Министерство образования Республики Беларусь

Учреждение образования
Белорусский государственный университет
информатики и радиоэлектроники

Факультет компьютерных систем и сетей
Кафедра информатики
Специальность «ИиТП»

Пояснительная записка к курсовому проекту на тему “Акционная биржа”

Выполнил:
студент группы 753503
Кознев Н. Д.

Руководитель:
Удовин И.А.

Минск 2020

Содержание

Введение

Фондовый рынок – это сегмент всеобщей капитализации, который позволяет торговать акциями в биржевом и во внебиржевом формате. В совокупности с финансово-кредитным рынком этот внушительный сектор капитала формирует мировую экономику. Многие начинающие компании не имеют в своём распоряжении больших капиталов, а развитие прогрессивных идей и технологий требует серьезных денег. В этом случае перед стартом бизнес-проекта проводится выпуск ценных бумаг – акций, которые приобретаются инвесторами в надежде на рост стоимости и ежегодные дивиденды.

Зачастую на определённом этапе развития коммерческое предприятие также выпускает первичные акции или предлагает к продаже дополнительные объемы акционных активов. Во всех случаях акции и другие ценные бумаги продаются на фондовом рынке через централизованные биржи или автономных брокеров.

Трейдер — торговец, действующий по собственной инициативе и стремящийся извлечь прибыль непосредственно из процесса торговли. Обычно подразумевается торговля ценными бумагами (акциями, облигациями, фьючерсами, опционами) на фондовой бирже. Трейдерами также называют торговцев на валютном (форекс) и товарном рынках: торговля осуществляется трейдером как на биржевом, так и на внебиржевом рынках.

Не следует путать трейдера с другими торговцами, которые проводят сделки по заявкам клиентов или от их имени: брокер, дилер, дистрибьютор.

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

1. Основные функциональные и нефункциональные требования

1.1 Назначение разработки

Начинающий трейдер, как правило, имеет довольно смутные, и чаще всего, в корне неверные представления о рынке и о трейдинге в целом. Для него рынок – хаотичная субстанция, которую невозможно понимать, предугадывать и систематизировать. Часто именно новички употребляют словосочетание «игра на бирже», подразумевая, что получить прибыль здесь можно лишь волей случая и удачи, как при игре в рулетку.

Поэтому, в первую очередь новичку необходимо изменить свой взгляд на рынок и чтобы это реализовать было решено разработать минимальный функционал который познакомит с трейдингом.

1.2 Перечень основных выполняемых функций

Как и любая современная торговая биржа система должна выполнять следующие функции:

- Предоставление доступа к системе, путем регистрирования пользователя внутри ее
- Предоставление личного кабинета для просмотра состояния профиля
- Предоставление фьючерсов акций для покупки
- Получение актуальных сведений о состоянии стоимости

1.3 Входные и выходные данные пользователя

Пользователь должен иметь возможность зайти в систему используя уникальный никнейм и пароль, который должен соответствовать определенным требованиям (длина более 8 символов, символы английского алфавита и цифры из которых обязательно должно быть: хотя бы 1 буква в верхнем регистре, 1 в нижнем, 1 цифра)

1.4 Требования к надежности

Пользователь не должен иметь возможность напрямую влиять на стоимость фьючерсов, а также не должен иметь возможность изменять все данные системы которые с ним не взаимодействуют.

1.5 Требования к организации системы

Система должна состоять из мало зависимых подсистем (отдельных приложений), которые обеспечивают работу всей системы в целом. Данные подсистемы должны реализовывать свой функционал и развиться как самостоятельные подсистемы.

В качестве архитектурного решения должно быть применено классическое Django-решение: паттерн MVT (Model-View-Template).
Основные элементы паттерна:

- URL dispatcher: при получение запроса на основании запрошенного адреса URL определяет, какой ресурс должен обрабатывать данный запрос.
- View: получает запрос, обрабатывает его и отправляет в ответ пользователю некоторый ответ. Если для обработки запроса необходимо обращение к модели и базе данных, то View взаимодействует с ними. Для создания ответа может применять Template или шаблоны. В архитектуре MVC этому компоненту соответствуют контроллеры (но не представления).
- Model: описывает данные, используемые в приложении. Отдельные классы, как правило, соответствуют таблицам в базе данных.
- Template: представляет логику представления в виде сгенерированной разметки html. В MVC этому компоненту соответствует View, то есть представления.

Когда к приложению приходит запрос, то URL dispatcher определяет, с каким ресурсом сопоставляется данный запрос и передает этот запрос выбранному ресурсу. Ресурс фактически представляет функцию или View, который получает запрос и определенным образом обрабатывает его. В процессе обработки View может обращаться к моделям и базе данных, получать из нее данные, или, наоборот, сохранять в нее данные. Результат обработки запроса отправляется обратно, и этот результат пользователь видит в своем браузере. Как правило, результат обработки запроса представляет сгенерированный html-код, для генерации которого применяются шаблоны (Template).

1.6 Требования к UI/UX

Интерфейс приложения должен быть разработан в программе figma, должны использоваться основные принципы дизайны. Интерфейс должен быть понятен пользователю.

Figma — онлайн-сервис для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени. Сервис имеет широкие возможности для интеграции с корпоративным мессенджером Slack и инструментом для высокоуровневого прототипирования Framer. Позиционируется создателями как основной конкурент программным продуктам компании Adobe.

Сервис доступен по подписке, предусмотрен бесплатный тарифный план для одного пользователя. Ключевой особенностью Figma является её облачность. Также Figma имеет оффлайн-версии для различных платформ (Windows, macOS, Linux). За счёт этого также достигается принцип кроссплатформенности, который не могут гарантировать ближайшие конкуренты — Sketch и Adobe XD.

1.7 Обоснование выбора языка программирования и среды разработки

В качестве языка программирования для разработки был выбран Python3.

Python — активно развивающийся язык программирования, новые версии с добавлением/изменением языковых свойств выходят примерно раз в два с половиной года. Язык не подвергался официальной стандартизации, роль стандарта де-факто выполняет CPython, разрабатываемый под контролем автора языка. В настоящий момент Python занимает второе место в рейтинге TIOBE с показателем 12,12 %. Аналитики отмечают, что это самый высокий балл Python за все время его присутствия в рейтинге.

В качестве среды разработки был выбран Django3.

Django — свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Проект поддерживается организацией Django Software Foundation.

Django проектировался для работы под управлением Apache с модулем mod python и с использованием PostgreSQL в качестве базы данных.

С включением поддержки WSGI, Django может работать под управлением FastCGI, mod wsgi, или SCGI на Apache и других серверах (lighttpd, nginx,…), сервера uWSGI.

2. Моделирование предметной области

2.1 Обзор используемых технологий

2.1.1 Python 3

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает структурное, обобщенное, объектно-ориентированное, функциональное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ. Он распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные. Есть реализация интерпретатора для JVM с возможностью компиляции, CLR, LLVM, другие независимые реализации. Проект PyPy использует JIT-компиляцию, которая значительно увеличивает скорость выполнения Python-программ.

Python — активно развивающийся язык программирования, новые версии с добавлением/изменением языковых свойств выходят примерно раз в два с половиной года. Язык не подвергался официальной стандартизации, роль стандарта де-факто выполняет CPython, разрабатываемый под контролем автора языка. В настоящий момент Python занимает третье место в рейтинге TIOBE с показателем 10,2 %. Аналитики отмечают, что это самый высокий балл Python за все время его присутствия в рейтинге.

Python — стабильный и распространённый язык. Он используется во многих проектах и в различных качествах: как основной язык программирования или для создания расширений и интеграции приложений. На Python реализовано большое количество проектов, также он активно используется для создания прототипов будущих программ. Python используется во многих крупных компаниях: Dropbox, Google (например некоторые части Youtube и Youtube API написаны на Python), Facebook, Instagram.

Python с пакетами NumPy, SciPy и MatPlotLib активно используется как универсальная среда для научных расчётов в качестве замены распространённым специализированным коммерческим пакетам Matlab, IDL и другим. Библиотека Astropy — популярный инструмент для астрономических расчётов.

