов технического анализа для обеспечения быстрого поиска на основании рассчитанных показателей, индикаторов, финансовых коэффициентов.

Для достижения поставленной цели необходимо решить следующие задачи:

- выполнить анализ предметной области;
- провести обзор существующих решений для предоставления информации на основе технического анализа акций на бирже;
- обосновать необходимость разработки информационной системы;
- определить ключевые функциональные требования к разрабатываемому приложению;
- спроектировать приложение;
- реализовать и протестировать спроектированное приложение.
- оценить экономическую эффективность от внедряемого приложения.

В работе представлены три основные главы.

Первая глава включает в себя функциональное моделирование предметной области, в которой проводится анализ предметной области, обоснование необходимости разработки информационной системы.

Вторая глава логического проектирования АИС, описывающая процессы проектирования базы данных, разработку логической и концептуальной модели данных.

В третьей главе описывается архитектура решения, физическая модель данных, реализация программного обеспечения и его тестирование. Так же в главе рассматриваются затраты на проект и оценивается его экономическая эффективность.

Глава 1 Функциональное моделирование предметной области

1.1 Технико-экономическая характеристика предметной области

Компания ООО «А-Лаб Групп» основана в 2008 году, и является относительно молодой компанией, она динамично развивается на рынке Российской Федерации, предоставляя информационно аналитические услуги, услуги по консультации и работе в области компьютерных технологий, услуги по разработке и внедрению программного обеспечения, но основной вид деятельности компании это - заключение свопов, опционов и других срочных сделок российском фондовом рынке.

В 2015 году компания расширила деятельность и основала проп-компанию, а также организовала собственный дилинг. В результате чего в компании активно развиваются 2 направления бизнеса: проп-трейдинг и обучение и подготовка трейдеров.

ООО «А-Лаб Групп» оперирует на различных рынках, таких как акции, фьючерсы, и индексы Московской и Санкт-петербуржской биржах. В компании преимущественно используется «ручная» торговля на высоколиквидных активах, таких как акции и валюта, с акцентом на внутридневную торговлю, включая скальпинг и интрадей-трейдинг. Некоторые стратегии компании используют торговых ботов и высокочастотные алгоритмы (HFT), но их процент в виду низкого профита и высоких рисков относительно мал.

Главной особенностью проп-компании являются условия работы с трейдерами. Трейдеры получают в управление капитал компании, который они используют в торговле [16]. По итогам 2023 года с компанией работает более 500 профессиональных трейдеров из разных стран. На рисунке 1 представлена структура компании.

Обучение и подготовка трейдеров – это второе направление бизнеса компании, оно не столько нацелено на организацию дохода - прибыли, сколько

на обеспечение и развитие потенциала компании, ее роста и развития. Хорошо обученный и успешный трейдер — это основа компании сейчас и перспектива в будущем. Поэтому компания уделяет этому направлению очень много времени и средств. Компания имеет собственную школу для обучения трейдеров, имеет специализированных софт для торговли и обучения торговли на биржах российского фондового рынка, основу обучения составляют стратегии скальпинг и интрадей-трейдинг.

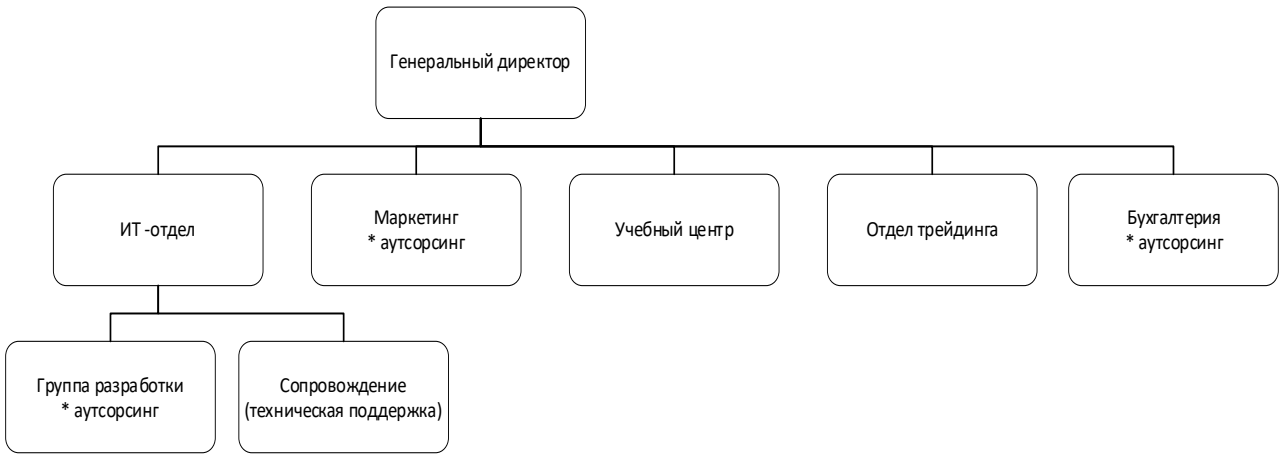


Рисунок 1 - Организационная структура ООО «А-Лаб Групп»

Основную часть прибыли компании обеспечивают торговые стратегии, разработанные трейдерами компании, следовательно, трейдерам нужна оперативная информация о состоянии рынка и аналитические инструменты для принятия взвешенного решения при разработке торговой стратегии. Ученикам в свою очередь нужен инструмент для анализа и закрепления усвоенного материала, применения знаний и умений при торговле на бирже.

Разработка программного обеспечения «Скринер» является стратегически важным этапом в оптимизации работы бизнес-процессов интрадей-трейдинга компании ООО «А-Лаб Групп». Основная идея — это автоматизация процесса подбора финансовых инструментов и принятия торговых решений на основании подобранных данных, исключение при этом человеческого фактора, должно повысить эффективность работы трейдеров,

сократить время на принятие решений входа в сделку и снизить риски. Как отмечает Майк Беллафиоре в книге «Один хороший трейд»: «Вы только настолько хороши, насколько хороши акции, которыми вы торгуете» [9]. Это подчёркивает важность отбора правильных инструментов для торговли.

Стратегической бизнес целью компании является ускорение реакции трейдеров на изменение рынка, автоматизация подбора торговых инструментов и актуальность торговых данных с помощью программного обеспечения позволит эту цель достич [18].

1.2 Концептуальное моделирование предметной области

Концептуальное моделирование предметной области — это один из наиболее критичных этапов проектирования информационных систем. На данном этапе создается структурированное представление всех ключевых элементов и их взаимосвязей, это позволяет обеспечить целостное понимание предметной области. Для успешного моделирования необходимо выбрать наиболее подходящую методологию, которая будет соответствовать специфике и требованиям проекта.

Существует множество методологий для концептуального моделирования, однако наиболее признанными и широко используемыми в индустрии являются три подхода: BPMN, ARIS, и IDEF0. Эти методологии предлагают разнообразные инструменты и методы для моделирования процессов и структур, что делает их универсальными решениями для различных типов проектов. В таблице 1 представлен сравнительный анализ трех вышеупомянутых методологий, для определения их сильных и слабых сторон, и дальнейшего выбора наиболее подходящего варианта для моделирования рассматриваемой предметной области.

Таблица 1 – Сравнение преимуществ и недостатков рассмотренных методологий

Методология	Преимущества	Недостатки
BPMN (Business Process Model and Notation)	Универсальность. BPMN является стандартом для моделирования бизнес-процессов и поддерживается множеством программных инструментов, что обеспечивает его широкое применение. Интуитивность. Схемы BPMN легко воспринимаются как техническими специалистами, так и бизнес-аналитиками, благодаря четко определенным символам и графическому представлению процессов. Гибкость. Методология позволяет моделировать различные аспекты процессов, включая события, потоки данных, задачи и решения, что делает ее подходящей для сложных бизнес-сценариев.	Узкая направленность. BPMN сосредоточен исключительно на моделировании бизнес-процессов и не охватывает такие аспекты, как данные, функции и ресурсы, что ограничивает его применение в комплексных системах. Сложность масштабирования. При моделировании больших и сложных систем модели BPMN могут становиться чрезмерно запутанными и сложными для управления.
ARIS (Architecture of Integrated Information Systems)	Всесторонний подход. ARIS предлагает целостную методологию для моделирования, включающую процессы, данные, функции и организационные структуры, что делает его мощным инструментом для комплексного проектирования систем. Поддержка мощных инструментов. Методология поддерживается продвинутыми инструментами, такими как ARIS Toolset, которые позволяют создавать детализированные и сложные модели. Поддержка на всех этапах. ARIS охватывает все этапы жизненного цикла информационной системы, от концептуального моделирования до реализации, что делает его подходящим для долгосрочных проектов.	Сложность освоения. Внедрение и использование ARIS требует значительных усилий и знаний, что может быть сложно для пользователей без опыта работы с подобными системами. Высокие затраты. Лицензирование и внедрение ARIS могут потребовать значительных ресурсов, что делает его доступным не для всех организаций.

Продолжение таблицы 1

Методология	Преимущества	Недостатки
IDEF0 (Integration Definition for Function Modeling)	Четкость и структурированность. IDEF0 обеспечивает ясную и логичную структуру для функционального моделирования, что облегчает понимание и анализ функциональных зависимостей в системе. Фокус на функциях. Методология идеально подходит для моделирования систем, где важны функциональные зависимости и управление потоками. Признанный стандарт. IDEF0 является стандартом для функционального моделирования в промышленности, что делает его надежным инструментом для создания моделей сложных систем.	Ограниченная область применения. IDEF0 сосредоточен в первую очередь на функциональных аспектах и не предоставляет инструментов для моделирования данных или динамических процессов. Необходимость подготовки. Для корректного чтения и интерпретации схем IDEF0 требуется предварительное обучение, особенно в случае работы со сложными моделями.

Анализ преимуществ и недостатков рассматриваемых методологий продемонстрировал, что оптимальным выбором для моделирования рассматриваемой предметной области является методология IDEF0. Методология IDEF0 (Integration Definition for Function Modeling) ранее известная как технология структурированного анализа и разработки (SADT – Structured Analysis and Design Technique), является стандартом технологии моделирования процессов. IDEF0 эффективно визуализирует функции и процессы, обеспечивая глубокое понимание предметной области, что делает его наилучшим инструментом для достижения поставленных целей нашего проекта [2]. Подход с использованием IDEF0 обеспечивает четкую структуризацию функциональных зависимостей, что особенно важно для создания целостной и понятной модели сложной системы.

1.3 Моделирование бизнес-процессов предметной области для постановки задачи автоматизированного варианта решения

1.3.1 Разработка и анализ модели бизнес-процесса «как есть»

Проведенный анализ основного бизнес-процесса компании - процесс внутридневной торговли (интрадей-трейдинг) торговыми инструментами на бирже, опираясь на методологию IDEF0, была разработана модель «как есть», которая позволит отразить текущую ситуацию, систематизировать протекающие процессы и информационные потоки в рамках этих процессов.

На рисунке 2 представлена функциональная модель процесса интрадей-трейдинга в нотификации IDEF0.

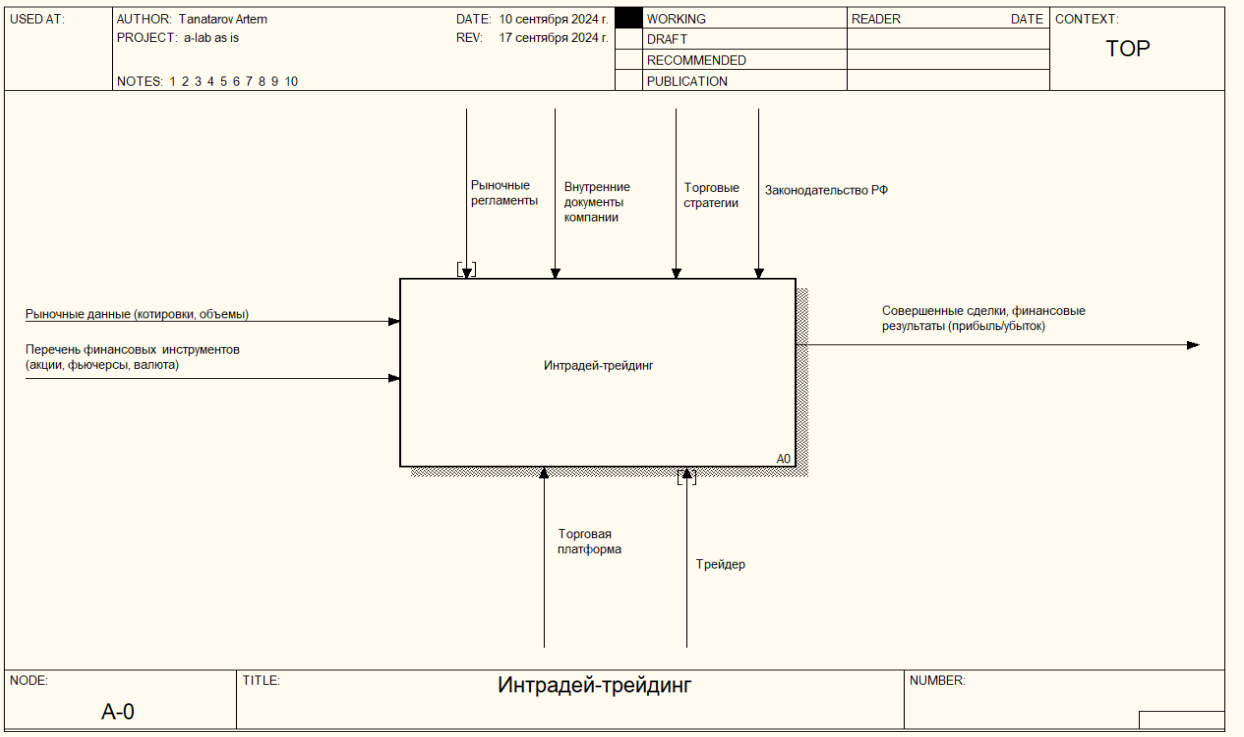


Рисунок 2 –Схема процесса интрадей-трейдинга в нотификации IDEF0 «как есть»

При подробном рассмотрении процесса «интрадей-трейдинга», на вход трейдеру поступают рыночные данные и финансовые инструменты, на выходе

процесса формируются финансовые результаты. Механизмы процесса: трейдер и торговая платформа. Управление: рыночные регламенты, внутренние документы компании, торговые стратегии и законодательство РФ.

Диаграмма декомпозиции процесса «как есть» представлена на рисунке 3.

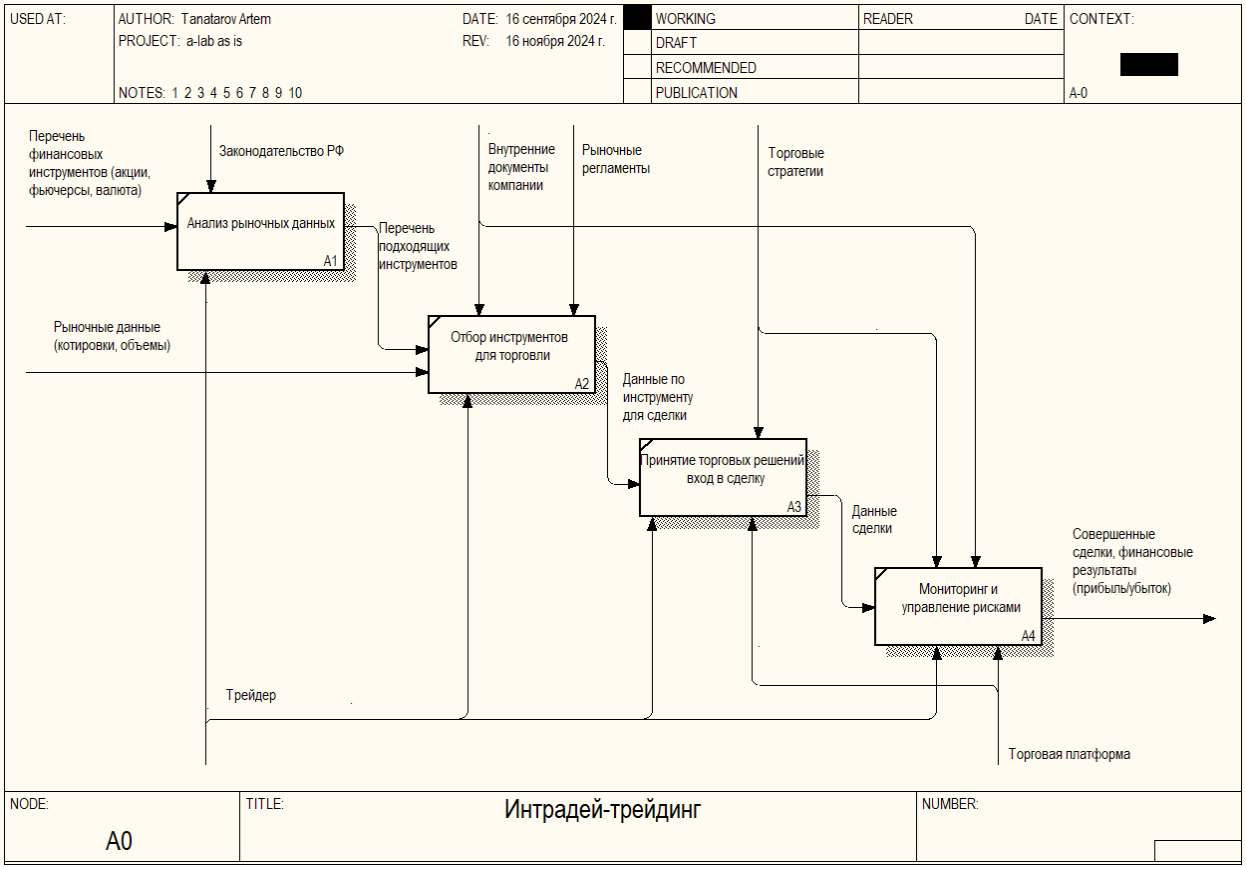


Рисунок 3 –Декомпозиция процесса интрадей-трейдинга «как есть»

Схема демонстрирует процесс интрадей-трейдинга, который состоит из четырех основных процессов (блоков), начиная с анализа рыночных данных и заканчивая мониторингом и управлением рисками. На входе в процесс поступают рыночные данные и перечень финансовых инструментов, а на выходе — совершенные сделки и финансовые результаты.

Процесс A1 «Анализ рыночных данных», трейдер получает из разных источников, (торговые графики, биржевые сводки и специализированные

сайты) на вход: рыночные данные (котировки, объемы), перечень активных финансовых инструментов. На этом этапе трейдеры анализируют поступающие данные с использованием внутренних документов компании и нормативно-правовой базы. Результатом этого анализа является перечень инструментов, которые подходят для дальнейшей торговли.

Процесс А2 «Отбор инструментов для торговли», после анализа данных трейдеры отбирают конкретные инструменты для заключения сделок. Здесь учитываются рыночные регламенты и внутренние стратегии компании. На выходе из процесса получаем данные по инструменту, подходящему для сделки которые соответствуют внутренним стратегиям компании, так же учитываются рыночные регламенты.

Процесс А3 «Принятие торговых решений, вход в сделку», на основе отобранных инструментов трейдер принимает решения о входе в сделки. Это включает в себя размещение ордеров на торговой платформе с учетом установленных стратегий.

Процесс А4 «Мониторинг и управление рисками», трейдеры осуществляют мониторинг текущих позиций и управление рисками, корректируют позиции в зависимости от изменений на рынке и в соответствии с торговыми стратегиями и внутренними документами компании. Результатом является завершение сделок с получением прибыли или убытка.

1.3.2 Обоснование необходимости автоматизированного варианта решения и формирование требований к новой технологии

Интрадей-трейдинг требует быстрого принятия решений и оперативного анализа больших объемов данных. Лучшие практики в этой области включают:

- использование алгоритмических стратегий. Автоматизация принятия решений и использования алгоритмов, основанных на техническом анализе и правилах управления рисками, позволяет ускорить процесс торговли.

- скринеры рынка. Современные трейдеры активно используют скринеры, которые позволяют быстро фильтровать и отслеживать активы по заданным критериям. Это значительно сокращает время на анализ рынка и позволяет находить лучшие торговые возможности.
- интеграция данных в реальном времени. Использование данных с минимальной задержкой (реальное время) дает возможность оперативно реагировать на изменения рынка.
- машинное обучение и искусственный интеллект. Использование машинного обучения для анализа рыночных данных и прогнозирования позволяет улучшить качество принимаемых решений.

В результате проведения анализа процесса «как есть» выявлено, что трейдер много времени и сил тратит на процесс «Анализа рыночных данных». В компании нет единого стандарта или методики анализа и отбора инструментов, весь процесс анализа выполняется на основе знаний, опыта и интуиции трейдера, что несет в себе множество рисков, в том числе человеческий фактор как на уровне торговли, так и на уровне обучения трейдеров и в дальнейшем, от того как трейдера обучат выполнять анализ и выборку инструментов будет зависеть его прибыль и соответственно прибыль компании.

Оптимизация процесса интрадей-трейдинга с использованием программного обеспечения «Скринер» который будет строиться на данных в реальном времени в процессе интрадей-трейдинга позволит:

- сократить время на анализ данных, так как ПО «Скринер» автоматизирует процесс фильтрации рыночных данных по заданным критериям, что позволяет трейдерам быстрее находить подходящие инструменты;
- увеличить точность торговых решений. Программное обеспечение поможет принимать более обоснованные решения, анализируя

данные по заранее определенным торговым стратегиям и правилам управления рисками.

– оптимизировать процесс торговли. С помощью ПО «Скринера» процесс торговли станет более управляемым, прозрачным и эффективным, что приводит к повышению прибыльности.

1.3.3 Сравнительная характеристика существующих разработок

Проведенный анализ существующих решений показал, что на рынке существует множество скринеров, которые предоставляют функционал для анализа финансовых инструментов. Некоторые из популярных решений:

– TradingView: Одна из наиболее популярных платформ с мощным встроенным скринером, который поддерживает множество индикаторов и критериев фильтрации. Преимущества: удобный интерфейс, поддержка множества рынков, настраиваемые фильтры.

– Finviz: Программное обеспечение, предлагающее базовый и продвинутый скринеры для акций. Поддерживает фильтрацию по техническим и фундаментальным показателям.

– MetaStock: Платформа для технического анализа, включающая скринер и инструменты для создания торговых стратегий. Преимущества: глубокий анализ данных, широкий спектр инструментов.

Сравнительный анализ существующих разработок представлен в таблице 2.