Сочетание простоты и лаконичности с возможностью использования сложных абстракций и мощных разнообразных инструментов делает Python удобным в качестве скриптового языка. Возможность его встраивания ограничивается объёмом интерпретатора, но в крупных системах это ограничение несущественно. В профессиональных программах трехмерной графики, таких как Autodesk Maya, Blender, Houdini и Nuke, Python используется для расширения стандартных возможностей программ. В Microsoft Power BI Desktop Python, наряду со встроенными языками запросов и языком программирования R, может использоваться на этапе загрузки данных в ETL-процессах, расчётах и графической визуализации данных.

Также Python подходит для выполнения нестандартных или сложных задач в системах сборки проектов, что обусловлено отсутствием необходимости предварительной компиляции исходных файлов. В проекте Google Test он используется для генерации исходного кода mock-объектов для классов языка C++.

Интерпретатор Python может использоваться в качестве мощной командной оболочки и скриптового языка для написания командных файлов ОС. Легкость обращения из Python-скриптов к внешним программам и наличие библиотек, дающих доступ к управлению системой, делают Python удобным инструментом для системного администрирования. Он широко используется для этой цели на платформе Linux: обычно Python поставляется с системой, во многих дистрибутивах инсталляторы и визуальный интерфейс системных утилит написаны именно на Python. Используется он и в администрировании других Unix-систем, в частности, в Solaris и macOS. Кросс-платформенность самого языка и библиотек делает его привлекательным для унифицированной автоматизации задач системного администрирования в гетерогенных средах, где совместно применяются компьютеры с операционными системами различных типов.

Dec 2020	Dec 2019	Change	Programming Language	Ratings	Change
1	2	^	C	16.48%	+0.40%
2	1	v	Java	12.53%	-4.72%
3	3		Python	12.21%	+1.90%
4	4		C++	6.91%	+0.71%
5	5		C#	4.20%	-0.60%
6	6		Visual Basic	3.92%	-0.83%
7	7		JavaScript	2.35%	+0.26%
8	8		PHP	2.12%	+0.07%
9	16	^^	R	1.60%	+0.60%
10	9	v	SQL	1.53%	-0.31%

Рисунок 2.1.1.1 — Рейтинг языков программирования TIOBE index

2.1.2 Django 3

Django — свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Проект поддерживается организацией Django Software Foundation.

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других (например, Ruby on Rails). Один из основных принципов фреймворка — DRY (англ. *Don't repeat yourself*)

Также, в отличие от других фреймворков, обработчики URL в Django конфигурируются явно при помощи регулярных выражений.

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

Веб-фреймворк Django используется в таких крупных и известных сайтах, как Instagram, Disqus, Mozilla, The Washington Times, Pinterest, YouTube, Google и др.

Также Django используется в качестве веб-компонента в различных проектах, таких как Graphite — система построения графиков и наблюдения, FreeNAS — свободная реализация системы хранения и обмена файлами и др.

Архитектура Django похожа на «Модель-Представление-Контроллер» (MVC). Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представление (англ. *View*), а презентационная логика Представления реализуется в Django уровнем Шаблонов (англ. *Template*). Из-за этого уровневую архитектуру Django часто называют «Модель-Шаблон-Представление» (MTV).

Django проектировался для работы под управлением Apache с модулем mod python и с использованием PostgreSQL в качестве базы данных.
С включением поддержки WSGI, Django может работать под управлением FastCGI, mod wsgi, или SCGI на Apache и других серверах (lighttpd, nginx,…), сервера uWSGI.

В настоящее время, помимо базы данных PostgreSQL, Django может работать с другими СУБД: MySQL, SQLite, Microsoft SQL Server, DB2, Firebird, SQL Anywhere и Oracle.

В составе Django присутствует собственный веб-сервер для разработки. Сервер автоматически определяет изменения в файлах исходного кода проекта и перезапускается, что ускоряет процесс разработки на Python. Но при этом он работает в однопоточном режиме и пригоден только для процесса разработки и отладки приложения.

Первоначальная разработка Django как средства для работы новостных ресурсов достаточно сильно отразилась на его архитектуре: он предоставляет ряд средств, которые помогают в быстрой разработке веб-сайтов информационного характера. Так, например, разработчику не требуется

создавать контроллеры и страницы для административной части сайта, в Django есть встроенное приложение для управления содержимым, которое можно включить в любой сайт, сделанный на Django, и которое может управлять сразу несколькими сайтами на одном сервере. Административное приложение позволяет создавать, изменять и удалять любые объекты наполнения сайта, протоколирует все совершенные действия, и предоставляет интерфейс для управления пользователями и группами (с объектным назначением прав).

В дистрибутив Django также включены приложения для системы комментариев, синдикации RSS и Atom, «статических страниц» (которыми можно управлять без необходимости писать контроллеры и представления), перенаправления URL и другое.

2.1.3 Figma

Figma — онлайн-сервис для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени. Сервис имеет широкие возможности для интеграции с корпоративным мессенджером Slack и инструментом для высокоуровневого прототипирования Framer. Позиционируется создателями как основной конкурент программным продуктам компании Adobe.

Сервис доступен по подписке, предусмотрен бесплатный тарифный план для одного пользователя. Ключевой особенностью Figma является её облачность, у сервиса нет оффлайн-версии. За счет этого также достигается принцип кроссплатформенности, который не могут гарантировать ближайшие конкуренты — Sketch и Adobe XD.

Figma подходит как для создания простых прототипов и дизайн-систем, так и сложных проектов (мобильные приложения, порталы). В 2018 году платформа стала одним из самых быстро развивающихся инструментов для разработчиков и дизайнеров.

Устойчивая модель развития Figma, упростившая сотрудничество во всем процессе создания цифровых продуктов для дизайнеров, разработчиков, менеджеров и маркетологов, позволила в начале 2018 года привлечь дополнительные 25 миллионов долларов инвестиций от партнеров.

В 2019 году создатели редактора привлекли 40 миллионов долларов инвестиций от венчурного фонда Sequoia Capital при оценке самой компании в 400 миллионов долларов. Решение об инвестициях было принято на основе факта, что около половины портфельных компаний фонда используют редактор в работе. К началу 2019 года Figma вышла на 1 миллион зарегистрированных пользователей, став серьезным конкурентом для традиционных графических редакторов и средств прототипирования.

2.1.4 PostgreSQL

PostgreSQL — свободная объектно-реляционная система управления базами данных (СУБД).

Существует в реализациях для множества UNIX-подобных платформ, включая AIX, различные BSD-системы, HP-UX, IRIX, Linux, macOS, Solaris/OpenSolaris, Tru64, QNX, а также для Microsoft Windows.

Функции являются блоками кода, исполняемыми на сервере, а не на клиенте БД. Хотя они могут писаться на чистом SQL, реализация дополнительной логики, например, условных переходов и циклов, выходит за рамки SQL и требует использования некоторых языковых расширений. Функции могут писаться с использованием одного из следующих языков:

- Встроенный процедурный язык PL/pgSQL, во многом аналогичный языку PL/SQL, используемому в СУБД Oracle;
- Скриптовые языки — PL/Lua, PL/LOLCODE, PL/Perl, PL/PHP, PL/Python, PL/Ruby, PL/sh, PL/Tcl, PL/Scheme, PL/v8 (Javascript);
- Классические языки — C, C++, Java (через модуль PL/Java);
- Статистический язык R (через модуль PL/R).

PostgreSQL допускает использование функций, возвращающих набор записей, который далее можно использовать так же, как и результат выполнения обычного запроса.

Функции могут выполняться как с правами их создателя, так и с правами текущего пользователя.

Иногда функции отождествляются с хранимыми процедурами, однако между этими понятиями есть различие. С девятой версии возможно написание автономных блоков, которые позволяют выполнять код на процедурных языках без написания функций, непосредственно в клиенте.

Механизм правил представляет собой механизм создания пользовательских обработчиков не только DML-операций, но и операции выборки. Основное отличие от механизма триггеров заключается в том, что правила срабатывают на этапе разбора запроса, до выбора оптимального плана выполнения и самого процесса выполнения. Правила позволяют переопределять поведение системы при выполнении SQL-операции к таблице. Хорошим примером является реализация механизма представлений (англ. *views*): при создании представления создается правило, которое определяет, что вместо выполнения операции выборки к представлению система должна выполнять операцию выборки к базовой таблице/таблицам с учетом условий выборки, лежащих в основе определения представления. Для создания представлений, поддерживающих операции обновления, правила для операций вставки, изменения и удаления строк должны быть определены пользователем.

2.1.5 Git / GitHub

Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.
Программа является свободной и выпущена под лицензией GNU GPL версии 2. По умолчанию используется TCP порт 9418.

Система спроектирована как набор программ, специально разработанных с учетом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Например, Cogito является именно таким примером оболочки к репозиториям Git, а StGit использует Git для управления коллекцией исправлений (патчей).

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удаленный доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и дает возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранятся операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы. Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создает в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «.git» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создается рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда commit).

Нижний уровень git является так называемой контентно-адресуемой файловой системой. Инструмент командной строки git содержит ряд команд по непосредственной манипуляции этим репозиторием на низком уровне. Эти команды не нужны при нормальной работе с git как с системой контроля версий, но нужны для реализации сложных операций (ремонт повреждённого репозитория и так далее), а также дают возможность создать на базе репозитория git свое приложение.

Для каждого объекта в репозитории вычисляется SHA-1-хеш, и именно он становится именем файла, содержащего данный объект в каталоге .git/objects. Для оптимизации работы с файловыми системами, не использующими деревья для каталогов, первый байт хеша становится именем подкаталога, а остальные — именем файла в нём, что снижает количество файлов в одном каталоге (ограничивающий фактор производительности на таких устаревших файловых системах).

Все ссылки на объекты репозитория, включая ссылки на один объект, находящийся внутри другого объекта, являются SHA-1-хешами.

Кроме того, в репозитории существует каталог refs, который позволяет задать читаемые человеком имена для каких-то объектов Git. В командах Git оба вида ссылок — читаемые человеком из refs, и нижележащие SHA-1 — полностью взаимозаменяемы.

В классическом обычном сценарии в репозитории git есть три типа объектов — файл, дерево и «коммит». Файл есть какая-то версия какого-то пользовательского файла, дерево — совокупность файлов из разных подкаталогов, «коммит» — дерево и некая дополнительная информация (например, родительские коммиты, а также комментарий).

В репозитории иногда производится сборка мусора, во время которой устаревшие файлы заменяются на «дельты» между ними и актуальными файлами (то есть, актуальная версия файла хранится не инкрементально, инкременты используются только для возврата к предыдущим версиям), после чего данные «дельты» складываются в один большой файл, к которому строится индекс. Это снижает требования по емкости хранения.

Репозиторий Git бывает локальный и удаленный. Локальный репозиторий — это подкаталог .git, создаётся (в пустом виде) командой git init и (в непустом

виде с немедленным копированием содержимого родительского удалённого репозитория и простановкой ссылки на родителя) командой git clone.

Практически все обычные операции с системой контроля версий, такие, как коммит и слияние, производятся только с локальным репозиторием. Удалённый репозиторий можно только синхронизировать с локальным как «вверх» (*push*), так и «вниз» (*pull*).

Наличие полностью всего репозитория проекта локально у каждого разработчика даёт Git ряд преимуществ перед SVN. Так, например, все операции, кроме push и pull, можно осуществлять без наличия интернет-соединения.

Очень мощной возможностью git являются ветви, реализованные куда более полно, чем в SVN: по сути, ветвь git есть не более чем именованная ссылка, указывающая на некий коммит в репозитории (используется подкаталог refs). Коммит без создания новой ветви всего лишь передвигает эту ссылку на себя, а коммит с созданием ветви — оставляет старую ссылку на месте, но создает новую на новый коммит, и объявляет её текущей. Заменить локальные девелоперские файлы на набор файлов из иной ветви, тем самым перейдя к работе с ней — так же тривиально.

Также поддерживаются субрепозитории с синхронизацией текущих ветвей в них.

Команда push передает все новые данные (те, которых еще нет в удалённом репозитории) из локального репозитория в репозиторий удаленный. Для исполнения этой команды необходимо, чтобы удалённый репозиторий не имел новых коммитов в себя от других клиентов, иначе push завершается ошибкой, и придётся делать pull и слияние.

Команда pull — обратна команде push. В случае, если одна и та же ветвь имеет независимую историю в локальной и в удаленной копии, pull немедленно переходит к слиянию.

Слияние в пределах разных файлов осуществляется автоматически (всё это поведение настраивается), а в пределах одного файла — стандартным двухпанельным сравнением файлов. После слияния нужно объявить конфликты как разрешенные.

Результатом всего этого является новое состояние в локальных файлах у того разработчика, что осуществил слияние. Ему нужно немедленно сделать коммит, при этом в данном объекте коммита в репозитории окажется информация о том, что коммит есть результат слияния двух ветвей и имеет два родительских коммита.

Кроме слияния, Git поддерживает ещё операцию перемещения. Эта операция есть получение набора всех изменений в ветви A, с последующим их «накатом» на ветвь B. В результате ветвь B продвигается до состояния AB. В отличие от слияния, в истории ветви AB не останется никаких промежуточных коммитов ветви A (только история ветви B и запись о самом rebase, это упрощает интеграцию крупных и очень крупных проектов).

Также Git имеет временный локальный индекс файлов. Это — промежуточное хранилище между собственно файлами и очередным коммитом (коммит делается только из этого индекса). С помощью этого индекса осуществляется добавление новых файлов (git add добавляет их в индекс, они попадут в следующий коммит), а также коммит не всех измененных файлов (коммит делается только тем файлам, которым был сделан git add). После git add можно редактировать файл далее, получатся три копии одного и того же файла — последняя, в индексе (та, что была на момент git add), и в последнем коммите.

Имя ветви по умолчанию: master. Имя удалённого репозитория по умолчанию, создаваемое git clone во время типичной операции «взять имеющийся проект с сервера себе на машину»: origin.

Таким образом, в локальном репозитории всегда есть ветвь master, которая есть последний локальный коммит, и ветвь origin/master, которая есть последнее состояние удаленного репозитория на момент завершения и
сполнения последней команды pull или push.
Команда fetch (частичный pull) — берёт с удалённого сервера все изменения в origin/master, и переписывает их в локальный репозиторий, продвигая метку origin/master.

Если после этого master и origin/master разошлись в стороны, то необходимо сделать слияние, установив master на результат слияния (команда pull есть fetch+merge). Далее возможно сделать push, отправив результат слияния на сервер и установив на него origin/master.

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

Веб-сервис основан на системе контроля версий Git и разработан на Ruby on Rails и Erlang компанией GitHub, Inc (ранее Logical Awesome). Сервис бесплатен для проектов с открытым исходным кодом и (с 2019 года) небольших частных проектов, предоставляя им все возможности (включая SSL), а для крупных корпоративных проектов предлагаются различные платные тарифные планы.

Слоган сервиса — «Social Coding» — на русский можно перевести как «Пишем код вместе». На футболках же печатают совсем другую фразу: «Fork you!». С одной стороны, она созвучна с англоязычным ругательством и намекает на неформальную атмосферу. С другой, эти слова напоминают, что создавать новые форки с Git можно легко и безболезненно — традиционно, к созданию веток разработчики проектов с открытым исходным кодом относятся негативно — а также созвучна названию одной из возможностей GitHub — очереди форков.
создатели сайта называют GitHub «социальной сетью для разработчиков».

Кроме размещения кода, участники могут общаться, комментировать правки друг друга, а также следить за новостями знакомых.

С помощью широких возможностей Git программисты могут объединять свои репозитории — GitHub предлагает удобный интерфейс для этого и может отражать вклад каждого участника в виде дерева.

Для проектов есть личные страницы, небольшие Вики и система отслеживания ошибок.

Прямо на сайте можно просмотреть файлы проектов с подсветкой синтаксиса для большинства языков программирования.

- Можно создавать приватные репозитории, которые будут видны только вам и выбранным вами людям.

  Раньше возможность создавать приватные репозитории была платной.
- Есть возможность прямого добавления новых файлов в свой репозиторий через веб-интерфейс сервиса.
- Код проектов можно не только скопировать через Git, но и скачать в виде обычных архивов с сайта.
- Кроме Git, сервис поддерживает получение и редактирование кода через SVN и Mercurial.
- На сайте есть pastebin-сервис gist.github.com для быстрой публикации фрагментов кода.

Ранее Ruby-проекты могли быть автоматически опубликованы в RubyGems-репозитории сервиса, но в октябре 2009 GitHub отказался от этого сервиса.