Таблица 2 - Сравнение стоимости и функционала существующих решений

Платформа	Мин. стоимость (мес.)	Макс. стоимость (мес.)	Предоставление данных в реальном времени	Доступные рынки	Особенности
TradingView	Бесплатно с ограничениями	$59.95	Только на графиках, обновление каждые 5 сек	Акции, крипто валюты, форекс, товары	Широкий функционал скринеров, удобный интерфейс, гибкая настройка фильтров. Подходит для разных уровней пользователей.

Продолжение таблицы 2

Платформа	Мин. стоимость (мес.)	Макс. стоимость (мес.)	Предоставление данных в реальном времени	Доступные рынки	Особенности
Finviz	Бесплатно (Ограниченные функции скринера, задержка данных до 15 минут, базовые фильтры и аналитика.)	$39.50	Обновление каждые 5 мин	Акции (в основном США)	Удобен для начинающих трейдеров на фондовом рынке, простой интерфейс, но ограничен фильтрами и рынками. Elite версия предлагает анализ в реальном времени.
MetaStock	$59.95	$149.95	Обновление каждые 5 мин	Акции, форекс, фьючерсы, товары	Высокая стоимость, мощный инструмент для глубокого анализа. Подходит для профессиональных трейдеров. Предлагает аналитику как в конце дня, так и в реальном времени.

При всем многообразии решений представленных на рынке у них есть значительные недостатки:

- гибкость и кастомизация. Большинство существующих решений ограничены стандартными функциями и индикаторами. Настройка под специфические стратегии компании требует значительных усилий.
- зависимость от сторонних поставщиков. Использование внешних решений создает зависимость от поставщика в плане технической поддержки и обновлений.
- интеграция данных в реальном времени. Большинство скринеров представляют анализ и расчетные данные технических и фундаментальных показателей за день, неделю, месяц и т.д., что для внутридневной торговли совсем не подходит. Для внутридневной торговли нужны более оперативные данные: 5 секунд, минута, 5, 15, 30 минут и час.

Разработка собственного скринера позволит полностью контролировать процесс, настраивать его под уникальные стратегии и требования компании.

1.4 Постановка задачи на разработку проекта создания/внедрения АИС

Для создания эффективного и функционального приложения для технического анализа акций, необходимо определить ключевые функциональные требования. Эти требования включают как базовые возможности, так и продвинутые функции, обеспечивающие полное удовлетворение потребностей пользователей. Для эффективной работы ПО «Скринера» в контексте интрадей-трейдинга, необходимо учесть следующие функциональные требования:

- реализация настраиваемых фильтров. Возможность создания и настройки фильтров по различным параметрам (объемы, волатильность, индикаторы технического анализа).
- доступ к рыночным данным в реальном времени. Приложение должно обеспечивать доступ к данным о ценах акций, объемах торгов, и другим важным финансовым показателям в режиме реального времени. Это может быть реализовано через API Московской биржи или сторонних поставщиков данных.
- исторические данные. Возможность просмотра и анализа исторических данных для проведения ретроспективного анализа и тестирования стратегий.
- актуализация данных. Регулярное обновление данных для обеспечения точности и надежности анализов и прогнозов.
- интуитивно понятный интерфейс. Приложение должно иметь простой и удобный интерфейс, который будет понятен как начинающим, так и опытным пользователям. Наличие пользовательских подсказок и обучающих материалов также будет полезным.
- наличие индикаторов. Включение популярных технических индикаторов, таких как скользящие средние (SMA, EMA) и другие.

- анализ объема. Функциональность для анализа объема торгов, включая индикаторы объема, такие как объем по цене (Volume by Price), балансовый объем (OBV) и другие.

Определение этих ключевых функциональных требований позволит создать мощное и удобное приложение для технического анализа акций. Приложение должно быть интуитивно понятным и функциональным, чтобы удовлетворять потребности как начинающих, так и опытных трейдеров, и инвесторов.

1.5 Разработка модели бизнес-процесса «как должно быть»

Процесс интрадей-трейдинга компании ООО "А-Лаб Групп" является ключевым элементом, обеспечивающим управление капиталом и получение прибыли на финансовых рынках. Модель бизнес-процесса «как есть» показала, что в текущей системе трейдеры выполняют большинство задач вручную, что создает определенные ограничения по времени и качеству принятия решений. Это связано с необходимостью обработки больших объемов рыночных данных и использования сложных торговых стратегий.

Для того чтобы повысить эффективность и снизить риски неодходимо внедрить автоматизированную систему, которая сможет быстро и точно анализировать данные, фильтровать инструменты и предоставлять готовые решения для пользователей (трейдеров). Автоматизация процесса также позволит минимизировать человеческий фактор, повысив скорость реакции на изменения рыночной ситуации.

На рисунке 4 представлена функциональная модель процесса интрадей-трейдинга «как должно быть» с учетом реинжиниринга процесса.

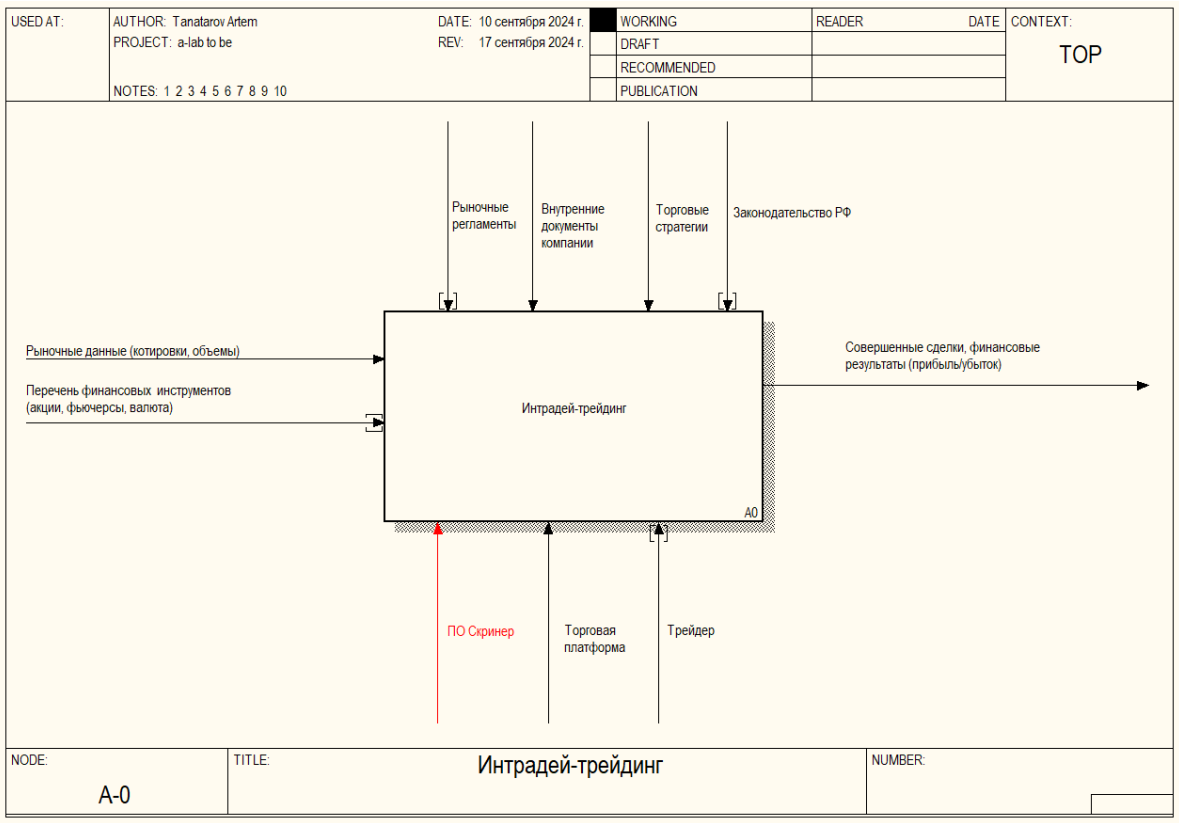


Рисунок 4 – Схема процесса интрадей-трейдинга «как должно быть»

Внедрение программного обеспечения «Скринер» позволит оптимизировать работы и повысить эффективность принятия решений. В процессе реинжиниринга бизнес-процесса интрадей-трейдинга будет детально рассмотрено, как должно быть организовано взаимодействие трейдеров с новым ПО, а также какие изменения произойдут в отдельных этапах принятия торговых решений.

Для понимания изменений в процессе, разработана новая модель бизнес-процесса «как должно быть». Основные изменения касаются двух этапов процесса: отбора инструментов для торговли и принятия торговых решений. ПО «Скринер» интегрировано в эти блоки, что позволяет автоматизировать часть задач, ускорить их выполнение и повысить продуктивность.

На рисунке 5 представлена декомпозиция модели процесса интрадей-трейдинга «как должно быть» с учетом реинжиниринга.

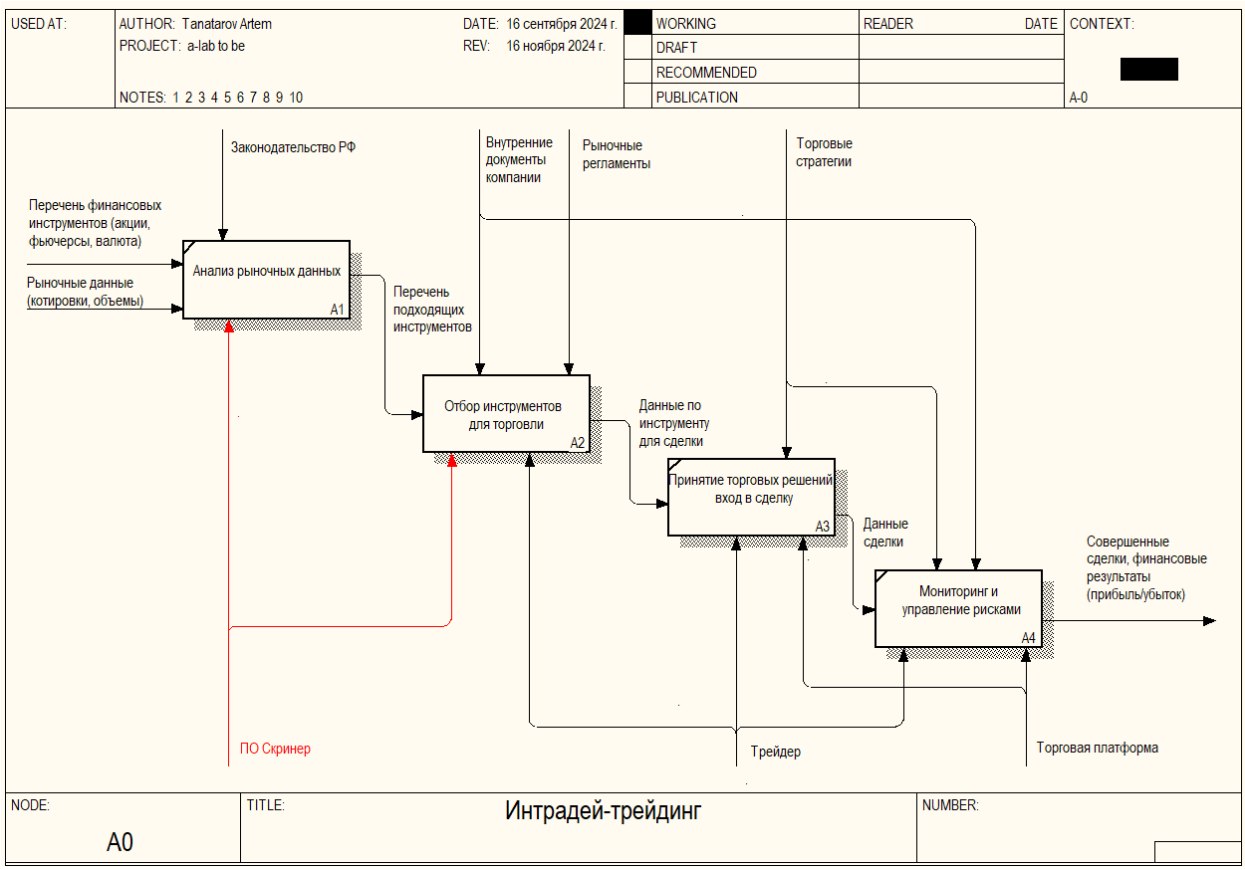


Рисунок 5 – Декомпозиция процесса «как должно быть»

Процесс A1 «Анализ рыночных данных», выполняется в автоматическом режиме ПО «Скринер», трейдер не участвует в данном процессе и не тратит время на анализ. Результатом этого анализа является перечень инструментов, которые подходят для дальнейшей торговли.

Процесс A2 «Отбор инструментов для торговли» является одним из ключевых этапов интрадей-трейдинга. В процессе «как есть» трейдеры вручную анализировали данные рыночных котировок и объемов, что отнимало значительное количество времени и могло привести к субъективным решениям. ПО «Скринер» позволяет трейдерам быстро отбирать финансовые инструменты на основе заранее определенных критериев, таких как уровень волатильности, объемы торгов, и другие факторы, индикаторы. Входом в процесс является: рыночные данные (котировки, объемы), перечень активных финансовых инструментов. Результатом процесса является инструмент(ы) для

сделки (конкретный актив, выбранный для торговли). Механизмы (исполнители): Трейдер, ПО «Скринер» — автоматизирует процесс отбора инструментов по критериям, настраиваемым трейдерами. Управляющие факторы: законодательство РФ, внутренние документы компании, рыночные регламенты.

ПО «Скринер» на этом этапе позволяет ускорить процесс анализа за счет автоматизации рутинных действий. Оно отсеивает малоперспективные инструменты и концентрирует внимание трейдера на активах, соответствующих его стратегиям и текущей рыночной ситуации.

Процесс А3 «Принятие торговых решений, вход в сделку», на основе отобранных инструментов трейдер принимает решения о входе в сделки. Это включает в себя размещение ордеров на торговой платформе с учетом установленных стратегий. С использованием ПО «Скринер» процесс принятия решений становится более автоматизированным. Этот анализ строится на основе технических индикаторов и сигналов, предоставляемых скринером.

Процесс А4 «Мониторинг и управление рисками», этот процесс практически не изменяется по сравнению с моделью «как есть». Трейдер продолжает мониторинг открытых позиций и управляет рисками, используя торговую платформу. Основное внимание уделяется контролю за текущими сделками, внесению корректировок в зависимости от изменения рыночных условий, а также своевременному выходу из сделок.

Рассмотрим преимущества и результаты бизнес-процесса «как должно быть». Внедрение ПО «Скринер» в процесс интрадей-трейдинга дает ряд ключевых преимуществ:

- ускорение анализа данных. Программное обеспечение автоматизирует отбор инструментов для торговли, позволяя трейдерам сосредоточиться на принятии решений, а не на рутинных операциях по анализу.

– увеличение количества сделок. Уменьшение времени на анализ и принятие решений позволяет трейдерам совершать больше сделок в течение торгового дня, что увеличивает общую прибыль.
– снижение рисков. Благодаря точному и своевременному анализу, ПО «Скринер» помогает трейдерам лучше управлять рисками, особенно в условиях высокой волатильности.

Таким образом, бизнес-процесс «как должно быть» обеспечит улучшение работы трейдеров и оптимизирует процессы интрадей-трейдинга. Это решение способствует повышению прибыльности операций, снижению временных затрат на анализ и принятию решений.

Выводы по главе 1.

В результате анализа и моделирования предметной области компании, были выявлены ключевые бизнес-процессы, связанные с внутридневной торговлей на российском фондовом рынке. Проведённый анализ текущего процесса «как есть» показал необходимость внедрения автоматизированной информационной системы (АИС) для повышения эффективности работы трейдеров и снижения рисков, связанных с человеческим фактором.

Внедрение программного обеспечения «Скринер» позволит автоматизировать процесс отбора торговых инструментов, обеспечит быструю и точную обработку данных, а также уменьшит временные затраты на анализ. Модель процесса «как должно быть» наглядно демонстрирует преимущества использования автоматизированного подхода и подчеркивает важность разработки и внедрения АИС для оптимизации бизнес-процессов компании.

Сравнительная характеристика существующих решений на рынке, показала, что разработка собственного решения является более целесообразной для удовлетворения специфических потребностей компании и обеспечения интеграции с данными биржи.

Глава 2 Логическое проектирование АИС

2.1 Выбор технологии логического моделирования АИС

Логическое проектирование автоматизированной информационной системы (АИС) — это важнейший этап разработки, который включает в себя моделирование ключевых аспектов системы на основе ее бизнес-процессов, функциональных требований и данных. Для успешного проектирования системы необходимо выбрать правильные технологии моделирования, которые позволят эффективно представить все взаимодействия между компонентами системы, а также учесть возможную интеграцию с внешними системами и базами данных.

Для проектирования программного обеспечения в данной работе используются следующие технологии: логическое моделирование и UML. UML-диаграммы используются для визуализации и документирования различных аспектов системы, включая классы, объекты, процессы и взаимодействия. Это позволяет разработчикам и заинтересованным сторонам иметь общее понимание структуры и поведения системы.

Преимущество логических моделей и UML заключается в их универсальности и способности детально описать сложные системы. Они обеспечивают стандартизированный способ представления системных компонентов и их взаимодействий, что облегчает коммуникацию между разработчиками, аналитиками и другими участниками проекта.

UML-диаграммы (Unified Modeling Language). UML — это универсальный язык моделирования, который включает в себя диаграммы классов, диаграммы вариантов использования, диаграммы последовательностей и другие инструменты [20]. UML-диаграммы используются для описания логической структуры системы, взаимодействия компонентов и бизнес-логики.

Диаграммы вариантов использования: для отображения взаимодействия трейдеров с системой через интерфейс (например, фильтрация и сортировка финансовых инструментов).

Диаграммы классов: для представления объектов системы и их атрибутов, включая финансовые инструменты, настройки фильтров и отчеты.

В результате анализа технологий для логического моделирования системы выбраны ER-диаграмма, UML, и концептуальные схемы. Эти инструменты обеспечивают всестороннее моделирование всех аспектов работы программного обеспечения «Скринер», включая взаимодействие с пользователями, обработку данных и интеграцию с внешними API.

2.2 Логическая модель АИС и ее описание

В разрабатываемом программном обеспечении использована многослойная системная архитектура, объединяющая клиентскую и серверную части, базу данных и интеграцию с внешними источниками данных. Архитектура разрабатывается с целью обеспечения высокой производительности, надежности и масштабируемости, необходимых для работы с данными финансовых рынков в реальном времени. В описании представлены компоненты системной архитектуры.

Клиентская часть (Frontend) программного обеспечения, планируется в реализацию с использованием фрэймворка Django [13], что позволяет использовать единый стек для фронтенда и бэкенда. На странице дополнительно применятюся Java-приложения для улучшения функциональности отображения данных и взаимодействия с пользователем, сортировке, фильтрации и т.д.

Технологический стек клиентской части:

- HTML, CSS, JavaScript - основные технологии для построения веб-интерфейса.

- Bootstrap - используется для адаптивной верстки и стилизации компонентов пользовательского интерфейса.
- Java-приложение - встроено на странице и отвечает за обработку и визуализацию данных в режиме реального времени.
- WebSockets - применяется для обеспечения связи в реальном времени между клиентом и сервером, позволяя моментально обновлять данные (например, котировки и графики). Протокол WebSocket — это независимый протокол, основанный на протоколе TCP. Он делает возможным более тесное взаимодействие между браузером и веб-сайтом, способствуя распространению интерактивного содержимого и созданию приложений реального времени [3].

Серверная часть отвечает за реализацию бизнес-логики и обработку данных, поступающих от клиента и внешних источников. В проекте используется Django как фреймворк для создания API и управления взаимодействием с базой данных.

Технологический стек серверной части:

- Django - фреймворк, используемый как для фронтенда, так и для бэкенда. Django отвечает за обработку запросов, взаимодействие с базой данных и клиентской частью.
- Библиотеки Django Channels, Daphne - библиотека для работы с WebSockets, которая позволяет обмениваться данными в реальном времени между сервером и клиентом.
- SQLAlchemy - ORM для работы с СУБД PostgreSQL, упрощает выполнение сложных SQL-запросов и управление транзакциями.
- Библиотека Celery –используется для выполнения задач в фоновом режиме, таких как обновление данных по API или генерация отчетов.

Фрэймворк Django предоставляет комплексное решение для разработки как фронтенда, так и бэкенда, что упрощает интеграцию и снижает затраты на разработку. Daphne обеспечивает асинхронную обработку WebSockets для передачи данных в реальном времени, что важно для трейдинга. Celery

позволяет эффективно управлять задачами по сбору данных с API, разгружая основную серверную часть.

В качестве хранилища данных используется PostgreSQL. В базе данных хранятся как текущие котировки и объемы торгов, так и исторические данные для ретроспективного анализа и тестирования торговых стратегий. PostgreSQL выбрана за ее способность эффективно обрабатывать большие объемы данных и поддерживать сложные аналитические запросы.

Основные функции PostgreSQL:

– хранение данных о финансовых инструментах. Данные о котировках, объемах, волатильности и других показателях инструментов.
– хранение исторических данных. Исторические котировки и данные о торгах для анализа и построения стратегий.
– управление транзакциями. PostgreSQL поддерживает ACID-транзакции, что гарантирует целостность и безопасность данных [19].

Концептуальная архитектура ПО «Скринер» представлена на рисунке 6.

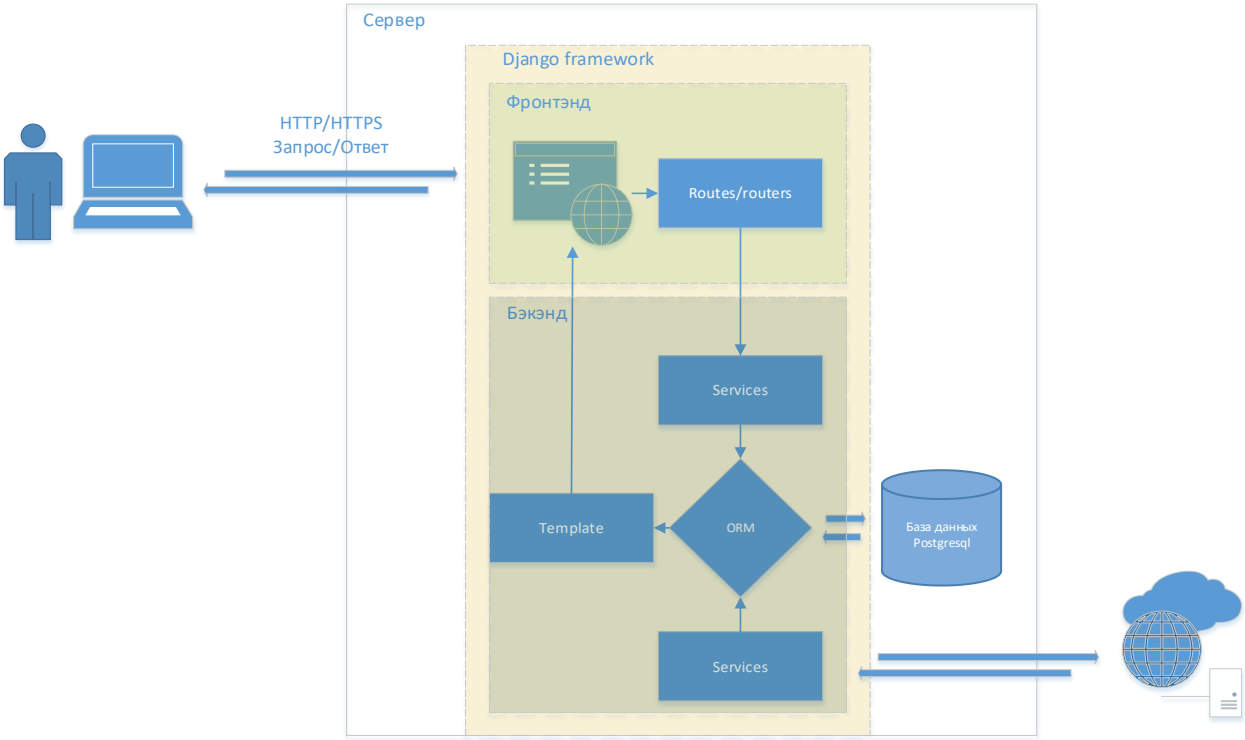


Рисунок 6 - Концептуальная архитектура проекта

Схема демонстрирует, поток данных проходящих через несколько слоев: от запросов пользователя к бизнес-логике на сервере, взаимодействию с базой данных и внешними API до возвращения ответа пользователю. Система является масштабируемой и гибкой благодаря использованию Django, PostgreSQL и интеграции с внешними API.

Преимущества такой реализации:

- масштабируемость. Асинхронная обработка данных позволяет легко масштабировать систему при увеличении количества пользователей или объема загружаемых данных.
- надежность. Механизмы обработки ошибок и повторных попыток загрузки данных гарантируют стабильную работу системы даже в случае сбоев на стороне внешнего API.
- актуальность данных. Постоянная загрузка данных в реальном времени через API Московской биржи позволяет трейдерам принимать решения на основе актуальной информации, что критически важно в условиях быстроменяющегося рынка.

2.3 Логическое моделирование данных АИС

Раздел посвящён проектированию структуры данных автоматизированной информационной системы (АИС). В нем представлено логическое описание системы, которое включает разработку диаграмм вариантов использования (use case), UML-диаграмм структуры данных, определение основных сущностей логической модели и связи.

Целью логического моделирования является формализация требований к системе на уровне данных и процессов, она обеспечивает основу для физического проектирования базы данных и реализации программного обеспечения. Использование UML (Unified Modeling Language) позволяет визуализировать структуру системы и взаимосвязи между её компонентами, что облегчает понимание и дальнейшую разработку [6].

Диаграмма вариантов использования описывает функциональность ИС, которая будет видна пользователям системы. «Каждая функциональность» изображается в виде «прецедентов использования» (use case) или просто прецедентов. На рисунке 7 отображена диаграмма вариантов использования (use case) на диаграме показаны взаимодействия между пользователем (трейдером) и программным обеспечением «Скринер», а также основные функции, которые предоставляет система. Логическая модель данных обеспечивает ясное определение сущностей и их атрибутов, включая отношения между ними, это гарантирует полноту проектирования. Данный подход поможет избежать избыточности и обеспечит качество проектируемой базы данных. Результаты логического моделирования позволят подготовить точные спецификации для последующего этапа физического проектирования, обеспечивая надежную основу для разработки системы.

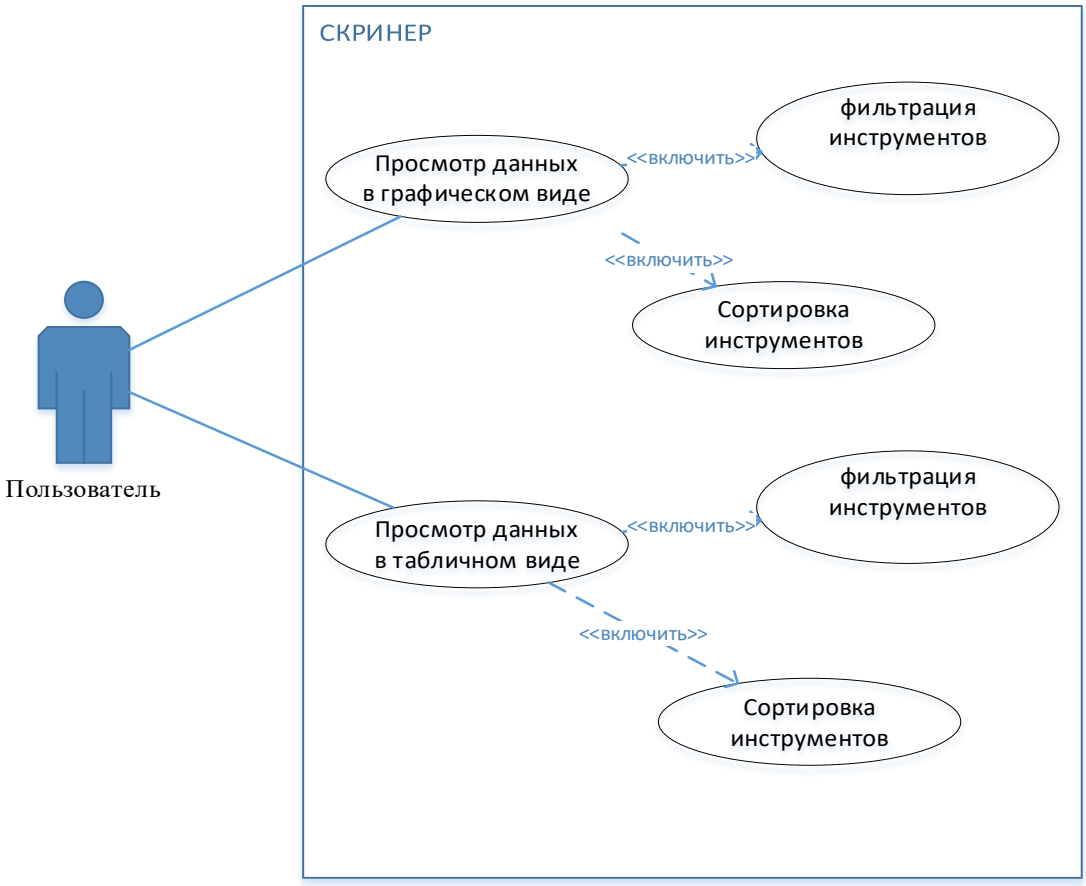


Рисунок 7 –Диаграма вариантов использования

На основании диаграммы представлены участники и варианты использования:

Участники (Actors):

Пользователь (Трейдер): главный участник, который взаимодействует с системой. Трейдер выполняет следующие действия:

– просмотр данных в графическом виде;
– просмотр данных в табличном виде.

Варианты использования (Use Cases):

Просмотр данных в графическом виде. Этот вариант использования предоставляет пользователю возможность визуализировать данные по финансовым инструментам в виде графиков.

Включает:

– функцию фильтрации инструментов, дает возможность выбрать инструменты на основе различных критериев (например, по объемам торгов, волатильности, типам инструментов);
– функцию сортировки инструментов, которая позволяет упорядочить инструменты по заданным параметрам (например, по цене, объему, изменениям).

Просмотр данных в табличном виде. Этот вариант использования позволяет трейдеру просматривать информацию в табличном формате. Это может быть полезно для более детального и структурированного анализа данных.

Включает:

– функцию фильтрации инструментов, дает возможность выбрать инструменты на основе различных критериев (например, по объемам торгов, волатильности, типам инструментов);
– функцию сортировки инструментов, которая позволяет упорядочить инструменты по заданным параметрам (например, по цене, объему, изменениям).

Каждый из сценариев (графический или табличный просмотр данных) включает возможность фильтрации и сортировки инструментов, что позволяет настраивать отображение данных в зависимости от его потребностей. Пользователь может применять фильтры для отбора активов по различным параметрам и сортировать их для удобного анализа.

Диаграмма вариантов использования демонстрирует, что программное обеспечение предоставляет пользователям возможности по визуализации и анализу данных, предлагая, как графический, так и табличный режимы отображения, с дополнительной функциональностью фильтрации и сортировки данных.

Логическая модель данных программного обеспечения представляет собой абстрактное описание компонентов системы и их взаимодействий. В данной системе, основной задачей которой является анализ финансовых инструментов и предоставление актуальных данных о котировках, объемах торгов и других параметрах, ключевые сущности моделируют объекты, связанные с финансовыми инструментами и их характеристиками. На рисунке 8 представлена логическая модель данных.

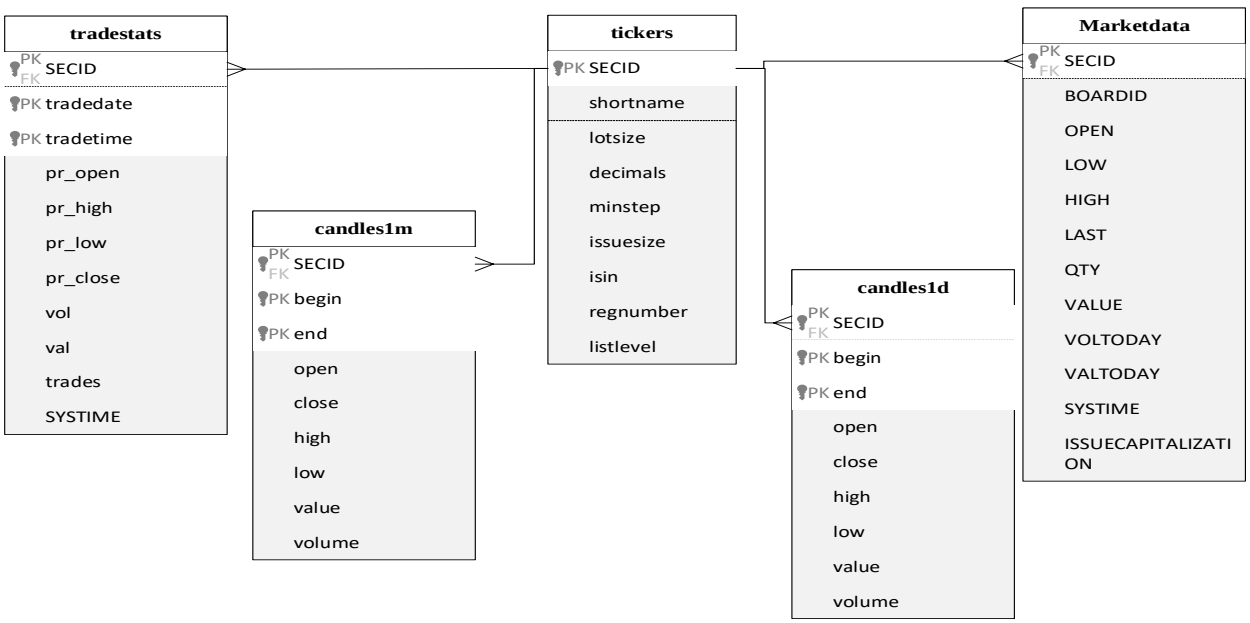


Рисунок 8 - Логическая модель данных

Основные сущности логической модели.

Tickers - сущность, которая представляет собой каждый финансовый инструмент на рынке. Каждый инструмент описан набором параметров, таких как SECID (уникальный идентификатор тикера), shortname (краткое название), lotsize (размер лота), decimals (количество знаков после запятой для цен), minstep (минимальный шаг цены), issuesize (размер выпуска акций), isin (международный код идентификации), regnumber (регистрационный номер), и listlevel (уровень листинга).

Marketdata - сущность содержит актуальные данные о состоянии финансового инструмента на рынке. Она включает в себя такие атрибуты, как SECID (ссылка на тикер), BOARDID (идентификатор рынка), OPEN (цена открытия), HIGH (максимальная цена за день), LOW (минимальная цена), LAST (последняя зарегистрированная цена), QTY (количество торгуемых лотов), VALUE (стоимость торгов), а также текущие объемы и данные по волатильности. Эта сущность обновляется в реальном времени через API Московской биржи и хранит текущие рыночные параметры.

Tradestats - сущность, которая хранит статистику сделок, совершенных по каждому финансовому инструменту. Включает такие атрибуты, как SECID (ссылка на тикер), tradedate (дата сделки), tradetime (время сделки), pr_open (цена открытия), pr_close (цена закрытия), pr_high (максимальная цена за период), pr_low (минимальная цена), а также объемы и количество сделок за определенный промежуток времени. Используется для детализированного анализа сделок, анализа волатильности.

Candles1d и Candles1m - сущности представляют данные для построения графиков по методу японских свечей за разные периоды: дневные (Candles1d) и минутные (Candles1m). Они содержат атрибуты, ticker (ссылка на тикер), begin (время начала периода), end (время окончания периода), а также цены открытия, закрытия, максимальные и минимальные цены за период (open, close, high, low) и объем торгов за этот период (volume).

LogEntry - сущность предназначена для логирования выполняемых в системе операций, она отслеживает события внутри системы. Каждая запись содержит атрибуты, такие как id (уникальный идентификатор), timestamp (время записи), level (уровень логирования: информация, предупреждение, ошибка), message (текст сообщения), ticker (ссылка на тикер) и table (таблица, к которой относится лог).

Взаимосвязи объектами модели. В логической модели важным аспектом является описание взаимосвязей между сущностями:

- связь между Tickers и Marketdata. Связь обозначена как «один ко многим», что означает, что каждый тикер может иметь несколько записей в таблице Marketdata. Это логично, так как каждый финансовый инструмент может одновременно торговаться на нескольких рынках, и для каждого рынка могут существовать свои параметры торговли (например, разные цены предложения и спроса).
- связь между Tickers и Tradestats представляет собой связь «один ко многим», где один тикер связан с несколькими записями в таблице статистики сделок. Это объясняется тем, что для каждого финансового инструмента может быть множество сделок в разные дни и время.
- связь между Tickers и Candle1d/Candle1m: Один тикер связан с несколькими записями в таблицах Candles1d и Candles1m, так как для каждого инструмента может быть множество свечей, отображающих изменение цены за различные временные промежутки (например, минуты или дни).

API Московской биржи предоставляет данные о текущих котировках и объемах торговли, а также исторических данных. Эти данные сохраняются в таблице «Marketdata», «Candle1d», «Candle1m» и «Tradestats», обновляются в реальном времени или исходя из временного периода от 1 до 5 минут, что позволяет трейдерам оперативно получать информацию о состоянии рынка.

Исторические данные и статистика сделок хранятся в таблицах «Tradestats», «Candle1d» и «Candle1m».

Логирование и мониторинг системы. Логи, которые записываются в таблицу «LogEntry», помогают администраторам системы отслеживать события, происходящие внутри системы, и своевременно реагировать на возможные ошибки или сбои. Логи могут включать информацию о выполнении запросов, ошибках при работе с API или изменениях в данных о тикерах.

2.4 Требования к аппаратно-программному обеспечению АИС

Разработка требований к аппаратно-программному обеспечению автоматизированной информационной системы (АИС) необходима для обеспечения эффективной и надежной работы системы. Для ПО «Скринер», которая обрабатывает и анализирует рыночные данные в реальном времени, требования можно разделить на несколько категорий: аппаратные требования, программные требования, и сетевые требования.

Аппаратные требования.

Для нормального функционирования ПО «Скринер» необходимо обеспечить минимальные и рекомендованные характеристики серверов, которые будут обслуживать систему.

Сервер для обработки данных и хранения базы данных:

- процессор (CPU): 4 ядра, 2.5 GHz
- оперативная память (RAM): 16 GB
- место на жестком диске (HDD/SSD): 1 TB SSD с возможностью увеличения до 3-5 TB (рекомендуется использовать SSD для обеспечения высокой скорости записи и чтения данных).
- пропускная способность сети: 10 Gbps для увеличения скорости передачи данных.

Сервер для веб-приложения (frontend и backend):

– процессор (CPU): 6 ядер, 3.0 GHz и выше
– оперативная память (RAM): 32 GB и выше
– место на жестком диске (HDD/SSD): 256 GB SSD
– пропускная способность сети: 10 Gbps для увеличения скорости передачи данных.

Программные требования.

Для работы ПО «Скринер» необходимо обеспечить установку программного обеспечения, которое будет поддерживать как серверную, так и клиентскую части системы.

Серверная часть (frontend, backend) включает в себя операционную систему Linux (Ubuntu Server, CentOS, РЕД ОС, Astra Linux) — предпочтительно для серверов. Язык программирования Python 3.9 и выше, наличие фреймворка Django 3.0 и выше (для работы с фронтендом и бэкендом) с установленными библиотеками SQLAlchemy - для работы с базой данных PostgreSQL, Celery - для фоновых задач и обработки данных в реальном времени, Daphne - для поддержки WebSockets, база данных (СУБД): PostgreSQL 12 и выше.

Сетевые требования.

Для обеспечения быстрой и надежной работы системы необходимо организовать соответствующую сетевую инфраструктуру, поддерживающую низкую задержку и высокую пропускную способность.

Пропускная способность сети серверной части должна составлять минимум 1 Gbps соединение между серверами и внешними источниками данных (API Московской биржи), рекомендуется 10 Gbps.

Клиентская часть минимум 100 Mbps соединение для рабочих станций трейдеров, рекомендуется 1 Gbps.

Системы резервного копирования данных должна обеспечивать регулярное резервное копирование базы данных (PostgreSQL) и файлов конфигурации.

Эти требования к аппаратно-программному обеспечению обеспечат надежную и быструю работу системы в условиях реального времени.

Выводы по главе 2.

Логическое проектирование ПО «Скринер» включает в себя выбор технологий моделирования, таких как ER-моделирование и UML, для визуализации и детального описания структуры базы данных и компонентов системы. В ходе проектирования были описаны ключевые сущности и их связи, такие как тикеры, рыночные данные, свечные графики и статистика сделок. Эти сущности взаимосвязаны через таблицы базы данных, что позволяет эффективно хранить и анализировать данные.

Представлена многослойная архитектура системы, которая включает клиентскую и серверную части, что обеспечивает высокую производительность, надёжность и масштабируемость системы. Использование технологий Python, фрэймворка Django [13], PostgreSQL и протокола WebSockets гарантирует работу с данными в режиме реального времени, что особенно важно для трейдеров, работающих с высокочастотной внутридневной торговлей.

В процессе проектирования были сформулированы требования к аппаратно-программному обеспечению, которые обеспечат надёжную работу системы в условиях высокой нагрузки и позволят трейдерам оперативно анализировать данные и принимать решения.

Таким образом, логическое проектирование АИС полностью отражает потребности компании и направлено на создание эффективного инструмента для автоматизации технического анализа финансовых инструментов.

Глава 3 Физическое проектирование АИС

3.1 Выбор архитектуры АИС

Для разработки ПО «Скринер», предназначенной для анализа данных, выбрана архитектура, основанная на фреймворке Django, которая объединяет элементы клиент-серверной и микросервисной архитектур. Это обеспечивает высокую производительность, гибкость и надежность системы, а также возможность легкой интеграции с внешними источниками данных, такими как API Московской биржи представленными в открытом источнике [11]. На рисунке 9 представлена диаграма развертывания программонго обеспечения.

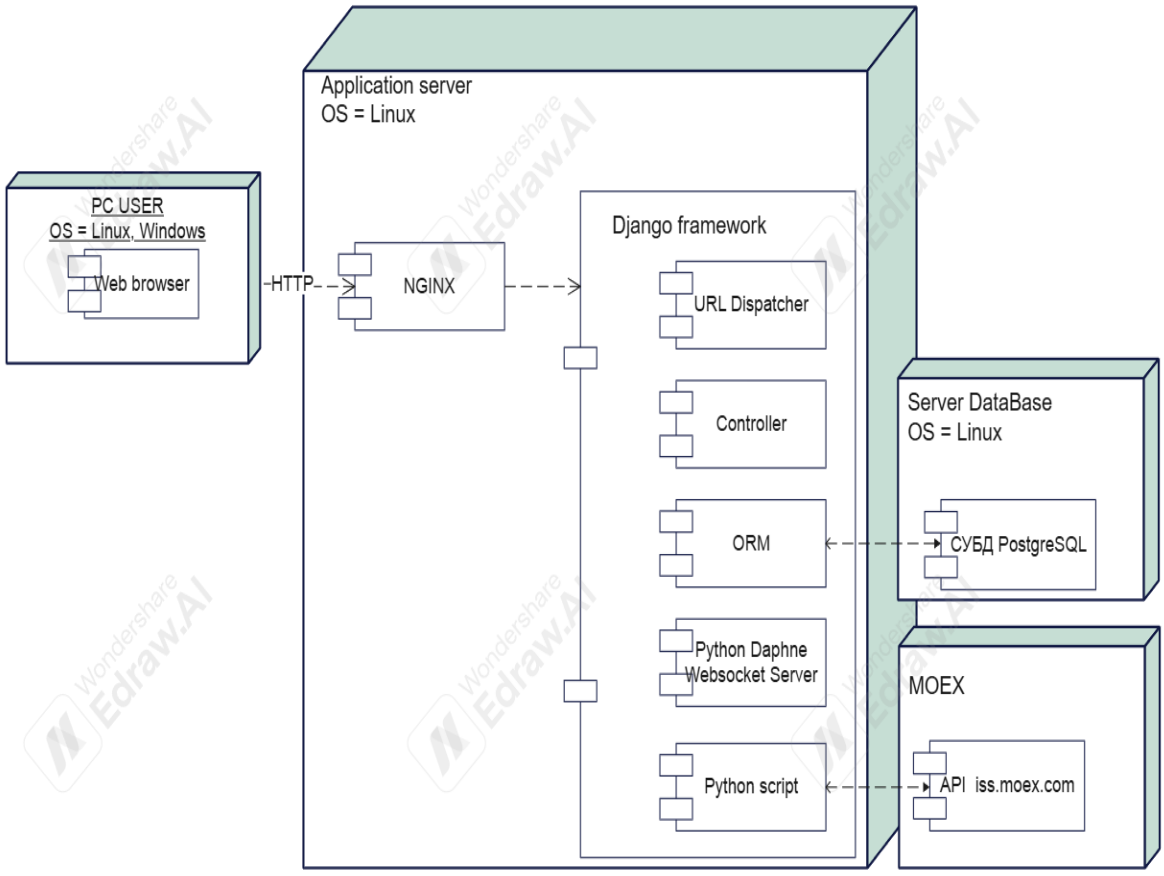


Рисунок 9 – Диаграма развертывания ПО «Скринер»

На представленной диаграме представлены ключевые компоненты системы:

- клиентская часть (Frontend). Пользователь взаимодействует с системой через веб-браузер, отправляя HTTP/HTTPS запросы на веб-сервер NGINX.
- Веб-сервер NGINX. NGINX выступает в качестве прокси-сервера для управления входящими HTTP/HTTPS запросами, направляя их на Django-приложение. Он обеспечивает высокую производительность и надежность при работе с большим количеством пользователей одновременно.
- Python Django Framework. Основная часть бизнес-логики системы реализована на Django, популярном фреймворке для веб-разработки. Он отвечает за обработку запросов, маршрутизацию, взаимодействие с базой данных и генерацию ответов в виде HTML-страниц или данных в формате JSON. Django поддерживает как синхронные, так и асинхронные задачи.
- Python Daphne (WebSocket Server). Для передачи данных в реальном времени, таких как котировки и графики, используется сервер Daphne для работы с WebSockets. Daphne позволяет поддерживать постоянное соединение с клиентом и передавать данные о рынке акций без необходимости повторных запросов. Это ключевая часть архитектуры, которая поддерживает связь в реальном времени.
- Python Scripts и ORM (Object-Relational Mapping). В системе используются Python-скрипты для обработки данных, полученных через API Московской биржи, и для выполнения фоновых задач, таких как обновление данных в базе данных. Django ORM обеспечивает удобное взаимодействие с базой данных PostgreSQL, позволяя работать с данными на уровне объектов, что упрощает процесс выполнения SQL-запросов и управление транзакциями.