2.1.6 JavaScript

JavaScript — мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией стандарта ECMAScript (стандарт ECMA-262).

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

JavaScript является объектно-ориентированным языком, но используемое в языке прототипирование обуславливает отличия в работе с объектами по сравнению с традиционными класс-ориентированными языками. Кроме того, JavaScript имеет ряд свойств, присущих функциональным языкам — функции как объекты первого класса, объекты как списки, карринг, анонимные функции, замыкания — что придаёт языку дополнительную гибкость.

Несмотря на схожий с Си синтаксис, JavaScript по сравнению с языком Си имеет коренные отличия:

- объекты с возможностью интроспекции;
- функции как объекты первого класса;
- автоматическое приведение типов;
- автоматическая сборка мусора;
- анонимные функции.

В языке отсутствуют такие полезные вещи, как:

- стандартная библиотека: в частности, отсутствует интерфейс программирования приложений по работе с файловой системой, управлению потоками ввода-вывода, базовых типов для бинарных данных;
- стандартные интерфейсы к веб-серверам и базам данных;
- система управления пакетами, которая бы отслеживала зависимости и автоматически устанавливала их.

2.1.7 HTML5 / CSS3

HTML5 — язык для структурирования и представления содержимого всемирной паутины. Это пятая версия HTML. Хотя стандарт был завершён (рекомендованная версия к использованию) только в 2014 году (предыдущая, четвёртая, версия опубликована в 1999 году), уже с 2013 года браузерами оперативно осуществлялась поддержка, а разработчиками — использование рабочего стандарта. Цель разработки HTML5 — улучшение уровня поддержки мультимедиа-технологий с одновременным сохранением обратной совместимости, удобочитаемости кода для человека и простоты анализа для парсеров.

Во всемирной паутине долгое время использовались стандарты HTML 4.01, XHTML 1.0 и XHTML 1.1. Веб-страницы на практике оказывались сверстаны с использованием смеси особенностей, представленных различными спецификациями, включая спецификации программных продуктов, например веб-браузеров, а также сложившихся общеупотребительных приёмов. HTML5 был создан как единый язык разметки, который мог бы сочетать синтаксические нормы HTML и XHTML. Он расширяет, улучшает и рационализирует разметку документов, а также добавляет единый API для сложных веб-приложений.

В HTML5 реализовано множество новых синтаксических особенностей. Например, элементы <video>, <audio> и <canvas>, а также возможность использования SVG и математических формул. Эти новшества разработаны для упрощения создания и управления графическими и мультимедийными объектами в сети без необходимости использования сторонних API и плагинов. Другие новые элементы, такие как <section>, <article>, <header> и <nav>, разработаны для того, чтобы обогащать семантическое содержимое документа (страницы). Новые атрибуты были введены с той же целью, хотя ряд элементов и атрибутов был удален. Некоторые элементы, например <a>, <menu> и <cite>, были изменены, переопределены или стандартизированы. API и DOM стали основными частями спецификации HTML5. HTML5 также определяет некоторые особенности обработки ошибок верстки, поэтому синтаксические ошибки должны рассматриваться одинаково всеми совместимыми браузерами.

CSS — формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML, но может также применяться к любым XML-документам, например, к SVG или XUL.

2.2 Информация о базе данных

В качестве основной базы данных была выбрана PostgreSQL — свободная объектно-реляционная система управления базами данных (СУБД).

Ниже представлена схема таблиц, которые используются в проекте.