– база данных PostgreSQL, для хранения данных о котировках, объемах торгов и исторической информации используется реляционная база данных PostgreSQL. Она надежна и масштабируема, что позволяет обрабатывать большие объемы данных, необходимые для анализа рынка и выполнения торговых стратегий.

Данная архитектура представляет следующие преимущества:

– веб-сервер NGINX и Django обеспечивают высокую производительность и возможность масштабирования системы. Архитектура поддерживает одновременное обслуживание большого количества пользователей без задержек.
– использование WebSockets через Daphne позволяет мгновенно передавать обновления данных с минимальной задержкой. Это особенно важно для работы с котировками в реальном времени.
– Django предоставляет мощную структуру для разработки, позволяя легко добавлять новые модули и функции без значительных изменений в коде. ORM упрощает взаимодействие с базой данных, а интеграция с внешними API позволяет получать данные с биржи без дополнительных трудностей.
– вся нагрузка на серверы распределяется между NGINX (обработка HTTP/HTTPS-запросов), Daphne (обслуживание WebSockets) и Django (управление бизнес-логикой), что снижает вероятность сбоев и улучшает отклик системы.

Выбор архитектуры с использованием Django, NGINX и WebSockets через Daphne позволил создать надежную и гибкую систему для анализа данных в реальном времени. Эта архитектура обеспечивает высокую производительность, стабильность и возможность масштабирования, что делает её идеальной для трейдинга на фондовом рынке, где скорость и точность данных играют ключевую роль.

3.2 Выбор технологии разработки программного обеспечения

При создании программного обеспечения для технического анализа финансовых инструментов были определены основные критерии выбора технологий:

- производительность и масштабируемость, обработка больших объемов данных в реальном времени;
- гибкость и расширяемость, возможность быстрого добавления новых функций;
- кроссплатформенность и открытый код, широкая поддержка сообщества.

Язык Python выбран как основной язык программирования из-за простоты и читабельности кода, что ускоряет разработку и упрощает поддержку. Так же у него богатая экосистема библиотек для работы с данными и веб-разработки. Python поддерживает возможности асинхронного программирования, важного для реального времени.

Python имеет богатую экосистему библиотек и фрэймворков, в том числе и фрэймворк Django. Django выбран для реализации ПО «Скринера» для фронтенда и бэкенда потому, что обеспечивает полный стэк разработки предоставляет все необходимые компоненты, у него высокая производительность, он оптимизирован для работы с большими объемами данных.

Рассмотренные альтернативы:

- Flask вместо Django: менее тяжеловесен, но Django предпочтительнее для крупных проектов.
- React или Angular для фронтенда: хотя они обеспечивают высокую интерактивность, Django выбран для унификации стека и упрощения разработки. Так же Django уже используется в архитектуре ИС компании.

Вывод: выбранный стек технологий оптимально соответствует требованиям проекта, обеспечивая производительность, надежность и масштабируемость, а также упрощая процесс разработки и дальнейшего сопровождения системы.

3.3 Выбор СУБД АИС

При разработке программного обеспечения был проведен анализ различных систем управления базами данных (СУБД) (таблица 3) с целью выбора оптимальной для данного проекта. В процессе анализа рассматривались ключевые СУБД: PostgreSQL, MySQL, Microsoft SQL Server и Oracle. Основными критериями сравнения были надежность, производительность, масштабируемость, поддержка сложных запросов и транзакций, а также стоимость эксплуатации.

Таблица 3 - Результаты анализа рассматриваемых СУБД

Параметр	PostgreSQL	MySQL	Microsoft SQL Server	Oracle
Производительность	Высокая, особенно на больших объемах данных благодаря мощной системе индексов и поддержке сложных запросов	Хорошая для небольших и средних проектов, но уступает при сложных аналитических задачах	Отличная производительность, оптимизирован для работы с большими объемами данных	Высокая производительность, особенно для корпоративных решений
Масштабируемость	Хорошая поддержка горизонтального масштабирования и кластеризации	Масштабируемость ограничена, сложнее масштабировать горизонтально	Отличная для вертикального масштабирования, поддержка кластеризации	Высокая масштабируемость, поддержка крупных кластеров
Надежность и отказоустойчивость	Высокая, поддержка ACID, мультиверсионность, активное сообщество	Высокая, но мультиверсионность слабее, особенно при высоких нагрузках	Поддержка ACID, встроенные средства резервирования и восстановления	Надежная, поддержка всех аспектов ACID, встроенные инструменты управления отказами

Продолжение таблицы 3

Параметр	PostgreSQL	MySQL	Microsoft SQL Server	Oracle
Поддержка сложных запросов	Поддержка сложных SQL-запросов, индексов, транзакций и аналитических функций	Хорошо работает с простыми запросами, но менее эффективна с более сложными	Отличная поддержка сложных запросов и транзакций	Поддержка сложных запросов, аналитических функций, транзакций
Поддержка временных рядов	Поддерживает временные ряды и расширения для работы с ними	Ограниченная поддержка временных рядов, требует внешних решений	Поддерживает временные ряды через дополнительные функции	Отличная поддержка временных рядов и аналитики на их основе
Поддержка NoSQL функций	Поддержка JSON и JSONB для работы с неструктурированными данными	Поддержка JSON, но менее эффективна	В основном ориентирована на SQL, но есть возможность работы с неструктурированными данными	Oracle поддерживает работу с JSON и XML
Лицензирование и стоимость	Бесплатная, open-source лицензия	Бесплатная, open-source лицензия (для Community Edition)	Платная, модель лицензирования зависит от объема использования	Платная, высокая стоимость лицензий для корпоративных решений
Совместимость с другими системами	Легко интегрируется с внешними системами через API, поддерживает множество языков программирования	Легкая интеграция, поддержка большого числа языков программирования	Хорошая интеграция с продуктами Microsoft, сложнее интегрировать с другими системами	Отличная интеграция с корпоративными системами, но сложнее с open-source продуктами
Сообщество и поддержка	Большое сообщество, активная поддержка и регулярные обновления	Большое сообщество, активная поддержка и большое количество документации	Платная поддержка от Microsoft, отличная документация	Платная поддержка Oracle, профессиональная поддержка и документация
Безопасность	Поддержка сложных ролей, аутентификации, шифрования данных	Хорошая, но меньше гибкости в настройке безопасности по сравнению с PostgreSQL	Высокий уровень безопасности, интеграция с Active Directory	Один из лидеров по безопасности, поддержка шифрования и контроля доступа

На основе проведенного анализа PostgreSQL была выбрана как наиболее подходящая СУБД для разработки АИС. СУБД обладает достаточной мощностью для обработки больших объемов данных, поддерживает сложные аналитические запросы и транзакции, а также легко масштабируется, что важно для системы, работающей с большим объемом данных предоставляемых Московской и Санкт-петербуржской биржами. Немаловажное значение имеет то, что версия СУБД Postgres Pro (созданная на базе PostgreSQL и по сути имеющая такую же архитектуру и возможности) входит в Реестр российского ПО и сертифицирована ФСТЭК для обработки конфиденциальной информации и персональных данных [14].

3.4 Разработка физической модели данных АИС

Физическая модель данных представляет собой описание структуры базы данных, которая хранит информацию, необходимую для функционирования системы. В данной модели определены таблицы, атрибуты, их типы данных, а также ключевые связи между таблицами. Модель была разработана с учетом особенностей системы «Скринер», так чтобы обеспечить более простую загрузку данных относящихся к торговле финансовыми инструментами, возможность хранения и обработки большого объема рыночных данных.

В представленной физической модели для ускорения выборок данных реализованы индексы. Так же были учтены требования к оптимизации хранения данных, что позволило минимизировать затраты на дисковое пространство без потери производительности системы. Представленная структура может гарантировать необходимую масштабируемость базы данных, что особенно важно для обработки данных с высокой скоростью, характерной для операций трейдинга. На рисунке 10 представлена физическая модель данных.

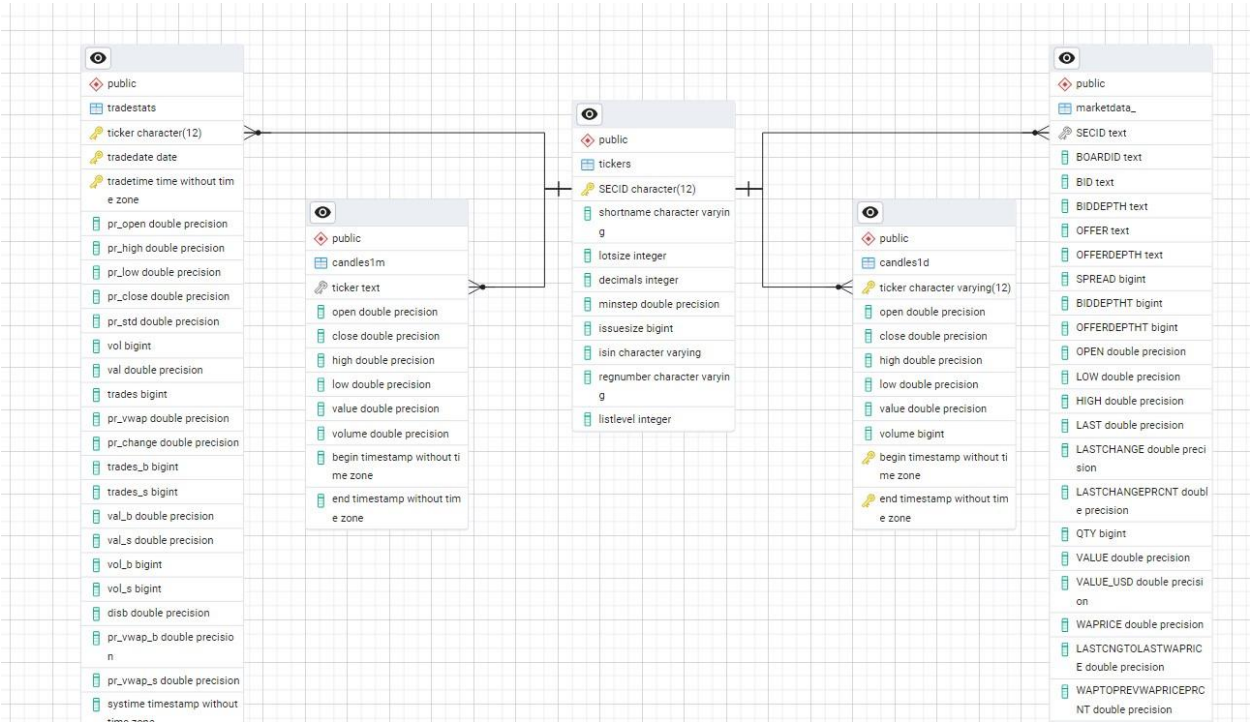


Рисунок 10 - Физическая модель данных

При разработке физической модели данных используя СУБД PostgreSQL были применены рекомендации по оптимизации запросов, в частности методы оптимизации индексов, нормализации таблиц и использования эффективных типов данных. Применение рекомендаций позволяет увеличить производительность системы при обработке больших объемов данных, ускорить выполнение сложных аналитических запросов. Созданные таблицы и их описание представлены в данном разделе в таблицах 4-8.

Таблица 4 - Tradestats, показатели, вычисляемые на основе сделок, совершенных на бирже: цены, объёмы, соотношение покупок и продаж за каждые 5 минут

Поле	Тип данных	Описание
ticker	character(12)	Уникальный идентификатор тикера
tradedate	date	Дата сделки

Продолжение таблицы 4

Поле	Тип данных	Описание
tradetime	time	Время сделки
pr_open	double precision	Цена открытия
pr_high	double precision	Максимальная цена за период
pr_low	double precision	Минимальная цена за период
pr_close	double precision	Цена закрытия
pr_std	double precision	Стандартное отклонение
vol	bigint	Объем торгов
val	double precision	Стоимость торгов
trades	bigint	Количество сделок
systime	timestamp	Время записи данных
sec_pr_open	bigint	Открытие второй сессии
sec_pr_high	bigint	Высшая цена второй сессии
sec_pr_low	bigint	Низшая цена второй сессии

Таблица 5 - Candles1d, свечи по тикеру за период 1 день

Field	Type	Description
ticker	character(12)	Уникальный идентификатор тикера
open	double precision	Цена открытия
close	double precision	Цена закрытия
high	double precision	Максимальная цена за день
low	double precision	Минимальная цена за день
value	double precision	Объем торгов
volume	bigint	Количество сделок
begin	timestamp	Время начала свечи
end	timestamp	Время окончания свечи

Таблица 6 - Candles1m, свечи по тикеру за период 1 минута

Field	Type	Description
ticker	text	Уникальный идентификатор тикера
open	double precision	Цена открытия
close	double precision	Цена закрытия
high	double precision	Максимальная цена за минуту
low	double precision	Минимальная цена за минуту
value	double precision	Объем торгов
volume	double precision	Количество сделок
begin	timestamp	Время начала свечи
end	timestamp	Время окончания свечи

Таблица 7- Tickers, список доступных инструментов рынка

Field	Type	Description
SECID	character varying (12)	Уникальный идентификатор тикера
shortname	character varying	Краткое название тикера
lotsize	integer	Размер лота
decimals	integer	Количество знаков после запятой
minstep	double precision	Минимальный шаг цены
issuesize	bigint	Объем выпуска акций
isin	character varying	Международный код идентификации
regnumber	character varying	Регистрационный номер
listlevel	integer	Уровень листинга

Таблица 8 - Marketdata, статистическая информация о инструментах рынка

Field	Type	Description
SECID	text	Уникальный идентификатор тикера
BOARDID	text	Идентификатор рынка
BID	text	Цена спроса
OFFER	text	Цена предложения
OPEN	double precision	Цена открытия
HIGH	double precision	Максимальная цена
LOW	double precision	Минимальная цена
LAST	double precision	Последняя зарегистрированная цена
QTY	bigint	Количество торгуемых лотов
VALUE	double precision	Стоимость торгов
VOLTODAY	bigint	Объем торгов за день
VALTODAY	bigint	Стоимость торгов за день
SYSTIME	text	Время записи данных

В данной модели реализованы связи между основными таблицами с помощью внешних ключей:

Таблицы tradestats, candles1d, candles1m и marketdata содержат внешние ключи, которые ссылаются на таблицу tickers по полю SECID. Это позволяет поддерживать целостность данных, а также эффективно управлять информацией о финансовых инструментах.

Физическая модель данных программного обеспечения «Скринер» обеспечивает эффективное хранение, управление и обработку больших объемов данных о сделках, котировках и инструментах биржи. Выбор PostgreSQL в качестве СУБД, а также нормализованная структура базы данных, позволяют ПО «Скринер» функционировать с высокой производительностью и поддерживать масштабируемость для увеличения

объемов данных в будущем. Скрипт создания структуры данных представлен в приложении А.

3.5 Разработка программного обеспечения АИС

Разработанное программное обеспечение можно разделить на 3 основных блока, это: загрузчик, обработчик и визуализация. Ниже представлено описание, состав и функциональность каждого из блоков.

3.5.1 Загрузчик

Для того чтобы программное обеспечение было более гибким, было принято решение по предобработке данных получаемых с биржи, загрузке не напрямую в пользовательский интерфейс, а через базу данных. Это решение позволяет реализовывать разные загрузчики исходя из точки представления информации, например, загрузчик, который будет брать данные не из API московской биржи, и у брокера такого как «Т-Банк Инвестиции» (борокер достаточно подробно представил описание API и постоянно развивает данное направление [10]) или БрокерКредитСервис (БКС) которые так же предоставляют доступ к API, а также могут предоставлять дополнительную информацию помимо основной торговой информации предоставляемой биржей.

В текущей версии, загрузчик использует библиотеку moexalgo [12]. Загрузчик данных состоит из скриптов. Во всех скриптах присутствует функция log_to_db() для записи сообщений о ходе выполнения загрузки, данные хряняться в таблице логов.

а) load_candles1d.py - этот файл отвечает за загрузку дневных свечей для тикеров. Основная функция process_ticker (secid, start_date, end_date, session):

1) функция принимает тикер (secid), начальную и конечную даты для получения данных, а также сессию;

2) получает данные свечей с помощью библиотеки moexalgo и обрабатывает их в виде DataFrame;
3) проверяет наличие существующих данных в таблице БД отфильтровывает только новые;
4) записывает их в таблицу candles1d.

б) load_candles1m.py – это аналог скрипта load_candles1d.py только отвечает за загрузку минутных свечей для тикеров. Записывает их в таблицу candles1m.

в) load_marketdata.py - этот файл отвечает за загрузку рыночных данных. Основная функция load_market_data():
1) функция получает рыночные данные, такие как цены спроса и предложения, объемы, для каждого тикера в текущий момент времени;
2) полученные данные сохраняются в таблицу marketdata в базе данных.

г) load_stats.py – этот файл занимается обработкой статистических данных. Основная функция load_stats_data():
1) функция отвечает за получение и обработку статистических данных по каждому тикеру за каждые 5 минут;
2) данные, такие как объем торгов и изменения цены, собираются и записываются в базу данных;
3) статистика сохраняется в таблице, связанной с тикерами Tradestats.

д) load_tickers.py - этот файл предназначен для загрузки и обновления списка тикеров. Основная функция: load_tickers():
1) функция загружает список всех доступных тикеров с биржи;
2) полученные тикеры обновляются в таблице tickers в базе данных;
3) функция проверяет актуальность данных и обновляет их в случае необходимости.

Для работы с СУБД Postgresql в среде Python Django используется библиотека SQLAlchemy. Для описания моделей взаимодействия с таблицами базы данных используется скрипт с описанием sqlalchemy_models.py. В скрипте Определены поля для хранения всех необходимых данных, таких как тикер, время начала и окончания, цена, объем, и другие.

Используемые модели: Candle1, Candle1d: модели для минутных и дневных свечей соответственно, MarketData: модель для рыночных данных, LogEntry: модель для логирования действий.

Эти скрипты обеспечивают загрузку данных о свечах, рыночных данных и статистике для тикеров и сохраняют их в базу данных, управляемую через SQLAlchemy.

3.5.2 Обработчик данных

Обработчик данных так же представлен несколькими скриптами где каждый скрипт подготавливает свой набор данных. Это обеспечивает гибкость, т.к. позволяет подготовить и внедрить новый набор данных или параметров, не меняя существующие скрипты или функции за исключением группирующего скрипта.

Скрипт get_candles2.py -скрипт который загружает данные из таблиц базы данных candles1m и marketdata, а затем вычисляет изменения цены для различных временных интервалов. Алгоритм скрипта можно описать следующим образом:

а) загрузка данных из базы:

1) функция get_candle_data_n(date) загружает данные свечей из таблицы candles1m, а также текущие рыночные данные из таблицы marketdata;
2) SQL-запрос соединяет таблицы по тикеру, используя поле ticker из таблицы candles1m и поле SECID из таблицы marketdata;
3) если дата не передана, используется текущая дата;
4) результаты SQL-запроса преобразуются в DataFrame с помощью pandas;

5) при возникновении ошибки логируется сообщение об ошибке, и возвращается пустой DataFrame.

б) поиск ближайшей записи к указанной дате:

1) функция find_closest_record(group, target_time) ищет запись в наборе данных свечей, которая имеет время начала (поле begin), наиболее близкое к указанной дате;
2) если точного совпадения нет, выбирается запись с временем, меньшим или равным target_time.

в) расчет изменений цен и объемов:

1) функция calculate_changes(df) рассчитывает изменения цены и объема для каждого тикера за указанные временные интервалы;
2) Сначала данные сортируются по времени (begin);
3) определяется максимальная дата для каждого тикера (временная метка последней свечи);
4) для каждого тикера рассчитываются изменения цены за следующие интервалы: 5 минут, 10 минут, 15 минут, 30 минут, 1 час.

г) формирование и вывод данных:

1) результаты для каждого тикера собираются в общий DataFrame, который содержит данные об изменении цены и объема за разные временные интервалы;
2) все значения NaN заменяются на None, чтобы избежать проблем с отсутствующими данными;
3) итоговый DataFrame возвращается для дальнейшего использования.

Скрипт get_candles3.py - этот код реализует функцию для расчета изменений объема торгов для определенной даты на основе данных из таблицы tradestats в базе данных. Давайте разберем алгоритм более подробно:

Функция calculate_volume_changes_for_date(date: str = None) - рассчитывает изменения объема торгов для каждого тикера за разные периоды времени (5, 10, 15, 30, 60 минут). Последовательность шагов данной функции:

а) получение данных из базы данных. Выполняется SQL-запрос, который загружает данные из таблицы tradestats для указанной даты:

1) поля, которые загружаются: ticker, tradedate, tradetime, vol (объем), val, и другие;
2) результат запроса преобразуется в DataFrame.

б) предобработка данных:

1) удаляются пробелы из значений столбца ticker;
2) создается новое поле datetime, которое объединяет значения даты (tradedate) и времени (tradetime) в одну временную метку для сортировки;
3) данные сортируются по тикеру и времени (datetime).

в) выполняется функция calculate_volume_changes(group, periods) - расчет изменений объема:

1) применяется к каждому тикеру;
2) для каждого тикера выполняется расчет изменения объема за указанные временные интервалы (5, 10, 15, 30, 60 минут);
3) рассчитывается суммарный объем за каждый период с помощью метода rolling(), который применяет скользящее окно по данным;
4) рассчитывается разница между текущим объемом и объемом за предыдущий такой же период (diff());
5) процентное изменение объема рассчитывается как отношение изменения объема к суммарному объему за предыдущий период.

г) фильтрация данных по максимальной дате. После расчетов данные фильтруются так, чтобы оставить только записи с максимальной датой и временем для каждого тикера.

д) возвращается DataFrame, содержащий тикеры, временные метки и рассчитанные изменения объема за различные временные интервалы (5, 10, 15, 30, 60 минут).

В коде предусмотрено логирование событий. В случае ошибки, при выполнении SQL-запроса или при обработке данных, функция логирует ошибку и возвращает пустой DataFrame.

Скрипт get_candles_data.py – этот код представляет собой модуль для получения и обработки данных свечей (candles) и рыночных данных. Он объединяет данные свечей и рыночные данные в единый DataFrame, а затем группирует их по тикерам, выбирая последние 60 записей для каждого тикера. Ниже рассмотрим алгоритм работы скрипта:

Основная функция get_candles_data (date: str = None), функция загружает данные свечей (candles) и рыночные данные, объединяет их, преобразует дату в формат Unix timestamp, а затем возвращает данные в виде DataFrame с двумя колонками: ticker и candles_data, где содержится набор минутных свечей для каждого тикера, ограниченный до последних 60 записей. Последовательность шагов данной функции:

а) загрузка данных свечей (candles) из базы данных:
   1) выполняется SQL-запрос для выборки данных минутных свечей из таблицы candles1m, фильтруемых по указанной дате;
   2) поля, которые загружаются: тикер, цены (открытие, закрытие, максимум, минимум), объем, время начала и конца свечи;
   3) данные сортируются по тикеру и времени начала свечи.

б) загрузка рыночных данных из таблицы marketdata:
   1) выполняется SQL-запрос для получения текущих рыночных данных, таких как цена закрытия (поле LAST) для каждого тикера;
   2) время округляется до минутного интервала, так как данные должны быть приведены к минутному формату;
   3) результат сохраняется в DataFrame.

в) объединение данных свечей и рыночных данных:

1) два DataFrame (candles_df и marketdata_df) объединяются в один общий DataFrame с помощью pd.concat();
2) для каждого тикера будут объединены данные из обеих таблиц.

г) поле begin (время начала свечи) преобразуется в формат Unix timestamp для использования в дальнейших расчетах и работе с графиками.

д) данные группируются по тикеру, и для каждого тикера выбираются последние 60 записей (минутных свечей).

Результат преобразуется в список записей, где каждая свеча представлена в виде словаря с полями: время, цены (открытие, максимум, минимум, закрытие) и объем.

е) возвращается DataFrame с двумя колонками: ticker (тикер инструмента) и candles_data (список свечей для каждого тикера).

Скрипт get_tickers.py – этот код представляет собой модуль для загрузки списка тикеров из таблицы tickers в базе данных. Ниже описан алгоритм работы программы:

Основная функция get_tickers() - эта функция загружает список тикеров из базы данных. Последовательность шагов данной функции.

Получение данных из базы данных:

- выполняется SQL-запрос, который загружает данные из таблицы tickers для указанной даты;
- результат запроса преобразуется в DataFrame;
- пробелы в значениях столбца ticker удаляются с помощью метода str.strip() для удаления лишних пробелов;
- если возникает ошибка во время выполнения запроса или работы с данными, она логируется с указанием причины ошибки, и функция возвращает пустой DataFrame;
- функция возвращает DataFrame с данными о тикерах.

Эта функция используется для загрузки списка всех тикеров с их идентификаторами (ticker), краткими именами (shortname) и уровнем листинга

(listlevel). Результаты могут использоваться в других частях приложения для анализа или отображения данных о компаниях.

Скрипт get_candles.py - этот код объединяет несколько функций для получения данных о свечах, изменения объема и списка тикеров, затем объединяет все эти данные в единый DataFrame.

Импорт необходимых модулей:

- calculate_volume_changes_for_date — рассчитывает изменения объема торгов;
- calculate_changes и get_candle_data_n — рассчитывает изменения цены;
- get_candles_data — для получения свечных данных;
- get_tickers — для получения списка тикеров.

Основная функция get_candles(date: str = None) - эта функция выполняет несколько шагов для получения и объединения данных о свечах, изменениях цены, объема и данных о тикерах на указанную дату. Сначала объединяются данные о ценах (из функции calculate_changes) и изменения объемов (из функции calculate_volume_changes_for_date) по полю ticker.

Далее к результату добавляются данные свечей (из функции get_candles_data), снова по полю ticker.

В конце добавляются данные о тикерах (из функции get_tickers), также по полю ticker.

Функция возвращает объединенный DataFrame с полной информацией о тикерах, изменениях цены, объема и свечных данных.

3.5.3 Визуализация данных

changes.html – HTML-шаблон который содержит набор функций на JavaScript, которые в свою очередь позволяют работать с данными в реальном времени через WebSocket-соединение и отображать их в двух представлениях: в виде таблицы и в виде графиков. Ниже представлнены основные функции и их описание:

Функция connectWebSocket(), устанавливает WebSocket-соединение для получения данных в реальном времени с сервера.

- при успешном соединении вызывает событие onopen, которое отображает сообщение о подключении;
- когда поступают данные (onmessage), они обрабатываются, данные сохраняются в переменную allData, и вызывается функция applyFilters() для применения фильтров к данным;
- если соединение закрывается (onclose), функция пытается переподключиться каждые 5 секунд.

Функция applyFilters(), эта функция применяет фильтры, которые пользователь устанавливает в модальном окне, такие как диапазон цен и изменение цены и объема за последние 5 минут.

- данные фильтруются по условиям, проверяющим, попадают ли записи в выбранные диапазоны;
- после фильтрации данных вызывается функция applySorting() для их сортировки.

Функция applySorting(), функция сортировки данных, основанная на выбранном ключе сортировки (currentSortKey), например, изменение цены за 5 минут, 10 минут и так далее.

Если выбранное представление данных — таблица, вызывается renderTable(), если графическое — renderGridPage().

Функция renderTable(), отображает данные в табличной форме.

- создает динамические строки таблицы на основе отфильтрованных данных, добавляя данные о тикерах, цене, объеме, изменениях за определенные временные интервалы;
- реализует подсветку изменений: зеленая подсветка — рост значений, красная — снижение. Для этого используется функция highlightChange(), которая сравнивает текущее значение с предыдущим и подсвечивает разницу.

Функция renderGridPage() и renderGrid(data).

– renderGridPage(), отображает текущую страницу данных в графическом виде, используя пагинацию. На одной странице отображается до 12 элементов;
– renderGrid(data), создает элементы сетки для графиков, отображает информацию о тикерах, их изменениях цены и объема, и строит свечные графики с помощью библиотеки Lightweight Charts. Для каждого тикера создается свечной график на основе последних 60 записей с ценами открытия, закрытия, максимальными и минимальными значениями.

Пагинация createPagination() и updatePagination().

– createPagination(): создает кнопки пагинации, основываясь на общем количестве элементов и количестве элементов на странице (по умолчанию 12).
– updatePagination(): обновляет активную страницу, подсвечивая текущую страницу в пагинации.

Функция highlightChange(element, newValue, previousValue), подсвечивает ячейки таблицы или элементы с изменениями, выделяя изменения в значениях. Используется для визуального выделения увеличений (зеленая подсветка) и уменьшений (красная подсветка) значений цены или объема.

Функция initApp() - инициализирует приложение. Запускает подключение к WebSocket через connectWebSocket().

Листинг всех скриптов ПО «Скринер» представлен в приложении Б.

3.6 Описание функциональности АИС

В разделе выше описаны основные блоки АИС включающие скрипты и функции в их составе, в данном разделе представлена функциональность ключевых функций системы.

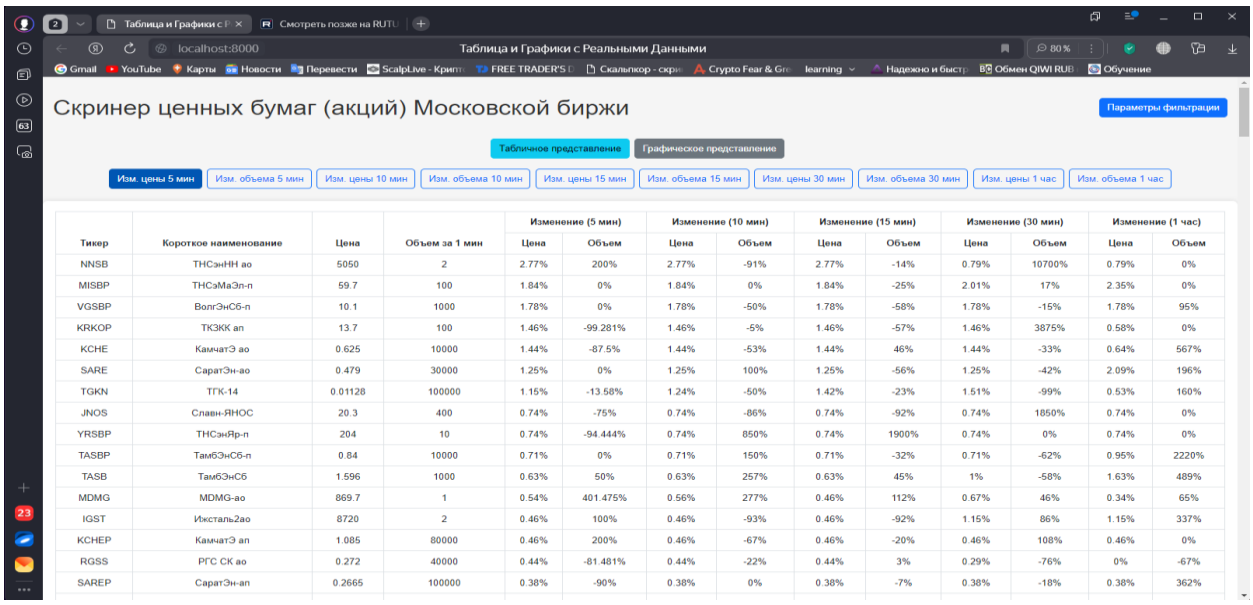
Блок загрузки данных, основные компоненты и описание:

- логирование событий (log_to_db), функция логирования записывает события в базу данных, таблицу LogEntry. Это помогает отслеживать процесс загрузки данных, соединения с MOEX API и любые возникшие ошибки;
- подключение к базе данных, используется SQLAlchemy для взаимодействия с базой данных. Настроено подключение через объект engine, а сессии создаются с помощью Session;
- загрузка данных, значений, параметров и т.д. через API MOEX с использованием библиотеки moexalgo;
- блок обработки данных, основные компоненты и описание;
- подключение к базе данных, используется SQLAlchemy для взаимодействия с базой данных. Настроено подключение через объект engine, а сессии создаются с помощью Session;
- получение данных, SQL-запрос извлекает данные из таблиц;
- обработка данных, данные обрабатываются согласно заданным выборкам, алгоритмам;
- обработка исключений, Если возникает ошибка на любом этапе выполнения, она логируется через logging.error(), и функция возвращает пустой DataFrame, что позволяет системе продолжить работу.

Блок визуализации данных, основные компоненты и описание.

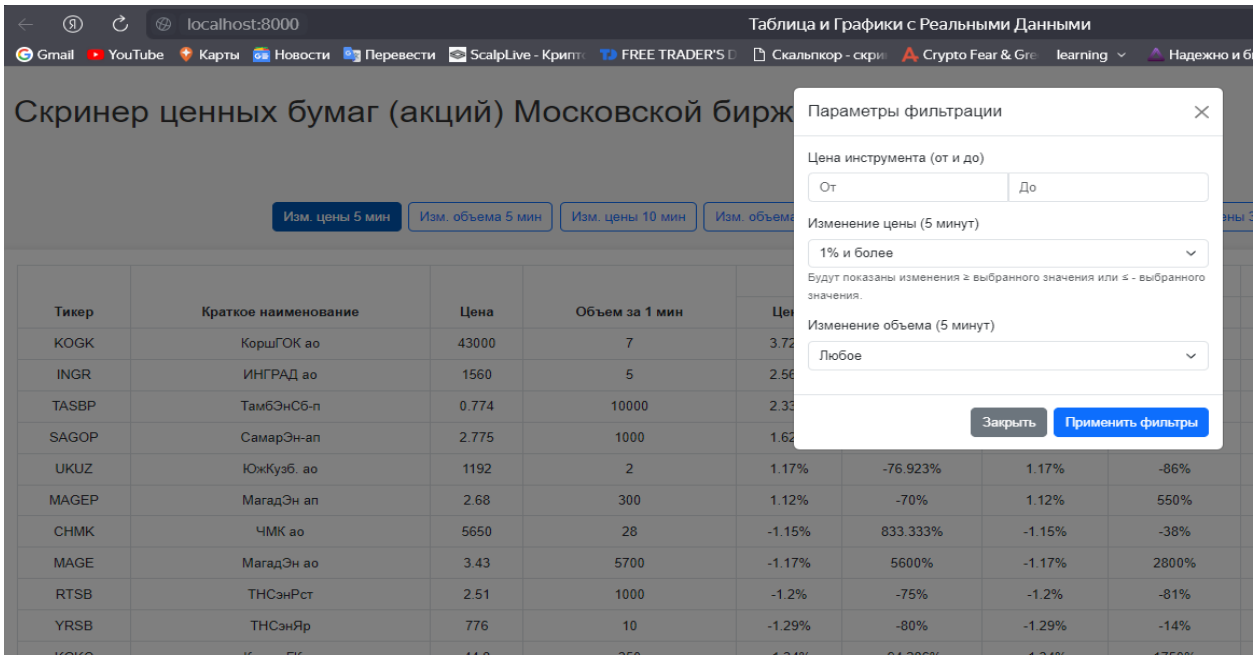
В части визуализации данных используется html страница приложения проекта реализованого на фрэймворке Django. Для этого на стороне фрэймворка разработан шаблон.

Визуализация табличного представления предоставляет детальную информацию об инструментах в структурированном формате, облегчающем сравнение и анализ данных. На рисунке 11, 12 отображено табличное представление программного обеспечения «Скринер» и окно параметров фильтрации.



Тикер	Короткое наименование	Цена	Объем за 1 мин	Изменение (5 мин)		Изменение (10 мин)		Изменение (15 мин)		Изменение (30 мин)		Изменение (1 час)	
				Цена	Объем	Цена	Объем	Цена	Объем	Цена	Объем	Цена	Объем
NNSB	ТНСэнНН ао	5050	2	2.77%	200%	2.77%	-91%	2.77%	-14%	0.79%	10700%	0.79%	0%
MISBP	ТНСэМаЭл-п	59.7	100	1.84%	0%	1.84%	0%	1.84%	-25%	2.01%	17%	2.35%	0%
VGSBP	ВолгЭнСб-п	10.1	1000	1.78%	0%	1.78%	-50%	1.78%	-58%	1.78%	-15%	1.78%	95%
KRKOP	ТКЗКК ап	13.7	100	1.46%	-99.281%	1.46%	-5%	1.46%	-57%	1.46%	3875%	0.58%	0%
KCHE	КамчатЭ ао	0.625	10000	1.44%	-87.5%	1.44%	-53%	1.44%	46%	1.44%	-33%	0.64%	567%
SARE	СаратЭн-ао	0.479	30000	1.25%	0%	1.25%	100%	1.25%	-56%	1.25%	-42%	2.09%	196%
TGKN	ТГК-14	0.01128	100000	1.15%	-13.58%	1.24%	-50%	1.42%	-23%	1.51%	-99%	0.53%	160%
JNOS	Славн-ЯНОС	20.3	400	0.74%	-75%	0.74%	-86%	0.74%	-92%	0.74%	1850%	0.74%	0%
YRSBP	ТНСэнЯр-п	204	10	0.74%	-94.444%	0.74%	850%	0.74%	1900%	0.74%	0%	0.74%	0%
TASBP	ТамбЭнСб-п	0.84	10000	0.71%	0%	0.71%	150%	0.71%	-32%	0.71%	-62%	0.95%	2220%
TASB	ТамбЭнСб	1.596	1000	0.63%	50%	0.63%	257%	0.63%	45%	1%	-58%	1.63%	489%
MDMG	MDMG-ao	869.7	1	0.54%	401.475%	0.56%	277%	0.46%	112%	0.67%	46%	0.34%	65%
IGST	Ижсталь2ао	8720	2	0.46%	100%	0.46%	-93%	0.46%	-92%	1.15%	86%	1.15%	337%
KCHEP	КамчатЭ ап	1.085	80000	0.46%	200%	0.46%	-67%	0.46%	-20%	0.46%	108%	0.46%	0%
RGSS	РГС СК ао	0.272	40000	0.44%	-81.481%	0.44%	-22%	0.44%	3%	0.29%	-76%	0%	-67%
SAREP	СаратЭн-ап	0.2665	100000	0.38%	-90%	0.38%	0%	0.38%	-7%	0.38%	-18%	0.38%	362%

Рисунок 11 - Табличное представление ПО «Скринер»



Тикер	Краткое наименование	Цена	Объем за 1 мин				
KOGK	КоршГОК ао	43000	7	3.72			
INGR	ИНГРАД ао	1560	5	2.56			
TASBP	ТамбЭнСб-п	0.774	10000	2.33			
SAGOP	СамарЭн-ап	2.775	1000	1.62			
UKUZ	ЮжКузб. ао	1192	2	1.17%	-76.923%	1.17%	-86%
MAGEP	МагадЭн ап	2.68	300	1.12%	-70%	1.12%	550%
CHMK	ЧМК ао	5650	28	-1.15%	833.333%	-1.15%	-38%
MAGE	МагадЭн ао	3.43	5700	-1.17%	5600%	-1.17%	2800%
RTSB	ТНСэнРст	2.51	1000	-1.2%	-75%	-1.2%	-81%
YRSB	ТНСэнЯр	776	10	-1.29%	-80%	-1.29%	-14%

Рисунок 12 - Параметры фильтрации ПО «Скринер»

Графическое представление предоставляет визуализацию данных каждого инструмента в виде графиков свечей, дополняя информацию ключевыми показателями. На рисунке 13 представлена визуализация в графическом представлении.

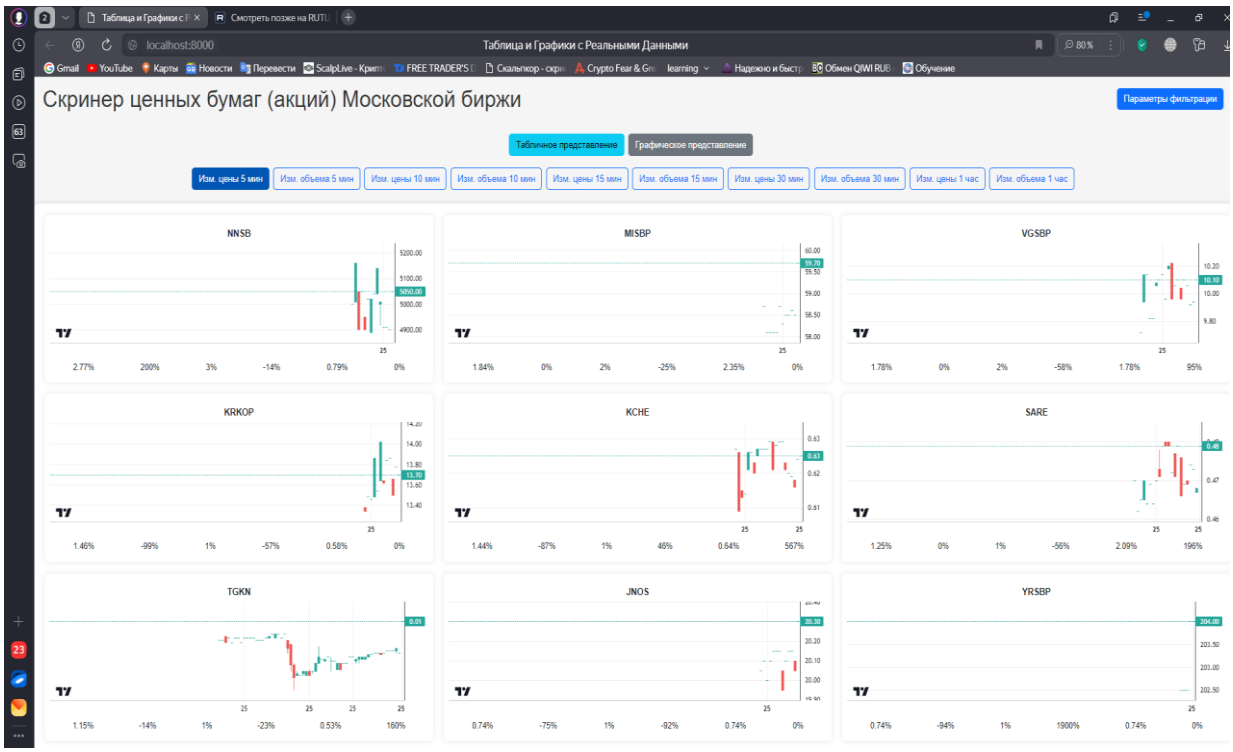


Рисунок 13 - Графическое представление ПО «Скринер»

Функциональные возможности:

– страница обновляется в реальном времени по мере поступления новых данных через WebSocket-соединение, обеспечивая актуальность информации;
– пользователь может сортировать таблицу и графики по любому из параметров изменения цены или объема, нажимая соответствующие кнопки сортировки;
– на странице реализована возможность фильтрации по полям;
– на странице реализована функция подсветки ячеек и изменившимися значениями. Ячейки, в которых значения увеличились по сравнению с предыдущими данными, подсвечиваются зелёным цветом. Ячейки с уменьшившимися значениями подсвечиваются красным цветом.

3.7 Оценка и обоснование экономической эффективности АИС

Для оценки экономической эффективности проекта, как и любого инвестиционного проекта, необходимо учесть затраты на разработку и внедрение системы, а также выгоды, которые система принесет после внедрения. В данной методике будут использованы такие показатели, как срок окупаемости (Payback Period), рентабельность инвестиций (ROI). Рассмотрим более детально этапы методики расчета.