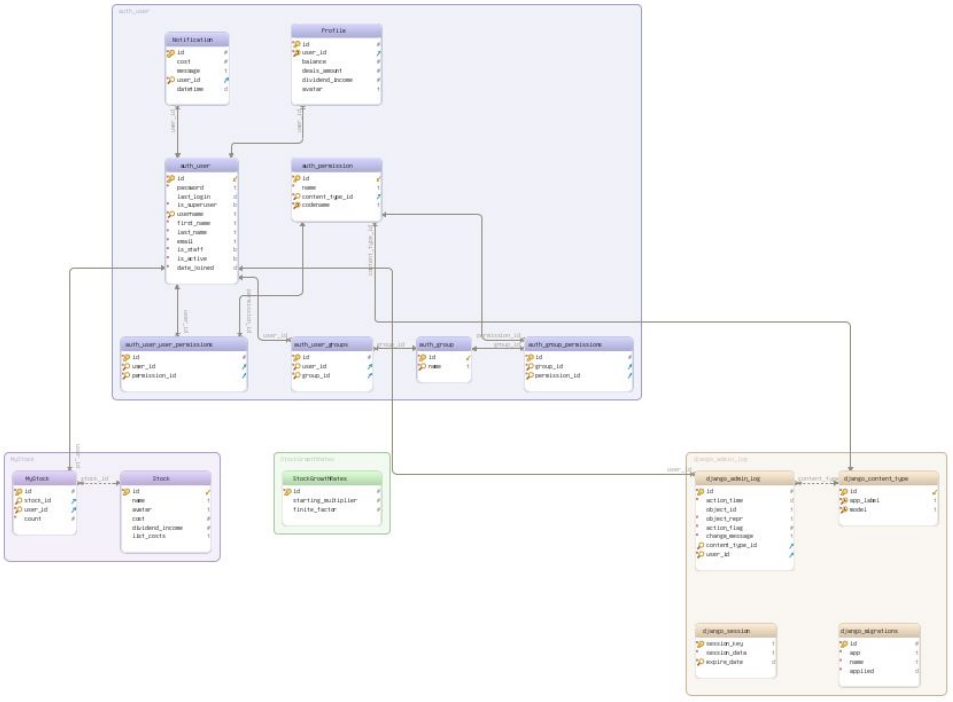

Рисунок 2.2.1 — Общая схема таблиц

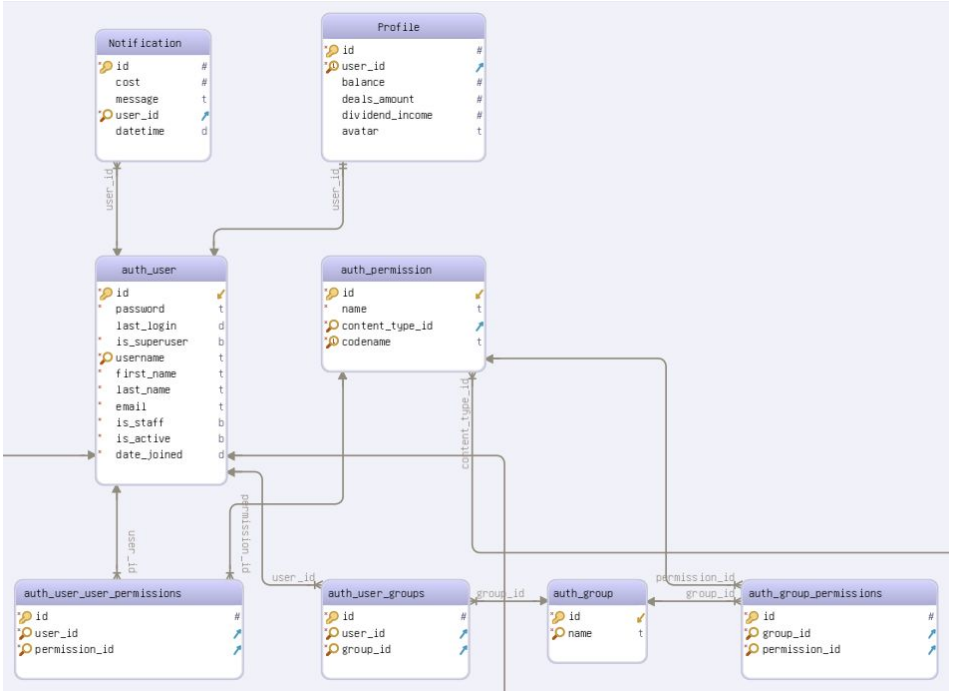

Рисунок 2.2.2 — Схема таблиц которые взаимодействуют с аккаунтом

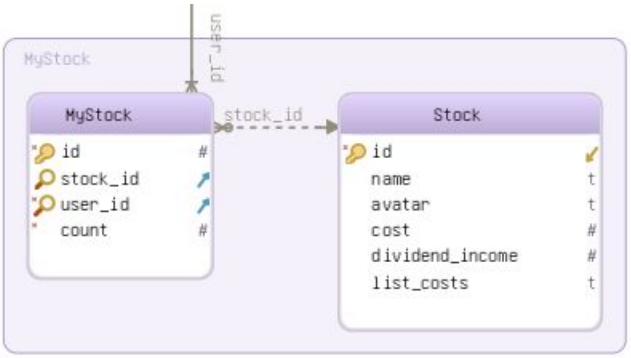

Рисунок 2.2.3 — Схема таблиц акций

Рисунок 2.2.4 — Таблица изменения акций

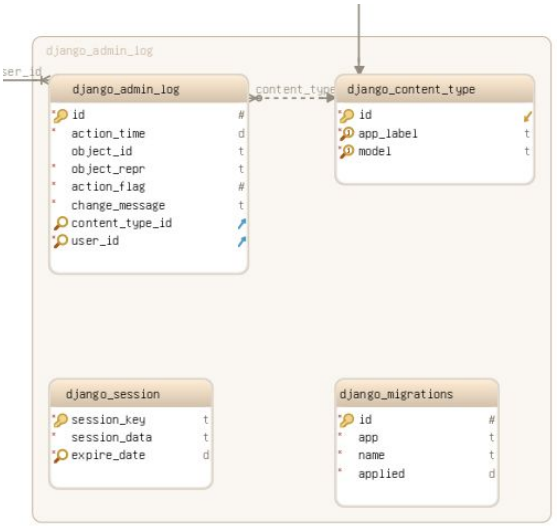

Рисунок 2.2.5 — Стандартные таблицы логирования

Table MyStock		
* Pk	id	integer
Idx	stock_id	integer
* Idx	user_id	integer
*	count	integer
Indexes		
Pk	MyStock_pkey	id
	MyStock_stock_id_5882d999	stock_id
	MyStock_user_id_bd8cecc4	user_id
Foreign Keys		
	MyStock_stock_id_5882d999_fk_Stock_id (stock_id) ref Stock (id)	
	MyStock_user_id_bd8cecc4_fk_auth_user_id (user_id) ref auth_user (id)	

Рисунок 2.2.6 — Таблица MyStock

Table Notification		
* Pk	id	integer
	cost	float8
	message	varchar(100)
* Idx	user_id	integer
	datetime	timestamptz(12)
Indexes		
Pk	Notification_pkey	id
	Notification_user_id_27901a99	user_id
Foreign Keys		
	Notification_user_id_27901a99_fk_auth_user_id (user_id) ref auth_user (id)	

Рисунок 2.2.7 — Таблица Notification

Table Profile		
* Pk	id	integer
* Unq	user_id	integer
	balance	float8
	deals_amount	integer
	dividend_income	float8
	avatar	varchar(100)
Indexes		
Pk	Users_pkey	id
Unq	Users_user_id_key	user_id
Foreign Keys		
	Users_user_id_d7df37b9_fk_auth_user_id (user_id) ref auth_user (id)	

Рисунок 2.2.8 — Таблица Profile

Table Stock		
* Pk	id	integer
	name	varchar(20)
	avatar	varchar(100)
	cost	float8
	dividend_income	float8
	list_costs	text
Indexes		
Pk	Stock_pkey	id

Рисунок 2.2.9 — Таблица Stock

Table StockGrowthRates		
* Pk	id	integer
	starting_multiplier	float8

Table StockGrowthRates		
	finite_factor	float8
Indexes		
Pk	StockGrowthRates_pkey	id

Рисунок 2.2.10 — Таблица StockGrowthRates

2.3 Информация о хранении данных

Для хранения основных данных использовалась база данных, для хранения графических изображение использовалось облачное хранилище Dropbox.

Dropbox — файловый хостинг компании Dropbox Inc., включающий персональное облачное хранилище, синхронизацию файлов и программу-клиент. Штаб-квартира компании расположена в Сан-Франциско.

Dropbox позволяет пользователям создать специальную папку на своих компьютерах, которую Dropbox синхронизирует таким образом, что она имеет одинаковое содержимое независимо от того, какое устройство используется для просмотра. Файлы, размещенные в этой папке, также доступны через веб-сайт Dropbox и мобильные приложения. Dropbox работает по модели Freemium, в которой пользователи имеют возможность создать бесплатный аккаунт с заданным количеством свободного пространства, в то время как для увеличения объёма аккаунта необходима платная подписка.