Этап 1 - определение затрат на реализацию проекта. Затраты на реализацию программного обеспечения «Скринер»" можно разделить на несколько категорий:

Затраты на разработку включают в себя:

– зарплата разработчика. Так как для проекта используется один разработчик со средней зарплатой ~ 90 000 рублей в месяц, общие затраты на разработку будут зависеть от количества месяцев, в течение которых проект будет разрабатываться;
– инфраструктура. Сюда входят расходы на серверы и другие технические ресурсы, необходимые для разработки и развертывания системы.

Затраты на внедрение, интеграция с внешними системами. Для успешного функционирования проекта необходимо интегрировать ПО «Скринер»" с API данных в режиме реального времени.

Эксплуатационные затраты: поддержка и обслуживание. После внедрения системы потребуется техническая поддержка, которая будет отвечать за обновление данных, исправление ошибок и расширение функциональности.

Этап 2 - оценка выгод от использования системы. После внедрения ПО «Скринер» можно выделить несколько ключевых направлений, где компания получит выгоду.

– экономия времени сотрудников, система автоматизирует процесс анализа финансовых инструментов, что существенно сократит время, которое трейдеры тратят на анализ данных. Если раньше на анализ могло уходить от 30 минут до 1 часа, то теперь это время сократится до 15-20 минут;
– экономия рабочего времени напрямую влияет на затраты компании на оплату труда трейдеров, так как за меньшее время они смогут проводить больше сделок и принимать решения;
– увеличение количества сделок, благодаря сокращению времени на анализ финансовых инструментов трейдеры смогут совершать больше сделок, что увеличит потенциальную прибыль компании;
– снижение риска ошибок, автоматизация процессов уменьшит количество ошибок, связанных с человеческим фактором, что также приведет к снижению финансовых потерь.

Этап 3 - расчет основных показателей экономической эффективности.

Срок окупаемости (Payback Period) - этот показатель показывает, через какое время проект окупит свои затраты и начнет приносить чистую прибыль.

Рентабельность инвестиций (ROI) - измеряет прибыльность проекта относительно его затрат и показывает, сколько прибыли проект принесет на каждую вложенную единицу средств [17].

Формула расчета ROI:

$$ROI = \frac{(\text{Доходы от проекта} - \text{Затраты на проект})}{\text{Затраты на проект}} \text{X } 100\%, \quad (1)$$

Формула для расчета Payback Period (срока окупаемости):

$$\text{Payback Period} = \frac{\text{Общие затраты на проект}}{\text{Ежегодный доход (экономия или прибыль)}}, \quad (2)$$

Фактические затраты представлены в таблице 9.

Таблица 9 - Фактические затраты на реализацию проекта

Категория затрат	Стоимость (руб.)	Кол-во	Итого (руб.)
Зарплата разработчика	90 000	3 мес	270 000
Инфраструктура (серверы)	300 000		300 000
Обучение сотрудников	50 000	-	50 000
Интеграция с API Московской биржи	200 000	ежегодно	201 000
Поддержка и обслуживание (годовые)	100 000	ежегодно	100 000
ИТОГО			921 000

Общая сумма затрат составляет 921 000 рублей.

Дополнительная прибыль за счет увеличения количества сделок:

Средняя прибыль от каждой сделки: ~1 000 рублей.

Дополнительные сделки: 100 сделок в месяц × 10 месяцев = 1 000 сделок в год.

Дополнительная прибыль: 1000×1000=1 000 000 рублей в год.

Экономия времени: ~250-260 часов в год (сокращение с 2-1,5 часов до 30 минут на анализ данных).

Стоимость часа работы трейдера: ~2 000 рублей.

Экономия: 260×2000 = ~520 000 рублей в год.

Экономическая эффективность проекта оценивается на основании двух ключевых показателей: срок окупаемости (Payback Period) и рентабельность инвестиций (ROI).

$$\text{Payback Period} = \frac{921\,000}{520\,000 + 1\,000\,000} = \frac{921\,000}{1\,520\,000} \approx 0{,}61 \text{ года} \approx 7{,}3 \text{ мес.}$$

Итого срок окупаемости составит ~ 7,3 месяца.

Рентабельность инвестиций (ROI) измеряет прибыльность проекта относительно его затрат и показывает, сколько прибыли проект принесет на каждую вложенную единицу средств. Этот показатель выражается в процентах.

$$ROI = \frac{1\,520\,000 - 921\,000}{921\,000} \text{ X } 100 = \frac{599\,000}{921\,000} \text{ X } 100 \approx 65\%$$

Это означает, что проект приносит 65% прибыли на каждый вложенный рубль, что делает его эффективным с экономической точки зрения. Результаты расчтета экономической эффективности представлены в таблице 10.

Таблица 10 - Расчет экономической эффективности проекта

Показатель	Значение
Общие затраты на проект - разово	921 000 рублей
Ежегодные затраты	301 000 рублей
Годовая экономия времени	520 000 рублей
Дополнительная прибыль	1 000 000 рублей
Доходы от проекта (в год)	1 520 000 рублей
Срок окупаемости (Payback Period)	0.61 года (7,3 месяца)
Рентабельность инвестиций (ROI)	65%

Срок окупаемости проекта составляет чуть более 7 месяцев, что является очень быстрым для подобных проектов.

Рентабельность инвестиций (ROI) равна 65%, что показывает высокую прибыльность проекта.

На рисунке 14 представлен график окупаимости проекта на ближайшие 5 лет с момента внедрения программного обеспечения.

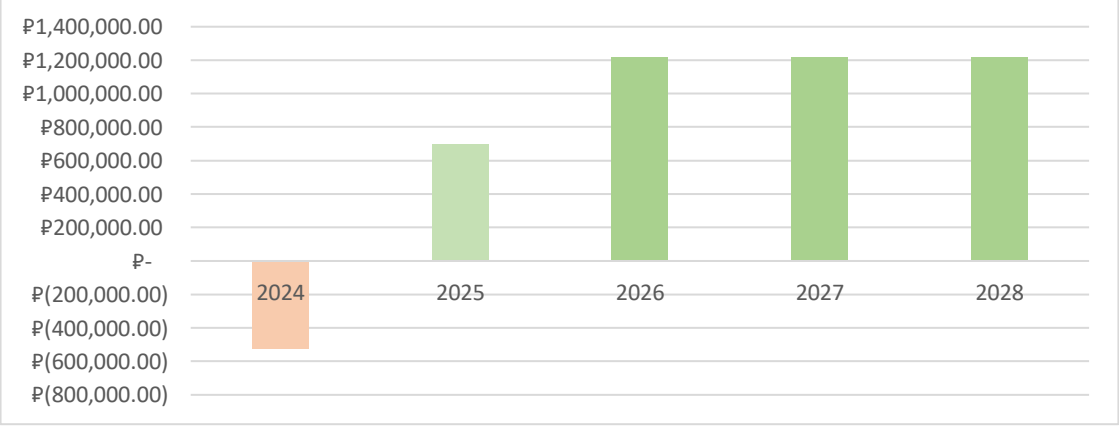


Рисунок 14 - Окупаемость (прибыль) после внедрения проекта, по годам

3.8 Тестирование программного проекта

«Каждый тест представляет собой работы с определенным элементом системы с целью определения на него определенной нагрузки или создание нестандартных ситуаций и наблюдение за их поведением» [4].

Для тестирования страницы, отображающей таблицу и графики с реальными данными, были разработаны следующие сценарии, охватывающие функциональность страницы (таблица 11).

Таблица 11 - Сценарии и результаты тестирования

Название теста	Цель	Шаги	Результат
Табличное отображение данных	Убедиться, что данные отображаются корректно в виде таблицы.	1. Нажать на кнопку "Таблица". 2. Ожидать появления таблицы с данными.	Таблица отображается корректно данные для всех полей с правильной разбивкой по колонкам и строкам.
Графическое отображение данных	Проверить корректность отображения графиков.	1. Нажать на кнопку "График". 2. Убедиться, что графики отображаются корректно.	Для каждого тикера отображается график и соответствующие показатели изменений.
Применение фильтров	Проверить корректность работы фильтров.	1. Открыть окно фильтрации. 2. Установить параметры для фильтров (цена, изменение цены за 5 минут, изменение объема за 5 минут). 3. Применить фильтры.	Отображаются только данные, соответствующие установленным фильтрам.
Сортировка данных	Убедиться, что данные сортируются по выбранным параметрам.	1. Нажать на кнопку сортировки (например, "Изм. цены 5 мин"). 2. Проверить правильность сортировки. 3. Переключить сортировку (возрастание/ убывание).	Данные корректно сортируются в порядке возрастания и убывания при каждом нажатии

Продолжение таблицы 11

Название теста	Цель	Шаги	Результат
Подключение WebSocket	Проверить обновление данных в реальном времени через WebSocket.	3. Переключить сортировку (возрастание/ убывание). 2. Ожидать обновления данных через WebSocket.	Данные обновляются в реальном времени без перезагрузки страницы.
Адаптивность интерфейса	Убедиться в корректности адаптации интерфейса на разных устройствах.	1. Открыть страницу на мобильном устройстве или уменьшить окно браузера. 2. Проверить отображение таблицы и графиков.	Таблица и графики корректно адаптируются под разные размеры экранов.
Подсветка изменений данных	Проверить подсветку изменений данных на странице.	Обновить данные через WebSocket.	При изменении значений в ячейках происходят цветовые изменения: зелёный это увеличение, красный это уменьшение значений.

Эти тест-кейсы покрывают основные функциональные сценарии работы страницы с таблицей и графиками.

Выводы по главе 3.

В третьей главе проведена разработка физического проектирования программного обеспечения «Скринер» для анализа данных. В результате работы были достигнуты следующие ключевые результаты:

Выбрана архитектура на основе фреймворка Django, объединяющая элементы клиент-серверной и микросервисной архитектур. Такое решение обеспечивает высокую производительность, гибкость и надежность системы, а также позволяет эффективно интегрироваться с внешними источниками данных, такими как API Московской биржи.

В процессе выбора технологий для разработки программного обеспечения рассмотрены различные языки программирования и фреймворки, включая Python, Java и JavaScript.

Проведен сравнительный анализа различных СУБД, выбрана СУБД PostgreSQL.

Создана физическая модель данных, включающая основные таблицы для хранения котировок, объемов торгов, исторической информации и данных о финансовых инструментах. Модель обеспечивает эффективное хранение и обработку больших объемов рыночных данных.

Выполнена разработка программного обеспечения «Скринер» которое представлено описание основных блоков: загрузчик данных, обработчик данных и визуализация. Использование Python для серверной логики и JavaScript для клиентских функций обеспечивает гибкость системы и позволяет легко модифицировать или расширять функциональность без существенных изменений в коде.

Выполнено описание функциональности АИС в части ключевых функций системы, включая автоматическую загрузку данных через API, обработку и анализ данных, а также визуализацию результатов в реальном времени с возможностью сортировки и фильтрации. JavaScript используется для динамического обновления интерфейса и улучшения пользовательского опыта.

Проведенное тестирование подтвердило корректность работы всех компонентов системы, включая загрузку данных, обработку, визуализацию и пользовательский интерфейс. Система продемонстрировала высокую производительность и устойчивость к возможным ошибкам и сбоям.

Таким образом, в ходе разработки физического проектирования АИС были достигнуты все поставленные цели, что позволяет считать систему готовой к внедрению и эксплуатации.

Заключение

В данной выпускной квалификационной работе рассмотрена разработка программного обеспечения для технического анализа инструментов финансового рынка для применения в бизнес-процессе компании ООО «А - Лаб групп». Были рассмотрены основные задачи, на решение которых нацелено разработанное программное обеспечение «Скринер».

Разработка программного обеспечения «Скринер» показала себя как успешный проект, направленный на создание эффективного инструмента для отбора и анализа финансовых инструментов российского фондового рынка. В процессе работы были тщательно рассмотрены и обоснованы: выбор архитектуры, технологий разработки и системы управления базами данных.

При выборе технологий разработки были рассмотрены Python, Java и JavaScript. Python выбран как основной язык программирования для серверной части благодаря своей простоте, быстроте разработки и богатой экосистеме библиотек, особенно полезных для анализа данных и веб-разработки. JavaScript использовался для написания клиентских скриптов в шаблонах Django, обеспечивая динамичность и интерактивность пользовательского интерфейса. Это сочетание технологий позволило создать эффективное и удобное в использовании приложение.

Использование фреймворка Django в сочетании с веб-сервером NGINX и сервером Daphne для WebSocket-соединений обеспечило высокую скорость обработки и передачи данных в реальном времени. Выбор PostgreSQL в качестве СУБД позволил эффективно управлять большими объемами данных и выполнять сложные аналитические запросы, что критически важно для трейдинга и технического анализа.

Разработанная физическая модель данных и разделение программного обеспечения на модули загрузки, обработки и визуализации, с использованием Python и JavaScript, обеспечили гибкость и расширяемость системы. Проведенное тестирование подтвердило надежность и корректность работы

всех компонентов, а также соответствие функциональности заявленным требованиям [17].

Разработанное ПО «Скринер» предоставляет трейдерам и аналитикам мощный инструмент для принятия обоснованных решений на основе актуальной и достоверной информации. Автоматизация процессов анализа рыночных данных не только повышает эффективность работы специалистов, но и способствует увеличению прибыли компании.

Проект достиг поставленных целей и рекомендуется к внедрению в профессиональную практику трейдеров компании. Дальнейшее развитие системы может включать интеграцию дополнительных источников данных, расширение функциональности аналитических инструментов и адаптацию под другие рынки и финансовые инструменты.

Список используемой литературы и используемых источников

1. Пресс-релиз Московской биржи «Количество частных инвесторов на Московской бирже превысило 30 миллионов» URL: - https://www.moex.com/n67292 (дата обращения: 10.09.2024).
2. Сунгатуллина, А. Т. Системный анализ и функциональное моделирование бизнес-процессов на основе структурного подхода: учебно-методическое пособие по дисциплине «Моделирование бизнес -процессов» / А. Т. Сунгатуллина, А. А. Базанова. — Москва: Российский университет транспорта (МИИТ), 2021. 115 с.
3. Computer communications protocol WebSocket URL: - https://en.wikipedia.org/wiki/WebSocket (дата обращения 08.09.2024).
4. Лекция «Тестирование на основе моделей» [электронный ресурс] / URL: - http://panda.ispras.ru/~kuliamin/lectures-mbt/Lecture02.pdf (дата обращения 08.09.2024).
5. «О планах удвоения капитилизации» на rbc.ru , URL: https://www.rbc.ru/quote/news/article/65e05bd49a794704415b7f6e?ysclid=m2oy61piop59883835 (дата обращения 25.09.2024).
6. Грекул В. И. Проектирование информационных систем: учебник и практикум / В. И. Грекул, Н. Л. Коровкина, Г. А. Левочкина. — Москва: Издательство Юрайт, 2023. 385 с.
7. Джон Дж. Мэрфи, Технический анализ фьючерсных рынков: Теория и практика, Издательство Альпина Паблишер 2011, 838 с.
8. Официальный сайт Московской биржи URL: - https://www.moex.com/ (дата обращения 14.09.2024).
9. «Один хороший трейд: скрытая информация о высококонкурентном мире частного трейдинга» / М. Беллафиоре; [пер. с анг. А.Соколов]: СмартБук: И-трейд,2012. 400 с.

10. Официальная документация «T-Invest API» [электронный ресурс] / URL: https://russianinvestments.github.io/investAPI/ (дата обращения: 14.09.2024).

11. Описание API Московской биржи URL: - https://iss.moex.com/iss/reference/ (дата обращения 14.09.2024).

12. Описание библиотеки MoexAlgo получение данных и аналитика по рынку акций Московской Биржи (MOEX) URL: - https://moexalgo.readthedocs.io/ru/latest (дата обращения 10.09.2024).

13. Сообщество фрэймворка Django [электронный ресурс] / URL: - https://www.djangoproject.com/community/ (дата обращения: 25.09.2024).

14. Страница СУБД «Postgres Pro» в реестре программного обеспечения URL: - https://reestr.digital.gov.ru/reestr/301574/?sphrase_id=5096043 (дата обращения: 25.09.2024).

15. Карпова И. П. Базы данных. Курс лекций и материалы для практических заданий. – Учебное пособие. – М.: Питер, 2013 – 240 с.

16. Proprietary Trading: What It Is, How It Works, Benefits [электронный ресурс] / URL: - https://www.investopedia.com/terms/p/proprietarytrading.asp (дата обращения: 25.09.2024).

17. Мкртычев, С. В. Прикладная информатика. Бакалаврская работа: электрон. учеб.-метод. пособие / С. В. Мкртычев, О. М. Гущина, А. В. Очеповский ; Тольяттинский государственный университет. – Тольятти: Изд-во ТГУ, 2019.

18. Статья «Objectives of Business» [электронный ресурс] / URL: - https://www.economicsdiscussion.net/managerial-economics/objectives-of-business/31843 (дата обращения 25.09.2024).

19. Статья в wikipedia «Набор требований к транзакционной системе» [электронный ресурс] / URL: - https://ru.wikipedia.org/wiki/ACID (дата обращения: 25.09.2024).

20. Unified Modeling Language, wikipedia [электронный ресурс] / URL: - https://en.wikipedia.org/wiki/Unified_Modeling_Language (дата обращения: 05.10.2024).

Приложение А

Листинг структуры таблиц базы данных

```
CREATE TABLE IF NOT EXISTS public.tradestats
(
    ticker character(12) COLLATE pg_catalog."default" NOT NULL,
    tradedate date NOT NULL,
    tradetime time without time zone NOT NULL,
    pr_open double precision,
    pr_high double precision,
    pr_low double precision,
    pr_close double precision,
    pr_std double precision,
    vol bigint,
    val double precision,
    trades bigint,
    pr_vwap double precision,
    pr_change double precision,
    trades_b bigint,
    trades_s bigint,
    val_b double precision,
    val_s double precision,
    vol_b bigint,
    vol_s bigint,
    disb double precision,
    pr_vwap_b double precision,
    pr_vwap_s double precision,
    systime timestamp without time zone,
    sec_pr_open bigint,
    sec_pr_high bigint,
    sec_pr_low bigint,
    sec_pr_close bigint,
    CONSTRAINT tradestats_pkey PRIMARY KEY (ticker, tradedate, tradetime)
);

CREATE TABLE IF NOT EXISTS public.candles1d
(
    ticker character varying(12) COLLATE pg_catalog."default" NOT NULL,
    open double precision,
    close double precision,
    high double precision,
```

```
    low double precision,
    value double precision,
    volume bigint,
    begin timestamp without time zone NOT NULL,
    "end" timestamp without time zone NOT NULL,
    CONSTRAINT candles1d_pkey PRIMARY KEY (ticker, begin, "end")
);

CREATE TABLE IF NOT EXISTS public.candles1m
(
    ticker text COLLATE pg_catalog."default",
    open double precision,
    close double precision,
    high double precision,
    low double precision,
    value double precision,
    volume double precision,
    begin timestamp without time zone,
    "end" timestamp without time zone
);

CREATE TABLE IF NOT EXISTS public.tickers
(
    "SECID" character(12) COLLATE pg_catalog."default" NOT NULL,
    shortname character varying COLLATE pg_catalog."default",
    lotsize integer,
    decimals integer,
    minstep double precision,
    issuesize bigint,
    isin character varying COLLATE pg_catalog."default",
    regnumber character varying COLLATE pg_catalog."default",
    listlevel integer,
    CONSTRAINT tickers_pkey PRIMARY KEY ("SECID")
);

CREATE TABLE IF NOT EXISTS public.marketdata
(
    "SECID" text COLLATE pg_catalog."default",
    "BOARDID" text COLLATE pg_catalog."default",
    "BID" text COLLATE pg_catalog."default",
```

"BIDDEPTH" text COLLATE pg_catalog."default",
"OFFER" text COLLATE pg_catalog."default",
"OFFERDEPTH" text COLLATE pg_catalog."default",
"SPREAD" bigint,
"BIDDEPTHT" bigint,
"OFFERDEPTHT" bigint,
"OPEN" double precision,
"LOW" double precision,
"HIGH" double precision,
"LAST" double precision,
"LASTCHANGE" double precision,
"LASTCHANGEPRCNT" double precision,
"QTY" bigint,
"VALUE" double precision,
"VALUE_USD" double precision,
"WAPRICE" double precision,
"LASTCNGTOLASTWAPRICE" double precision,
"WAPTOPREVWAPRICEPRCNT" double precision,
"WAPTOPREVWAPRICE" double precision,
"CLOSEPRICE" text COLLATE pg_catalog."default",
"MARKETPRICETODAY" double precision,
"MARKETPRICE" double precision,
"LASTTOPREVPRICE" double precision,
"NUMTRADES" bigint,
"VOLTODAY" bigint,
"VALTODAY" bigint,
"VALTODAY_USD" bigint,
"ETFSETTLEPRICE" text COLLATE pg_catalog."default",
"TRADINGSTATUS" text COLLATE pg_catalog."default",
"UPDATETIME" text COLLATE pg_catalog."default",
"LASTBID" text COLLATE pg_catalog."default",
"LASTOFFER" text COLLATE pg_catalog."default",
"LCLOSEPRICE" double precision,
"LCURRENTPRICE" double precision,
"MARKETPRICE2" double precision,
"NUMBIDS" text COLLATE pg_catalog."default",
"NUMOFFERS" text COLLATE pg_catalog."default",
"CHANGE" double precision,
"TIME" text COLLATE pg_catalog."default",
"HIGHBID" text COLLATE pg_catalog."default",
"LOWOFFER" text COLLATE pg_catalog."default",

```
    "PRICEMINUSPREVWAPRICE" double precision,
    "OPENPERIODPRICE" double precision,
    "SEQNUM" bigint,
    "SYSTIME" text COLLATE pg_catalog."default",
    "CLOSINGAUCTIONPRICE" double precision,
    "CLOSINGAUCTIONVOLUME" double precision,
    "ISSUECAPITALIZATION" double precision,
    "ISSUECAPITALIZATION_UPDATETIME" text COLLATE pg_catalog."default",
    "ETFSETTLECURRENCY" text COLLATE pg_catalog."default",
    "VALTODAY_RUR" bigint,
    "TRADINGSESSION" text COLLATE pg_catalog."default",
    "TRENDISSUECAPITALIZATION" double precision);

ALTER TABLE IF EXISTS public.tradestats
    ADD FOREIGN KEY (ticker)
    REFERENCES public.tickers ("SECID") MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;
ALTER TABLE IF EXISTS public.candles1d
    ADD FOREIGN KEY (ticker)
    REFERENCES public.tickers ("SECID") MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;
ALTER TABLE IF EXISTS public.candles1m
    ADD FOREIGN KEY (ticker)
    REFERENCES public.tickers ("SECID") MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;
ALTER TABLE IF EXISTS public.marketdata
    ADD FOREIGN KEY ("SECID")
    REFERENCES public.tickers ("SECID") MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;
END;
```