3. Реализация приложения

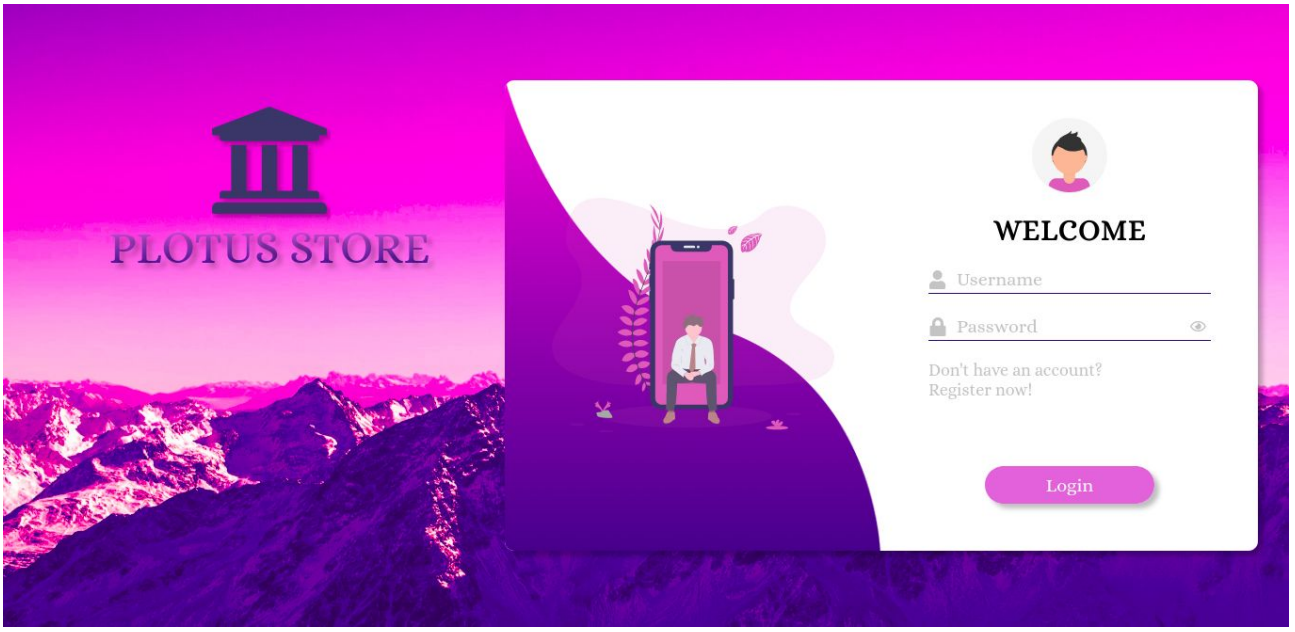

Рисунок 3.1 — Страница входа

Рисунок 3.2 — Страница регистрации

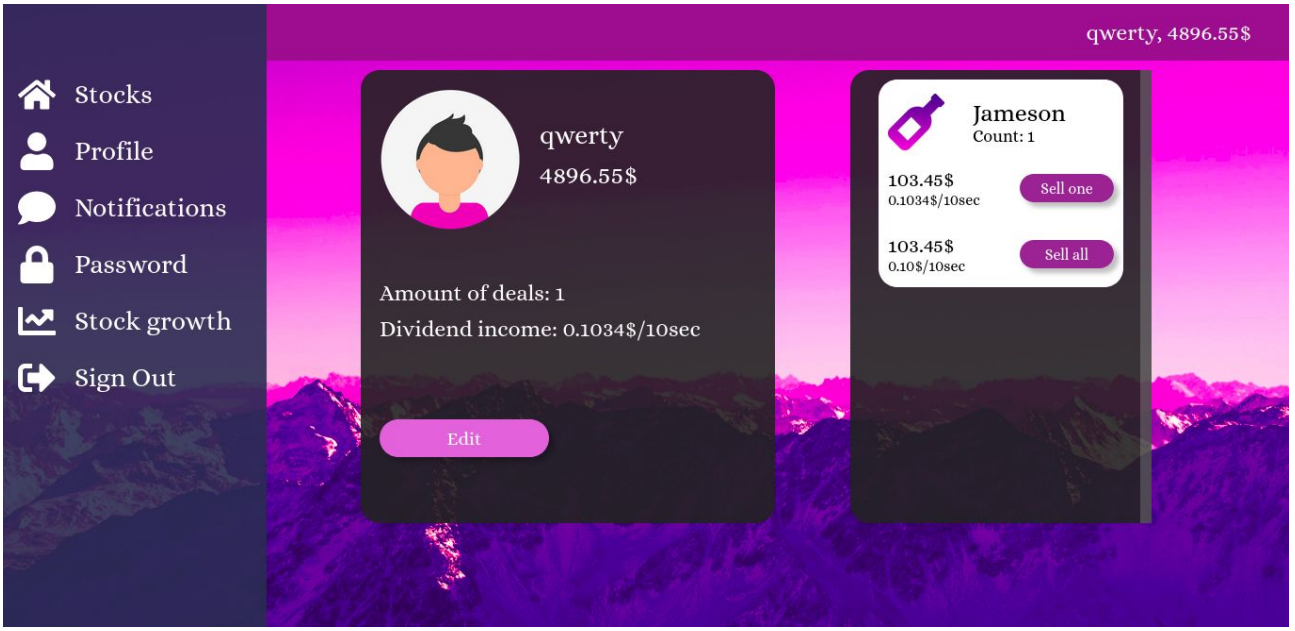

Рисунок 3.3 — Страница профиля

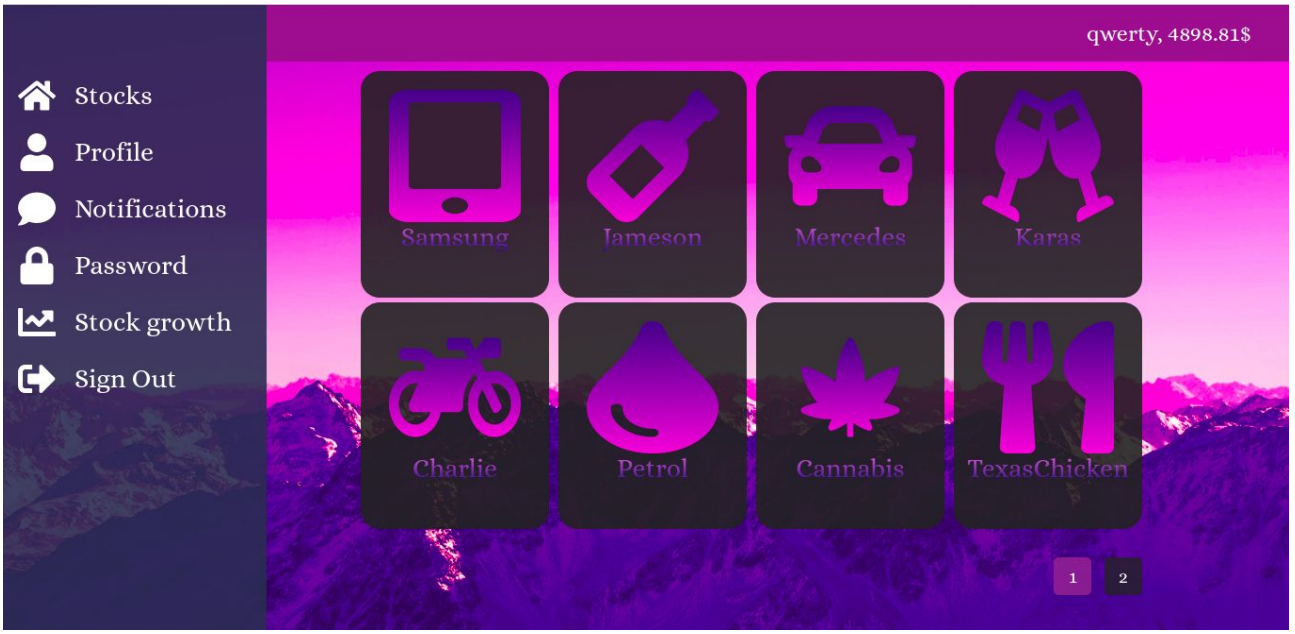

Рисунок 3.4 — Страница со фьючерсами на акции

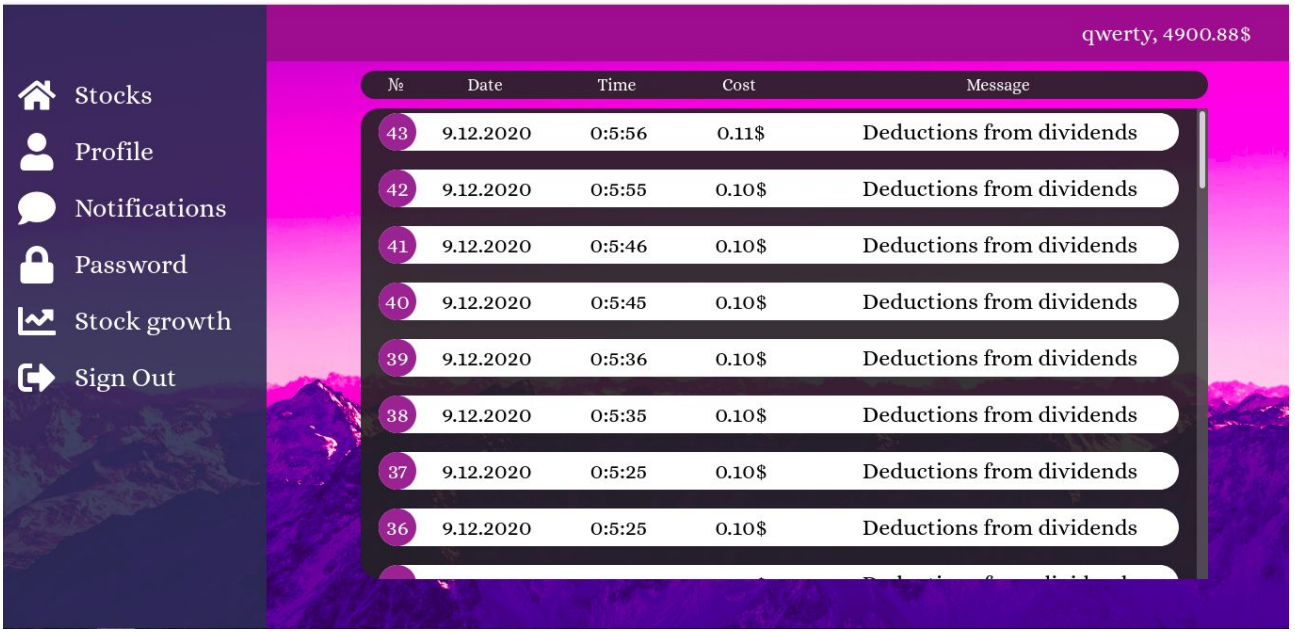

Рисунок 3.5 — Страница уведомлений

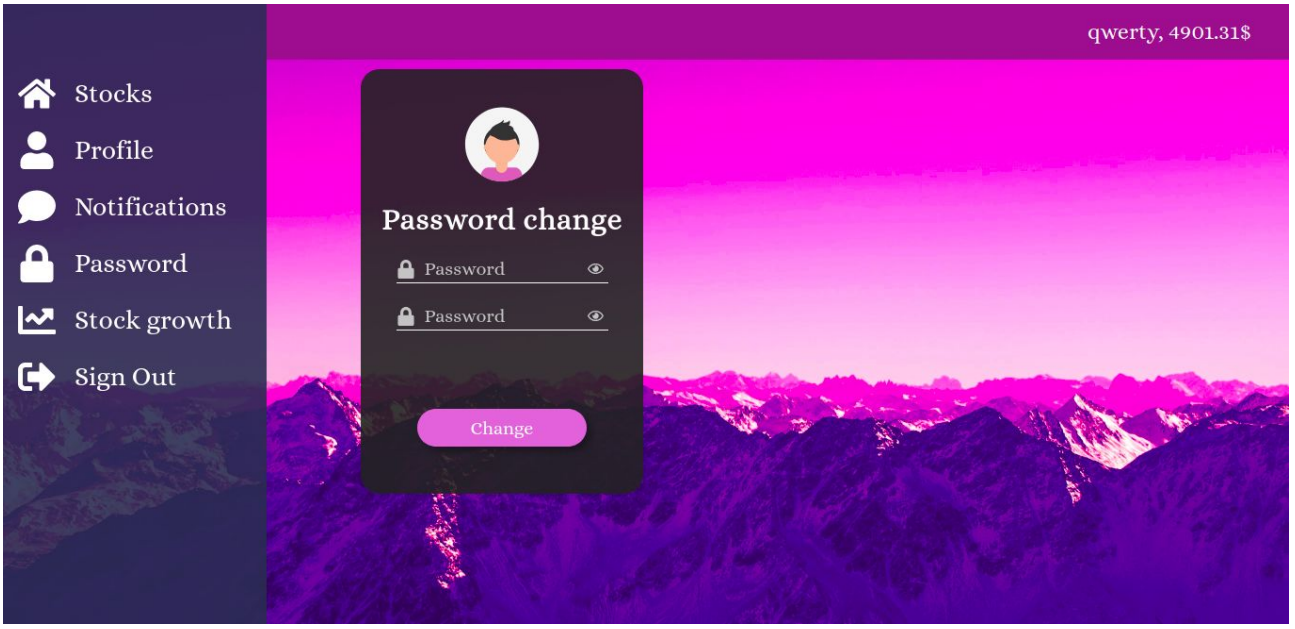

Рисунок 3.6 — Страница смены пароля

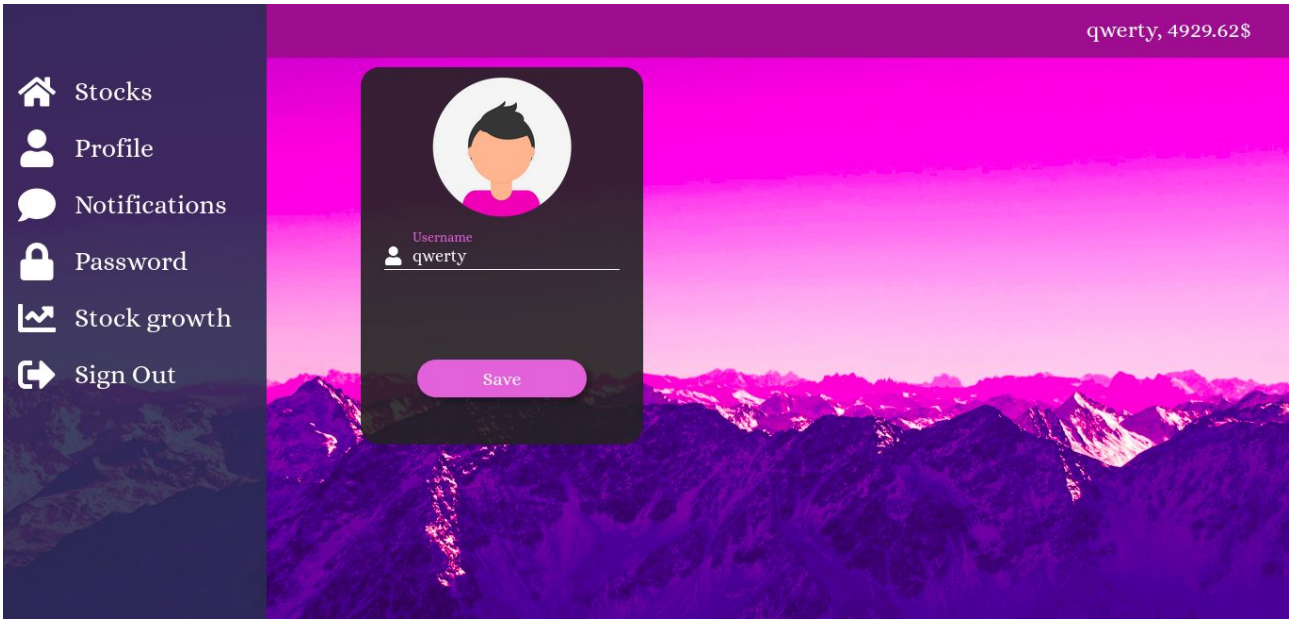

Рисунок 3.7 — Страница смены логина

Заключение

В ходе курсового проекта было разработано приложение, которое имитирует сервис фондового рынка, который в свою же очередь позволяет покупать фьючерсы на акции и получать с них дивиденды, что позволяет новичку в данной сфере понять основной принцип трейдинга.

Список используемых источников

1. Python 3 documentation — https://docs.python.org/3/
2. Django 3 documentation — https://docs.djangoproject.com/en/3.0/
3. Heroku deploy — https://developer.mozilla.org/ru/docs/Learn/Server-side/Django/%D0%A0%D0%B0%D0%B7%D0%B2%D0%BE%D1%80%D0%B0%D1%87%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5