Приложение Б

Листинг программного обеспечения

load_candles1d.py

```
import pandas as pd
from sqlalchemy import create_engine, func
from sqlalchemy.orm import sessionmaker
from sqlalchemy_models import Candle1d, Ticker, Base, LogEntry
from moexalgo import Ticker as MoexTicker, session as session_MOEX,
CandlePeriod
from datetime import datetime
from Config import Config as ConfigMOEX, DATABASE_URL
import logging
from concurrent.futures import ThreadPoolExecutor
import time

from datetime import datetime, timedelta

def generate_dates(year, month):
    # Определяем первый день месяца
    start_date = datetime(year, month, 1)

    # Определяем следующий месяц и год
    if month == 12:
        next_month = 1
        next_year = year + 1
    else:
        next_month = month + 1
        next_year = year

    # Определяем первый день следующего месяца
    end_date = datetime(next_year, next_month, 1)

    # Создаем список дат
    dates = []
    current_date = start_date
    while current_date < end_date:
        dates.append(current_date.strftime('%Y-%m-%d'))
        current_date += timedelta(days=1)

    return dates

def log_to_db(session, level, message, ticker=None, table=None):
    log_entry = LogEntry(level=level, message=message, ticker=ticker,
table=table)
    session.add(log_entry)
    session.commit()

# Настройка логирования
logger = logging.getLogger('candles_logger')
logging.basicConfig(level=logging.INFO)

# Конфигурация
engine = create_engine(DATABASE_URL)
Session = sessionmaker(bind=engine)

def process_ticker(secid, start_date, end_date, session):
    ticker = MoexTicker(secid)
```

```
    # Получаем данные свечей в виде DataFrame
    candles = ticker.candles(start=start_date, end=end_date,
period=CandlePeriod.ONE_DAY)
    #print(f' обозначим непустой массив {candles}')

    # Добавляем столбец tickers со значением secid, и удаляем пробелы

    candles['ticker'] = secid

    # Удаляем пробелы из значений в столбце tickers
    candles['ticker'] = candles['ticker'].str.strip()

    # Перемещаем поле 'tickers' в начало
    columns = ['ticker'] + [col for col in candles.columns if col !=
'ticker']
    candles = candles[columns]

    existing_data =
pd.read_sql(session.query(Candle1d).filter(Candle1d.ticker == secid,

func.DATE(Candle1d.begin) == start_date
                                                                ).statement,
engine
                                    )

    # Удаление пробелов из значений в столбце ticker
    existing_data['ticker'] = existing_data['ticker'].str.strip()


    # Объединение данных для нахождения новых записей
    if not existing_data.empty:
        merged_data = pd.merge(
            candles,
            existing_data,
            on=['ticker', 'begin', 'end'],
            how='left',
            indicator=True
        )
        # Переименование столбцов с суффиксом _x
        merged_data.rename(columns=lambda col: col.replace('_x', '') if
col.endswith('_x') else col, inplace=True)

        # Удаление столбцов с суффиксами _x и _y
        suffixes_to_remove = ['_y']
        for suffix in suffixes_to_remove:
            columns_to_drop = [col for col in merged_data.columns if
col.endswith(suffix)]
            merged_data.drop(columns=columns_to_drop, inplace=True)
        # Фильтрация новых данных
        new_data = merged_data[merged_data['_merge'] ==
'left_only'].drop(columns=['_merge'])
    else:
        new_data = candles

    if not new_data.empty:
        # Сохранение новых данных в базу данных
        new_data.to_sql('candles1d', engine, if_exists='append', index=False)
def load_candle_data(date: str = None):
    """Загружает свечи для всех тикеров и сохраняет их в базу данных."""
    session = Session()
    start = datetime.now()
```

```
    authorize = session_MOEX.authorize(ConfigMOEX.Login, ConfigMOEX.Password)
    if authorize:
        log_to_db(session, 'INFO', "Статус соединения с MOEX API: Соединение
установлено.", table='candles')

    # Если дата не указана, используем текущую дату
    if date is None:
        date_now = datetime.now()
        date_old = date_now - timedelta(days=1)
        date = date_old.strftime('%Y-%m-%d')
    try:
        tickers = session.query(Ticker.SECID).all()
        log_to_db(session, 'INFO', f"Загружено {len(tickers)} тикеров для
обработки.", table='candles')

        with ThreadPoolExecutor(max_workers=6) as executor:
            for secid_tuple in tickers:
                secid = secid_tuple[0].strip()
                executor.submit(process_ticker, secid, date, date, session)

    except Exception as e:
        log_to_db(session, 'ERROR', f"Ошибка при загрузке данных свечей:
{e}", table='candles')

    finally:
        fin = datetime.now()
        log_to_db(session, 'INFO', f'Время выполнения запроса {str(fin -
start)}', table='candles')
        session.close()

if __name__ == "__main__":

    load_candle_data()
```

load_candles1m.py

```
      import pandas as pd
from sqlalchemy import create_engine, func
from sqlalchemy.orm import sessionmaker
from sqlalchemy_models import Candle1, Ticker, Base, LogEntry
from moexalgo import Ticker as MoexTicker, session as session_MOEX,
CandlePeriod
from datetime import datetime
from Config import Config as ConfigMOEX, DATABASE_URL
import logging
from concurrent.futures import ThreadPoolExecutor
import time

def log_to_db(session, level, message, ticker=None, table=None):
    log_entry = LogEntry(level=level, message=message, ticker=ticker,
table=table)
    session.add(log_entry)
    session.commit()

# Настройка логирования
logger = logging.getLogger('candles_logger')
logging.basicConfig(level=logging.INFO)
```

```
                    # Конфигурация
                engine = create_engine(DATABASE_URL)
                 Session = sessionmaker(bind=engine)

            def process_ticker(secid, start_date, end_date, session):
                            ticker = MoexTicker(secid)

                    # Получаем данные свечей в виде DataFrame
                candles = ticker.candles(start=start_date, end=end_date,
                            period=CandlePeriod.ONE_MINUTE)
                    #print(f' обозначим непустой массив {candles}')

        # Добавляем столбец tickers со значением secid, и удаляем пробелы

                            candles['ticker'] = secid

                    # Удаляем пробелы из значений в столбце tickers
                candles['ticker'] = candles['ticker'].str.strip()

                        # Перемещаем поле 'tickers' в начало
        columns = ['ticker'] + [col for col in candles.columns if col !=
                                    'ticker']
                            candles = candles[columns]

    existing_data = pd.read_sql(session.query(Candle1).filter(Candle1.ticker
                                    == secid,

                        func.DATE(Candle1.begin) == start_date
                                                                ).statement,
                                        engine
                                                    )

                    # Удаление пробелов из значений в столбце ticker
                existing_data['ticker'] = existing_data['ticker'].str.strip()

                    # Объединение данных для нахождения новых записей
                                if not existing_data.empty:
                                    merged_data = pd.merge(
                                            candles,
                                        existing_data,
                                    on=['ticker', 'begin', 'end'],
                                            how='left',
                                        indicator=True
                                            )
                        # Переименование столбцов с суффиксом _x
                 merged_data.rename(columns=lambda col: col.replace('_x', '') if
                        col.endswith('_x') else col, inplace=True)

                        # Удаление столбцов с суффиксами _x и _y
                                suffixes_to_remove = ['_y']
                            for suffix in suffixes_to_remove:
                    columns_to_drop = [col for col in merged_data.columns if
                                col.endswith(suffix)]
                    merged_data.drop(columns=columns_to_drop, inplace=True)
                                # Фильтрация новых данных
                        new_data = merged_data[merged_data['_merge'] ==
                         'left_only'].drop(columns=['_merge'])
                                        else:
                                    new_data = candles
```

```
    if not new_data.empty:

        # Сохранение новых данных в базу данных

        new_data.to_sql('candles1m', engine, if_exists='append', index=False)
        #log_to_db(session, 'INFO', f"{secid} - Загружены свечи с
{start_date} по {end_date}", table='candles')
    #else:
        #log_to_db(session, 'INFO', f"{secid} - Нет новых данных для
вставки.", table='candles')

def load_candle_data(date: str = None):
    """Загружает свечи для всех тикеров и сохраняет их в базу данных."""
    session = Session()
    start = datetime.now()

    authorize = session_MOEX.authorize(ConfigMOEX.Login, ConfigMOEX.Password)
    if authorize:
        log_to_db(session, 'INFO', "Статус соединения с MOEX API: Соединение
установлено.", table='candles')

    # Если дата не указана, используем текущую дату
    if date is None:
        date = datetime.now().strftime('%Y-%m-%d')

    try:
        tickers = session.query(Ticker.SECID).all()
        log_to_db(session, 'INFO', f"Загружено {len(tickers)} тикеров для
обработки.", table='candles')

        with ThreadPoolExecutor(max_workers=6) as executor:
            for secid_tuple in tickers:
                secid = secid_tuple[0].strip()
                executor.submit(process_ticker, secid, date, date, session)

    except Exception as e:
        log_to_db(session, 'ERROR', f"Ошибка при загрузке данных свечей:
{e}", table='candles')

    finally:
        fin = datetime.now()
        log_to_db(session, 'INFO', f'Время выполнения запроса {str(fin -
start)}', table='candles')
        session.close()

if __name__ == "__main__":
    while True:
        load_candle_data('2024-09-25')
        time.sleep(5)  # 5 секунд
```

load_stats.py

```
     import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy_models import TradeStats, Ticker, Base, LogEntry
from moexalgo import Ticker as MoexTicker, session as session_MOEX
from datetime import datetime, timedelta
```

```
from Config import Config as ConfigMOEX, DATABASE_URL
import logging
from concurrent.futures import ThreadPoolExecutor
import time

def log_to_db(session, level, message, ticker=None, table=None):
    log_entry = LogEntry(level=level, message=message, ticker=ticker,
table=table)
    session.add(log_entry)
    session.commit()

pd.set_option('display.max_columns', None)

# Настройка логирования
logger = logging.getLogger('tradestats_logger')
logging.basicConfig(level=logging.INFO)

# Конфигурация
engine = create_engine(DATABASE_URL)
Session = sessionmaker(bind=engine)

def process_ticker(secid, date, session):
    ticker = MoexTicker(secid)

    # Получаем данные в виде DataFrame
    stats = ticker.tradestats(start=date, end=date, offset=0)

    if stats.empty:
        #logger.info(f"Нет данных для {secid} на дату {date}")
        log_to_db(session, 'INFO', f"Нет данных для {secid} на дату {date}",
table = 'tradestats')
        return
    # Проверка на наличие значений в колонке 'ticker'
    if stats['ticker'].isnull().any():
        #logger.warning(f"Обнаружены null значения в колонке 'ticker' для
{secid}.")
        log_to_db(session, 'WARNING', f"Обнаружены null значения в колонке
'ticker' для {secid}.", table = 'tradestats')
        stats = stats[stats['ticker'].notnull()]

    # Получение существующих данных
    existing_data = pd.read_sql(
        session.query(TradeStats).filter(
            TradeStats.ticker == secid,
            TradeStats.tradedate == date
        ).statement,
        engine
    )
    # Удаление пробелов из значений в столбце ticker
    existing_data['ticker'] = existing_data['ticker'].str.strip()

    # Объединение данных для нахождения новых записей
    if not existing_data.empty:
        merged_data = pd.merge(
            stats,
            existing_data,
            on=['ticker', 'tradedate', 'tradetime'],
            how='left',
            indicator=True
```

```
        )

        # Переименование столбцов с суффиксом _y
        merged_data.rename(columns=lambda col: col.replace('_x', '') if
col.endswith('_x') else col, inplace=True)

        # Удаление столбцов с суффиксами _x и _y
        suffixes_to_remove = ['_y']
        for suffix in suffixes_to_remove:
            columns_to_drop = [col for col in merged_data.columns if
col.endswith(suffix)]
            merged_data.drop(columns=columns_to_drop, inplace=True)

        # Фильтрация новых данных
        new_data = merged_data[merged_data['_merge'] ==
'left_only'].drop(columns=['_merge'])
    else:
        new_data = stats

    if not new_data.empty:
        # Сохранение новых данных в базу данных
        new_data.to_sql('tradestats', engine, if_exists='append',
index=False)
        #log_to_db(session, 'INFO', f"{secid} - Получены данные по {date}",
table = 'tradestats')
    #else:
        #log_to_db(session, 'INFO', f"{secid} - Нет новых данных для
вставки.", table = 'tradestats')


def load_trade_stats(date: str = None):
    """Загружает торговую статистику для всех тикеров и сохраняет их в базу
данных."""
    session = Session()
    start = datetime.now()

    authorize = session_MOEX.authorize(ConfigMOEX.Login, ConfigMOEX.Password)
    if authorize:
        #logger.info("Статус соединения: Соединение установлено.")
        log_to_db(session, 'INFO', "Статус соединения с MOEX API: Соединение
установлено.", table = 'tradestats')

    # Если дата не указана, используем текущую дату
    if date is None:
        date = datetime.now().strftime('%Y-%m-%d')

    try:
        tickers = session.query(Ticker.SECID).all()
        #logger.info(f"Загружено {len(tickers)} тикеров для обработки.")
        log_to_db(session, 'INFO', f"Загружено {len(tickers)} тикеров для
обработки.", table = 'tradestats')

        with ThreadPoolExecutor(max_workers=4) as executor:
            for secid_tuple in tickers:
                secid = secid_tuple[0].strip()
                executor.submit(process_ticker, secid, date, session)

    except Exception as e:
        #logger.error(f"Ошибка при загрузке торговых данных: {e}")
```

```
        log_to_db(session, 'ERROR', f"Ошибка при загрузке торговых данных:
{e}", table = 'tradestats')

    finally:
        fin = datetime.now()
        #logger.info(f'Время выполнения запроса  {str(fin - start)}')
        log_to_db(session, 'INFO', f'Время выполнения запроса  {str(fin -
start)}', table = 'tradestats')
        session.close()

if __name__ == "__main__":
    while True:
        load_trade_stats('2024-09-25')
        time.sleep(2 * 60)  # 2 минуты = 2 * 60 секунд
```

get_candles2.py

```
import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from datetime import datetime, timedelta
import logging
import time
from .load.Config import DATABASE_URL
import numpy as np

pd.set_option('display.max_columns', None)

# Настройка логирования
logger = logging.getLogger('candles_logger')
logging.basicConfig(level=logging.INFO)

# Подключение к базе данных
engine = create_engine(DATABASE_URL)
Session = sessionmaker(bind=engine)

# Функция загрузки данных из таблицы candles1m и marketdata
def get_candle_data_n(date: str = None):
    session = Session()

    # Если дата не указана, используем текущую дату
    if date is None:
        date = datetime.now().strftime('%Y-%m-%d')

    query = f"""
        SELECT c.*, m."LAST"
        FROM public.candles1m c
        LEFT JOIN public.marketdata m ON c.ticker = m."SECID"
        WHERE DATE(c.begin) = '{date}'
    """
    try:
        df = pd.read_sql(query, engine)
        df['begin'] = pd.to_datetime(df['begin'])
    except Exception as e:
        logger.error(f"Ошибка при загрузке данных из таблицы candles1m: {e}")
        df = pd.DataFrame()  # В случае ошибки возвращаем пустой DataFrame
```

```
    finally:

        session.close()
    return df

# Функция для поиска ближайшей записи к указанной дате
def find_closest_record(group, target_time):
    closest_record = group[group['begin'] == target_time]
    if closest_record.empty:
        closest_record = group.loc[group['begin'] <=
target_time].sort_values(by='begin', ascending=False).head(1)
    return closest_record

# Функция расчета изменений цены и объема
def calculate_changes(df):
    df = df.sort_values(by='begin')
    max_dates = df.groupby('ticker')['begin'].max().reset_index()
    max_dates.columns = ['ticker', 'max_date']
    df = df.merge(max_dates, on='ticker', how='left')

    results = []

    def calculate_for_ticker(group):
        max_date = group['max_date'].iloc[0]
        max_data = group[group['begin'] == max_date]

        if max_data.empty:
            return pd.DataFrame()

        intervals = {
            '5m': pd.Timedelta(minutes=5),
            '10m': pd.Timedelta(minutes=10),
            '15m': pd.Timedelta(minutes=15),
            '30m': pd.Timedelta(minutes=30),
            '1h': pd.Timedelta(minutes=60)
        }

        changes = {
            'ticker': group['ticker'].iloc[0],
            'max_date': max_date,
            'price_at_max': max_data['LAST'].iloc[0],
            'volume_at_max': max_data['volume'].iloc[0]
        }

        for label, delta in intervals.items():
            time_before_max = max_date - delta
            previous_data = find_closest_record(group, time_before_max)
            if not previous_data.empty:
                changes[f'price_at_{label}_before'] =
previous_data['close'].iloc[0]
                changes[f'price_change_{label}'] =
round((changes['price_at_max'] - changes[f'price_at_{label}_before']) * 100 /
changes['price_at_max'], 2)
                #changes[f'volume_at_{label}_before'] =
previous_data['value'].iloc[0]
                #changes[f'volume_change_{label}'] =
round((changes['volume_at_max'] - changes[f'volume_at_{label}_before']) * 100
/ changes['volume_at_max'],  3)
            else:
                changes[f'price_at_{label}_before'] = None
                changes[f'price_change_{label}'] = None
                #changes[f'volume_at_{label}_before'] = None
```

```
                #changes[f'volume_change_{label}'] = None

        return pd.DataFrame([changes])

    results = [calculate_for_ticker(group) for _, group in
df.groupby('ticker') if not group.empty]
    results = [result for result in results if not result.empty]
    final_df = pd.concat(results, ignore_index=True) if results else
pd.DataFrame()

    for column in final_df.columns:
        if 'price_change_' in column or 'volume_change_' in column:
            final_df[column] = final_df[column].round(3)

    final_df = final_df.replace({np.nan: None})

    return final_df
```

get_candles3.py

```
from .load.Config import DATABASE_URL
import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from datetime import datetime, timedelta
import numpy as np
import logging

# Подключение к базе данных
engine = create_engine(DATABASE_URL)
Session = sessionmaker(bind=engine)

def calculate_volume_changes_for_date(date: str = None):


    # Если дата не указана, используем текущую дату
    if date is None:
        date = datetime.now().strftime('%Y-%m-%d')

    try:
        # Настройка подключения к базе данных
        #logging.info(f"Подключение к базе данных установлено.")

        # Загрузка данных из базы данных с фильтрацией по дате
        query = f"""
            SELECT ticker, tradedate, tradetime, vol, val, trades, val_b,
val_s, vol_b, vol_s
            FROM public.tradestats
            WHERE tradedate = '{date}';
        """
        df = pd.read_sql(query, engine)

        if df.empty:
            logging.warning(f"Нет данных для даты: {date}")
            return pd.DataFrame()
```

```
        #logging.info(f"Данные загружены, количество записей: {len(df)}")
        # Удаление пробелов из значений столбца 'ticker'
        df['ticker'] = df['ticker'].str.strip()

        # Преобразование данных
        df['datetime'] = pd.to_datetime(df['tradedate'].astype(str) + ' ' +
df['tradetime'].astype(str))
        df = df.sort_values(by=['ticker', 'datetime'])

        # Функция для расчета изменения объема
        def calculate_volume_changes(group, periods):
            result = group.copy()
            for period in periods:
                # Вычисление суммарного объема за период
                result[f'vol_{period}min'] =
group['vol'].rolling(window=period//5, min_periods=1).sum()
                # Вычисление разницы между текущим объемом и объемом за
предыдущий аналогичный период
                result[f'vol_diff_{period}min'] =
result[f'vol_{period}min'].diff(periods=period//5).fillna(0)
                # Вычисление процентного изменения объема
                result[f'volume_change_{period}m'] =
(result[f'vol_diff_{period}min'] /
result[f'vol_{period}min'].shift(periods=period//5)).fillna(0) * 100
                # Округление до 3 знаков после запятой
                result[f'volume_change_{period}m'] =
result[f'volume_change_{period}m'].round(3)
            return result

        # Периоды для расчета в минутах
        periods = [5, 10, 15, 30, 60]

        # Применение функции к каждому тикеру с параметром
include_groups=False
        df_result = df.groupby('ticker',
group_keys=False).apply(calculate_volume_changes, periods=periods)

        # Фильтрация по максимальной дате и времени для каждого тикера
        df_max_time =
df_result.groupby('ticker').tail(1).reset_index(drop=True)

        #logging.info("Расчет завершен.")
        # Возвращаем результат в виде DataFrame
        return df_max_time[['ticker', 'datetime'] + [f'vol_diff_{p}min' for p
in periods] + [f'volume_change_{p}m' for p in periods]]
    except Exception as e:
        logging.error(f"Ошибка при выполнении: {str(e)}")
        return pd.DataFrame()
```

get_candles_data.py

```
import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from datetime import datetime
import logging
from .load.Config import DATABASE_URL
```

```
# Настройка логирования
logger = logging.getLogger('candles_logger')
logging.basicConfig(level=logging.INFO)

# Подключение к базе данных
engine = create_engine(DATABASE_URL)
Session = sessionmaker(bind=engine)

pd.set_option('display.max_columns', None)

def get_candles_data(date: str = None):
    """
    Возвращает DataFrame с двумя полями: ticker и candles_data.
    В поле candles_data содержится набор минутных свечей для каждого тикера,
ограниченный до 60 записей.
    """
    session = Session()

    # Если дата не указана, используем текущую дату
    if date is None:
        date = datetime.now().strftime('%Y-%m-%d')

    try:
        # Запрос для получения данных из таблицы candles1m
        candles_query = f"""
            SELECT ticker, open, close, high, low, volume, begin, "end"
            FROM public.candles1m
            WHERE DATE(begin) = '{date}'
            ORDER BY ticker, begin ASC
        """
        candles_df = pd.read_sql(candles_query, engine)
        candles_df['begin'] = pd.to_datetime(candles_df['begin'])

        # Запрос для получения данных из таблицы marketdata
        marketdata_query = """
            SELECT "SECID" as ticker,
                   "LAST" as open,
                   "LAST" as close,
                   "LAST" as high,
                   "LAST" as low,
                   date_trunc('minute', "SYSTIME"::timestamp) as begin,
                   date_trunc('minute', "SYSTIME"::timestamp) + interval '1
minute' as "end",
                   0 as volume
            FROM public.marketdata
            ORDER BY ticker, begin ASC
        """

        marketdata_df = pd.read_sql(marketdata_query, engine)
        marketdata_df['begin'] = pd.to_datetime(marketdata_df['begin'])

        # Объединяем два DataFrame
        combined_df = pd.concat([candles_df, marketdata_df],
ignore_index=True)

        # Преобразуем дату в Unix timestamp
        combined_df['begin'] = combined_df['begin'].apply(lambda x:
int(x.timestamp()))

        # Группировка по тикеру, отбор последних 60 записей и создание списка
свечей для каждого тикера
```

```
        grouped = combined_df.groupby('ticker').apply(
            lambda x: x.iloc[-60:][['begin', 'open', 'high', 'low', 'close',
'volume']].to_dict('records')
        ).reset_index()
        grouped.columns = ['ticker', 'candles_data']

        return grouped
    except Exception as e:
        logger.error(f"Ошибка при загрузке данных из таблиц: {e}")
        return pd.DataFrame(columns=['ticker', 'candles_data'])  # Возвращаем
пустой DataFrame в случае ошибки
    finally:
        session.close()
"""data = get_candles_data('2024-09-16')
print(data)
output_file = 'candles_data.csv'
data.to_csv(output_file, index=False)"""
```

Шаблон changes2.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <!-- Метатег для адаптивности -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Таблица и Графики с Реальными Данными</title>
    <!-- Подключаем Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css
" rel="stylesheet">
    <!-- Подключаем библиотеку lightweight-charts -->
    <script src="https://unpkg.com/lightweight-charts/dist/lightweight-
charts.standalone.production.js"></script>
    <!-- Подключаем библиотеку Tippy.js для отображения тултипов -->
    <link rel="stylesheet"
href="https://unpkg.com/tippy.js@6/animations/scale.css">
    <script src="https://unpkg.com/@popperjs/core@2"></script>
    <script src="https://unpkg.com/tippy.js@6"></script>
    <style>
        body {
            background-color: #f5f5f5;
            font-family: Arial, sans-serif;
        }
        h1 {
            margin: 20px 0;
            text-align: center;
            color: #333;
        }
        .grid-container, .table-container {
            display: none;
            margin: 20px auto;
            padding: 20px;
            box-shadow: 0 0 10px rgba(0,0,0,0.1);
            border-radius: 8px;
            background: #fff;
            width: 100%; /* Занимаем всю ширину */
        }
```

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr); /* 3 столбца */
    grid-auto-rows: auto;
    gap: 20px;
}
.grid-item {
    background-color: #ffffff;
    border-radius: 8px;
    padding: 10px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
    text-align: center;
    transition: background-color 0.5s ease;
}
.chart-container {
    width: 100%; /* Подстраивается под сетку */
    margin: 0 auto;
    margin-top: 10px;
    position: relative;
}
.chart-title {
    position: relative;
    font-size: 16px;
    font-weight: bold;
    margin-bottom: 5px;
}
.chart {
    width: 100%;
    height: 200px;
}
.values-container {
    display: flex;
    flex-wrap: nowrap;
    overflow-x: auto;
    justify-content: flex-start;
    font-size: 0.9em;
}
.values-container div {
    flex: 1 1 auto;
    margin: 5px;
    cursor: pointer; /* Указатель при наведении */
    white-space: nowrap;
    transition: background-color 0.5s ease; /* Для подсветки */
}
.increase {
    background-color: #d4edda !important;
}
.decrease {
    background-color: #f8d7da !important;
}
.table-container table {
    width: 100%; /* Занимаем всю ширину */
    text-align: center;
}
.pagination {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    margin-top: 20px;
}
.pagination button {
    margin: 5px;
```

```
        }
        .btn.active {
            background-color: #0056b3;
            border-color: #0056b3;
            color: #fff;
        }
        /* Адаптивность для мобильных устройств */
        @media (max-width: 768px) {
            .grid-container {
                grid-template-columns: repeat(1, 1fr); /* 1 столбец на узких
экранах */
            }
            .values-container {
                flex-wrap: wrap;
            }
            .chart-container {
                width: 100%; /* На мобильных устройствах ширина 100% */
            }
        }
    </style>
</head>
<body>
    <!-- Заголовок с кнопкой фильтрации -->
    <div class="d-flex justify-content-between align-items-center p-3">
        <h1>Скринер ценных бумаг (акций) </h1>
        <button type="button" class="btn btn-primary" data-bs-toggle="modal"
data-bs-target="#filterModal">
            Параметры фильтрации
        </button>
    </div>

    <!-- Кнопки переключения между видами -->
    <div class="d-flex justify-content-center mb-3">
        <button id="tableViewButton" class="btn btn-info me-2">Табличное
представление</button>
        <button id="chartViewButton" class="btn btn-secondary me-
2">Графическое представление</button>
    </div>

    <!-- Кнопки сортировки -->
    <div id="sortingButtons" class="d-flex justify-content-center mb-3 flex-
wrap">
        <button class="btn btn-outline-primary m-1" data-
sort="price_change_5m">Изм. цены 5 мин</button>
        <button class="btn btn-outline-primary m-1" data-
sort="volume_change_5m">Изм. объема 5 мин</button>
        <button class="btn btn-outline-primary m-1" data-
sort="price_change_10m">Изм. цены 10 мин</button>
        <button class="btn btn-outline-primary m-1" data-
sort="volume_change_10m">Изм. объема 10 мин</button>
        <button class="btn btn-outline-primary m-1" data-
sort="price_change_15m">Изм. цены 15 мин</button>
        <button class="btn btn-outline-primary m-1" data-
sort="volume_change_15m">Изм. объема 15 мин</button>
        <button class="btn btn-outline-primary m-1" data-
sort="price_change_30m">Изм. цены 30 мин</button>
        <button class="btn btn-outline-primary m-1" data-
sort="volume_change_30m">Изм. объема 30 мин</button>
        <button class="btn btn-outline-primary m-1" data-
sort="price_change_1h">Изм. цены 1 час</button>
        <button class="btn btn-outline-primary m-1" data-
sort="volume_change_1h">Изм. объема 1 час</button>
```

```
    </div>

    <!-- Модальное окно для фильтрации -->
    <div class="modal fade" id="filterModal" tabindex="-1" aria-
labelledby="filterModalLabel" aria-hidden="true">
        <!-- Содержимое модального окна -->
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <!-- Заголовок модального окна -->
                    <h5 class="modal-title" id="filterModalLabel">Параметры
фильтрации</h5>
                    <button type="button" class="btn-close" data-bs-
dismiss="modal" aria-label="Закрыть"></button>
                </div>
                <div class="modal-body">
                    <!-- Форма фильтрации -->
                    <form id="filterForm">
                        <!-- Поля для фильтрации price_at_max -->
                        <div class="mb-3">
                            <label class="form-label">Цена инструмента (от и
до)</label>
                            <div class="input-group">
                                <input type="number" class="form-control"
id="priceAtMaxFrom" placeholder="От" min="0" max="300000">
                                <input type="number" class="form-control"
id="priceAtMaxTo" placeholder="До" min="0" max="300000">
                            </div>
                        </div>

                        <!-- Выпадающее меню для price_change_5m -->
                        <div class="mb-3">
                            <label for="priceChange5m" class="form-
label">Изменение цены (5 минут)</label>
                            <select class="form-select" id="priceChange5m">
                                <option value="">Любое</option>
                                <option value="0.5">0,5% и более</option>
                                <option value="1">1% и более</option>
                                <option value="3">3% и более</option>
                                <option value="5">5% и более</option>
                            </select>
                            <small class="text-muted">Будут показаны
изменения ≥ выбранного значения или ≤ - выбранного значения.</small>
                        </div>

                        <!-- Выпадающее меню для volume_change_5m -->
                        <div class="mb-3">
                            <label for="volumeChange5m" class="form-
label">Изменение объема (5 минут)</label>
                            <select class="form-select" id="volumeChange5m">
                                <option value="">Любое</option>
                                <option value="10">10% и более</option>
                                <option value="30">30% и более</option>
                                <option value="50">50% и более</option>
                                <option value="100">100% и более</option>
                            </select>
                        </div>
                    </form>
                </div>
                <!-- Кнопки модального окна -->
                <div class="modal-footer">
```

```
                    <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Закрыть</button>
                    <button type="button" class="btn btn-primary"
onclick="applyFilters()" data-bs-dismiss="modal">Применить фильтры</button>
                </div>
            </div>
        </div>
    </div>

    <!-- Табличное отображение -->
    <div id="tableContainer" class="table-container">
        <table id="dataTable" class="table table-bordered">
            <thead>
                <tr>
                    <th rowspan="2">Тикер</th>
                    <th rowspan="2">Краткое наименование</th>
                    <th rowspan="2">Цена</th>
                    <th rowspan="2">Объем за 1 мин</th>
                    <th colspan="2">Изменение (5 мин)</th>
                    <th colspan="2">Изменение (10 мин)</th>
                    <th colspan="2">Изменение (15 мин)</th>
                    <th colspan="2">Изменение (30 мин)</th>
                    <th colspan="2">Изменение (1 час)</th>
                </tr>
                <tr>
                    <th>Цена</th>
                    <th>Объем</th>
                    <th>Цена</th>
                    <th>Объем</th>
                    <th>Цена</th>
                    <th>Объем</th>
                    <th>Цена</th>
                    <th>Объем</th>
                    <th>Цена</th>
                    <th>Объем</th>
                </tr>
            </thead>
            <tbody>
                <!-- Табличные данные будут добавляться динамически -->
            </tbody>
        </table>
    </div>

    <!-- Графическое отображение -->
    <div id="gridContainer" class="grid-container">
        <!-- Здесь будут отображаться графики -->
    </div>

    <!-- Пагинация -->
    <div class="pagination" id="pagination"></div>

    <!-- Подключаем Bootstrap JS -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.mi
n.js"></script>

    <!-- Основной JavaScript код -->
    <script>
        let socket;
        let previousData = {};
        let allData = [];
```

```
        let filteredData = [];
        let isTableView = true;
        let currentPage = 1;
        const itemsPerPage = 12;
        let currentSortKey = null;
        let sortOrder = 'desc'; // 'asc' или 'desc'

        // Обработчики событий для переключения видов
        document.getElementById('tableViewButton').addEventListener('click',
function() {
            isTableView = true;
            document.getElementById('tableContainer').style.display =
'block';
            document.getElementById('gridContainer').style.display = 'none';
            document.getElementById('pagination').style.display = 'none';
            renderTable();
        });

        document.getElementById('chartViewButton').addEventListener('click',
function() {
            isTableView = false;
            document.getElementById('tableContainer').style.display = 'none';
            document.getElementById('gridContainer').style.display = 'grid';
            document.getElementById('pagination').style.display = 'flex';
            renderGridPage();
            createPagination();
            updatePagination();
        });

        // Обработчики событий для кнопок сортировки
        const sortingButtons = document.querySelectorAll('#sortingButtons
button');
        sortingButtons.forEach(button => {
            button.addEventListener('click', function() {
                const sortKey = this.getAttribute('data-sort');
                if (currentSortKey === sortKey) {
                    // Переключаем порядок сортировки
                    sortOrder = sortOrder === 'asc' ? 'desc' : 'asc';
                } else {
                    currentSortKey = sortKey;
                    sortOrder = 'desc';
                }
                // Обновляем внешний вид кнопок
                sortingButtons.forEach(btn =>
btn.classList.remove('active'));
                this.classList.add('active');
                // Применяем сортировку и обновляем отображение
                applySorting();
            });
        });

        // Функция подключения к WebSocket
        function connectWebSocket() {
            const protocol = window.location.protocol === 'https:' ? 'wss://'
: 'ws://';
            socket = new WebSocket(protocol + window.location.host +
'/ws/screener/changes2/');

            socket.onopen = function() {
                console.log('WebSocket соединение установлено');
            };
```

```
            socket.onmessage = function(e) {
                document.getElementById('loadingIndicator').style.display =
'none'; // Скрываем индикатор загрузки
                try {
                    const message = JSON.parse(e.data);
                    console.log('Получены данные:', message); // Для отладки
, что то не так , или так не разберусь
                    if (message && message.data) {
                        const data = message.data;
                        allData = data;
                        applyFilters();
                    } else {
                        console.warn("Получено сообщение без данных:",
message);
                    }
                } catch (err) {
                    console.error("Ошибка при разборе сообщения WebSocket:",
err);
                }
            };

            socket.onclose = function() {
                console.error('WebSocket закрыт, попытка
переподключения...');
                setTimeout(connectWebSocket, 5000);
            };
        }
        // Функция применения фильтров
        function applyFilters() {
            const priceAtMaxFromElement =
document.getElementById('priceAtMaxFrom');
            const priceAtMaxToElement =
document.getElementById('priceAtMaxTo');
            const priceChange5mElement =
document.getElementById('priceChange5m');
            const volumeChange5mElement =
document.getElementById('volumeChange5m');

            if (!priceAtMaxFromElement || !priceAtMaxToElement ||
!priceChange5mElement || !volumeChange5mElement) {
                console.error('Элементы фильтрации не найдены в DOM.');
                return;
            }
            const priceAtMaxFrom = priceAtMaxFromElement.value;
            const priceAtMaxTo = priceAtMaxToElement.value;
            const priceChange5m = priceChange5mElement.value;
            const volumeChange5m = volumeChange5mElement.value;

            const priceAtMaxFromValue = priceAtMaxFrom !== '' ?
parseFloat(priceAtMaxFrom) : null;
            const priceAtMaxToValue = priceAtMaxTo !== '' ?
parseFloat(priceAtMaxTo) : null;
            const priceChange5mValue = priceChange5m !== '' ?
parseFloat(priceChange5m) : null;
            const volumeChange5mValue = volumeChange5m !== '' ?
parseFloat(volumeChange5m) : null;

            filteredData = allData.filter(row => {
                const priceAtMaxCondition = (priceAtMaxFromValue === null ||
row.price_at_max >= priceAtMaxFromValue) &&
```

```
                                                (priceAtMaxToValue === null ||
row.price_at_max <= priceAtMaxToValue);

                const priceChangeCondition = (priceChange5mValue === null ||
                    row.price_change_5m >= priceChange5mValue ||
row.price_change_5m <= -priceChange5mValue);

                const volumeChangeCondition = (volumeChange5mValue === null
|| row.volume_change_5m >= volumeChange5mValue);

                return priceAtMaxCondition && priceChangeCondition &&
volumeChangeCondition;
            });

            applySorting();
        }

        // Функция сортировки данных
        function applySorting() {
            if (currentSortKey) {
                filteredData.sort((a, b) => {
                    if (sortOrder === 'asc') {
                        return a[currentSortKey] - b[currentSortKey];
                    } else {
                        return b[currentSortKey] - a[currentSortKey];
                    }
                });
            }

            if (isTableView) {
                renderTable();
            } else {
                currentPage = 1; // Сбрасываем текущую страницу
                renderGridPage();
                createPagination();
                updatePagination();
            }
        }

        // Функции для работы с пагинацией
        function createPagination() {
            const totalPages = Math.ceil(filteredData.length / itemsPerPage);
            const paginationContainer =
document.getElementById('pagination');
            paginationContainer.innerHTML = '';

            for (let i = 1; i <= totalPages; i++) {
                const button = document.createElement('button');
                button.textContent = i;
                button.className = 'btn btn-primary' + (i === currentPage ? '
active' : '');
                button.onclick = () => changePage(i);
                paginationContainer.appendChild(button);
            }
        }
        function updatePagination() {
            const buttons = document.querySelectorAll('.pagination .btn');
            buttons.forEach(button => {
                button.classList.remove('active');
                if (parseInt(button.textContent) === currentPage) {
                    button.classList.add('active');
```

```
        }
    });
}
function changePage(page) {
    currentPage = page;
    renderGridPage();
    updatePagination();
}
// Функция подсветки изменений
function highlightChange(element, newValue, previousValue) {
    element.classList.remove('increase', 'decrease');
    if (newValue > previousValue) {
        element.classList.add('increase');
    } else if (newValue < previousValue) {
        element.classList.add('decrease');
    }
}

// Функция отображения таблицы
function renderTable() {
    const tableBody = document.querySelector("#dataTable tbody");
    tableBody.innerHTML = '';

    filteredData.forEach(row => {
        const newRow = document.createElement('tr');
        newRow.innerHTML = `
            <td>${row.ticker}</td>
            <td>${row.shortname}</td>
            <td>${row.price_at_max}</td>
            <td>${Math.round(row.volume_at_max)}</td>
            <td>${row.price_change_5m}%</td>
            <td>${row.volume_change_5m}%</td>
            <td>${row.price_change_10m}%</td>
            <td>${Math.round(row.volume_change_10m)}%</td>
            <td>${row.price_change_15m}%</td>
            <td>${Math.round(row.volume_change_15m)}%</td>
            <td>${row.price_change_30m}%</td>
            <td>${Math.round(row.volume_change_30m)}%</td>
            <td>${row.price_change_1h}%</td>
            <td>${Math.round(row.volume_change_60m)}%</td>
        `;
        tableBody.appendChild(newRow);

        // Подсветка изменений
        const columnsToHighlight = [
            { index: 2, key: 'price_at_max' },
            { index: 3, key: 'volume_at_max' },
            { index: 4, key: 'price_change_5m' },
            { index: 5, key: 'volume_change_5m' },
            { index: 6, key: 'price_change_10m' },
            { index: 7, key: 'volume_change_10m' },
            { index: 8, key: 'price_change_15m' },
            { index: 9, key: 'volume_change_15m' },
            { index: 10, key: 'price_change_30m' },
            { index: 11, key: 'volume_change_30m' },
            { index: 12, key: 'price_change_1h' },
            { index: 13, key: 'volume_change_60m' },
        ];

        columnsToHighlight.forEach(col => {
            const cell = newRow.children[col.index];
```

```
                    const newValue = row[col.key];
                    const previousValue = previousData[row.ticker]?.[col.key]
|| 0;
                    highlightChange(cell, newValue, previousValue);
                });

                // Обновляем предыдущие данные
                previousData[row.ticker] = { ...row };
            });
        }

        // Функция отображения текущей страницы графиков
        function renderGridPage() {
            const startIndex = (currentPage - 1) * itemsPerPage;
            const endIndex = startIndex + itemsPerPage;
            const pageData = filteredData.slice(startIndex, endIndex);
            renderGrid(pageData);
        }

        // Функция отображения графиков
        function renderGrid(data) {
            const gridContainer = document.getElementById('gridContainer');
            gridContainer.innerHTML = '';

            data.forEach(row => {
                const gridItem = document.createElement('div');
                gridItem.className = 'grid-item';

                const chartId = `chart-${row.ticker}`;
                gridItem.innerHTML = `
                    <div class="chart-container">
                        <div class="chart-title">${row.ticker}</div>
                        <div class="chart" id="${chartId}"></div>
                    </div>
                    <div class="values-container">
                        <div data-tooltip="Изм. цены
(5м)">${row.price_change_5m}%</div>
                        <div data-tooltip="Изм. объема
(5м)">${Math.round(row.volume_change_5m)}%</div>
                        <div data-tooltip="Изм. цены
(15м)">${Math.round(row.price_change_15m)}%</div>
                        <div data-tooltip="Изм. объема
(15м)">${Math.round(row.volume_change_15m)}%</div>
                        <div data-tooltip="Изм. цены
(1ч)">${row.price_change_1h}%</div>
                        <div data-tooltip="Изм. объема
(1ч)">${Math.round(row.volume_change_60m)}%</div>
                    </div>
                `;

                gridContainer.appendChild(gridItem);

                // Добавляем тултипы к значениям
                const valueElements = gridItem.querySelectorAll('.values-
container div');
                valueElements.forEach(el => {
                    tippy(el, {
                        content: el.getAttribute('data-tooltip'),
                        placement: 'top',
                        animation: 'scale',
                    });
```

```
                    // Подсветка изменений в графическом виде
                    const key = el.getAttribute('data-tooltip');
                    let dataKey;
                    switch (key) {
                        case 'Изм. цены (5м)': dataKey = 'price_change_5m';
break;
                        case 'Изм. объема (5м)': dataKey =
'volume_change_5m'; break;
                        case 'Изм. цены (15м)': dataKey = 'price_change_15m';
break;
                        case 'Изм. объема (15м)': dataKey =
'volume_change_15m'; break;
                        case 'Изм. цены (1ч)': dataKey = 'price_change_1h';
break;
                        case 'Изм. объема (1ч)': dataKey =
'volume_change_60m'; break;
                    }
                    const newValue = row[dataKey];
                    const previousValue = previousData[row.ticker]?.[dataKey]
|| 0;
                    highlightChange(el, newValue, previousValue);
                });

                // Обновляем предыдущие данные
                previousData[row.ticker] = { ...row };

                // Отображение графика
                if (row.candles_data && row.candles_data.length > 0) {
                    const chartContainer =
gridItem.querySelector(`#${chartId}`);
                    const chart =
LightweightCharts.createChart(chartContainer, {
                        width: chartContainer.clientWidth,
                        height: 200,
                        layout: {
                            backgroundColor: '#ffffff',
                            textColor: '#000000',
                        },
                        grid: {
                            vertLines: {
                                color: '#eeeeee',
                            },
                            horzLines: {
                                color: '#eeeeee',
                            },
                        },
                        priceScale: {
                            borderColor: '#cccccc',
                        },
                        timeScale: {
                            borderColor: '#cccccc',
                            visible: true,
                        },
                        handleScroll: false,
                        handleScale: false,
                    });

                    const candleSeries = chart.addCandlestickSeries();
                    const candlesData = row.candles_data.slice(-
60).map(candle => ({
```

```
                        time: candle.begin,
                        open: candle.open,
                        high: candle.high,
                        low: candle.low,
                        close: candle.close
                    }));

                    candleSeries.setData(candlesData);
                } else {
                    console.warn(`Нет данных для отображения графика для
тикера ${row.ticker}`);
                }
            });
        }

        // Индикатор загрузки
        const loadingIndicator = document.createElement('div');
        loadingIndicator.id = 'loadingIndicator';
        loadingIndicator.style.position = 'fixed';
        loadingIndicator.style.top = '50%';
        loadingIndicator.style.left = '50%';
        loadingIndicator.style.transform = 'translate(-50%, -50%)';
        loadingIndicator.style.fontSize = '1.5em';
        loadingIndicator.style.color = '#007bff';
        loadingIndicator.innerHTML = 'Загрузка данных...';
        document.body.appendChild(loadingIndicator);

        // Инициализация приложения
        function initApp() {
            connectWebSocket();
        }

        // Запуск приложения
        initApp();
    </script>
</body>
</html>
```