Приложение. Код программы

Код и структура проекта — https://github.com/spanickroon/Market-Place

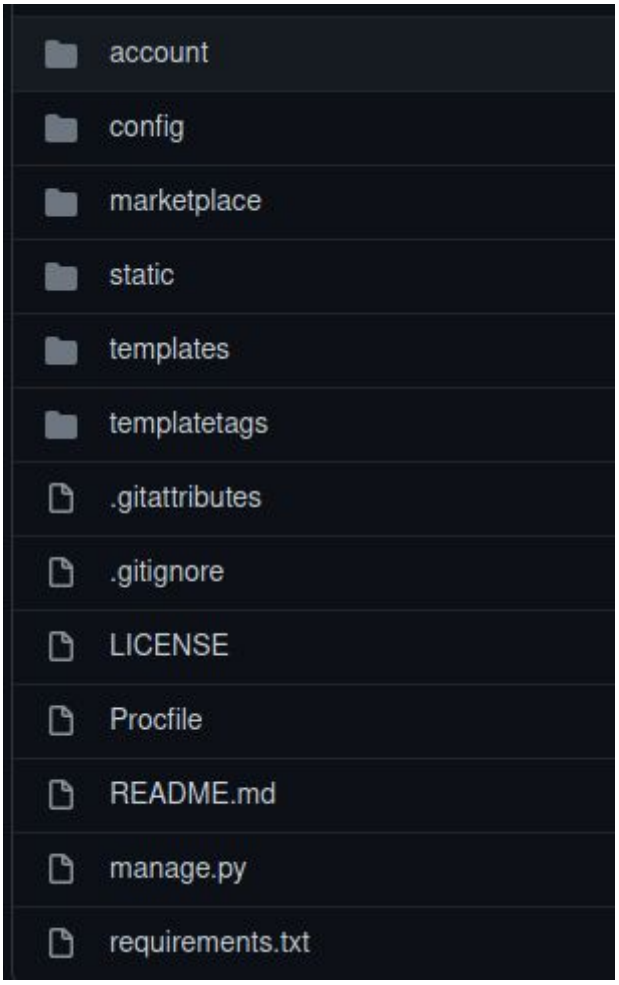

Рисунок А — Структура проекта

Config/Settings.py

```
"""
Django settings for marketplace project.

Using Django 3.0.8.

"""

import os
import dropbox

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

SECRET_KEY = os.environ.get('SECRET_KEY')
```

```
DEBUG = False

ALLOWED_HOSTS = ['0.0.0.0', '.herokuapp.com', '127.0.0.1', 'localhost']


# Application definition

INSTALLED_APPS = [
        'django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.messages',
        'django.contrib.staticfiles',
        'storages',

        'account.apps.AccountConfig',
        'marketplace.apps.MarketplaceConfig'
]

MIDDLEWARE = [
        'django.middleware.security.SecurityMiddleware',
        'django.contrib.sessions.middleware.SessionMiddleware',
        'django.middleware.common.CommonMiddleware',
        'django.middleware.csrf.CsrfViewMiddleware',
        'django.contrib.auth.middleware.AuthenticationMiddleware',
        'django.contrib.messages.middleware.MessageMiddleware',
        'django.middleware.clickjacking.XFrameOptionsMiddleware',

        'whitenoise.middleware.WhiteNoiseMiddleware',
]

ROOT_URLCONF = 'config.urls'

TEMPLATES = [
        {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates/'), ],
        'APP_DIRS': True,
        'OPTIONS': {
        'context_processors': [
        'django.template.context_processors.debug',
        'django.template.context_processors.request',
        'django.contrib.auth.context_processors.auth',
        'django.contrib.messages.context_processors.messages',
        'django.template.context_processors.media',
        'django.template.context_processors.static',
        ],
        'builtins': [
        'templatetags.marketplacetags',
        ]
        },
        },
]
```

```
WSGI_APPLICATION = 'config.wsgi.application'


# Database

DATABASES = {
        'default': {
        'ENGINE': os.environ.get('DATABASE_ENGINE'),
        'NAME': os.environ.get('DATABASE_NAME'),
        'USER': os.environ.get('DATABASE_USER'),
        'PASSWORD': os.environ.get('DATABASE_PASSWORD'),
        'HOST': os.environ.get('DATABASE_HOST'),
        'PORT': os.environ.get('DATABASE_PORT')
        }
}

# Password validation

AUTH_PASSWORD_VALIDATORS = [
        {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
        },
        {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
        },
        {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
        },
        {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
        },
]


# Internationalization

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'Europe/Minsk'

USE_I18N = True

USE_L10N = True

USE_TZ = False


# Static files (CSS, JavaScript, Images)

STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATIC_URL = '/static/'

MEDIA_URL = 'img/'
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'static/img')

STATICFILES_DIRS = [
        os.path.join(BASE_DIR, 'static')
]

STATICFILES_STORAGE = 'whitenoise.storage.ManifestStaticFilesStorage'
DEFAULT_FILE_STORAGE = 'storages.backends.dropbox.DropBoxStorage'

DROPBOX_APP_KEY = os.environ.get('DROPBOX_APP_KEY')
DROPBOX_APP_SECRET_KEY = os.environ.get('DROPBOX_APP_SECRET_KEY')
DROPBOX_OAUTH2_TOKEN = os.environ.get('DROPBOX_OAUTH2_TOKEN')
DROPBOX_ROOT_PATH = os.environ.get('DROPBOX_ROOT_PATH')
```

Account/views.py

```
"""Module with account views."""
from django.shortcuts import render
from django.views import View
from django.http import JsonResponse
from django.contrib.auth import logout
from django.views.decorators.csrf import csrf_protect
from django.utils.decorators import method_decorator

from .forms import LoginForm, SignupForm
from .services.authentication import AuthenticationHandler


class LoginView(View):
        """Login View."""

        @method_decorator(csrf_protect)
        def get(self: object, request: object) -> object:
        """Get request login view."""
        logout(request)
        return render(request, template_name='index.html')

        @method_decorator(csrf_protect)
        def post(self: object, request: object) -> object:
        """Post request login view."""
        login_form = LoginForm(request.POST)

        if login_form.is_valid():
        return AuthenticationHandler.login_handler(request, login_form)

        return JsonResponse(
        {'message': AuthenticationHandler.form_erors(login_form)})


class SingupView(View):
        """Signup View."""

        @method_decorator(csrf_protect)
```

```
        def get(self: object, request: object) -> object:
        """Get request signip view."""
        logout(request)
        return render(request, template_name='index.html')

        @method_decorator(csrf_protect)
        def post(self: object, request: object) -> object:
        """Post request signip view."""
        signup_form = SignupForm(request.POST)

        if signup_form.is_valid():
        return AuthenticationHandler.signup_handler(request, signup_form)

        return JsonResponse(
        {'message': AuthenticationHandler.form_erors(signup_form)})
```

Account/services/authentication.py

```
"""This module contain functions for authentication."""

import json
from django.http import JsonResponse
from django.contrib.auth import login
from django.contrib.auth.models import User
from django.contrib.auth.hashers import make_password, check_password
from django.http import JsonResponse
from django.shortcuts import render
from django.template.loader import render_to_string

from ..models import Profile
from marketplace.models import MyStock


class AuthenticationHandler:
        """Class responsible for regulating authentication processes."""

        @staticmethod
        def form_erors(signup_form: object) -> str:
        """Method that shows form errors."""
        res = []
        for k, v in json.loads(signup_form.errors.as_json()).items():
        res.append(v[0]['message'])
        return ' '.join(res)

        @staticmethod
        def login_handler(request: object, login_form: object) -> object:
        """Account login processing method."""
        request_username = login_form.cleaned_data['username']

        if not User.objects.filter(username=request_username).exists():
        return JsonResponse({'message': 'User not found'})

        user = User.objects.get(username=request_username)
```

```
if not check_password(
login_form.cleaned_data['password'],
user.password):
return JsonResponse({'message': 'Incorrect password'})

login(request, user)

return JsonResponse(
{'message': 'ok',
'template': render_to_string(
            request=request,
            template_name='marketplace/profile.html',
            context={
            'user': request.user,
            'mystocks': MyStock.objects.filter(
            user=request.user)})})

@staticmethod
def signup_handler(request: object, signup_form: object) -> object:
"""Account registration processing method."""
user = User(
username=signup_form.cleaned_data['username'],
password=make_password(signup_form.cleaned_data['password1'])
)

profile = Profile(
user=user,
)

user.is_active = True
user.save()
profile.save()

return JsonResponse({'message': 'ok'})

@staticmethod
def change_password_handler(
request: object, change_password_form: object) -> object:
"""The method that is responsible for changing the account password."""
user = User.objects.get(username=request.user.username)

password1 = change_password_form.cleaned_data.get('password1')
password2 = change_password_form.cleaned_data.get('password2')

user.password = make_password(password2)
user.save()

login(request, user)

return JsonResponse({'message': 'ok'})

@staticmethod
def edit_pofile_handler(
```

```
        request: object, edit_profile_info: object) -> object:
        """Change user data in profile."""
        new_username = edit_profile_info.cleaned_data.get('username')

        if User.objects.filter(username=new_username).exists() \
        and new_username != request.user.username:
        return JsonResponse(
        {'message': 'A user with this username already exists'})

        user = User.objects.get(username=request.user.username)

        user.username = new_username

        user.profile.save()
        user.save()

        login(request, user)

        return JsonResponse(
        {'message': 'ok',
        'template': render_to_string(
                request=request,
                template_name='marketplace/profile.html',
                context={'user': request.user})})
```

Marketplace/views.py

```
"""Module with marketplace views."""
from django.shortcuts import render, redirect
from django.views import View
from django.http import JsonResponse
from django.contrib.auth.decorators import login_required
from django.views.decorators.csrf import csrf_protect
from django.utils.decorators import method_decorator
from django.template.loader import render_to_string

from account.forms import ChangePasswordForm, EditProfileInfo
from account.services.authentication import AuthenticationHandler
from .services.marketplace import MarketPlaceHandler


class StocksView(View):
        """Stock view."""

        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def get(self: object, request: object) -> object:
        """Get request stock view."""
        return render(
        request, template_name='index.html',
        context={
        'stocks': MarketPlaceHandler.get_stocks(1),
        'stocks_pages': MarketPlaceHandler.get_stocks_pages()})
```

```
        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def post(self: object, request: object) -> object:
        """Post request stock view."""
        if request.POST['message'] == 'buy':
        return MarketPlaceHandler.post_buy_stock(request)
        else:
        return MarketPlaceHandler.post_display_stocks(request)


class ProfileView(View):
        """Profile view."""

        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def get(self: object, request: object) -> object:
        """Get request profile view."""
        return render(
        request, template_name='index.html',
        context={
        'user': request.user,
        'mystocks': MarketPlaceHandler.get_my_stocks(request.user)})

        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def post(self: object, request: object) -> object:
        """Post request profile view."""
        return MarketPlaceHandler.post_sell_stock(request)


class NotificationsView(View):
        """Notification view."""

        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def get(self: object, request: object) -> object:
        """Get request notification view."""
        return render(
        request, template_name='index.html',
        context={
        'notifications': MarketPlaceHandler.get_notifications(
                request)})


class PasswordView(View):
        """Password view."""

        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def get(self: object, request: object) -> object:
        """Get request passsword view."""
        return render(request, template_name='index.html')
```

```
        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def post(self: object, request: object) -> object:
        """Post request password view."""
        change_password_form = ChangePasswordForm(request.POST)

        if change_password_form.is_valid():
        return AuthenticationHandler.change_password_handler(
        request, change_password_form)

        return JsonResponse(
        {'message': AuthenticationHandler.form_erors(
        change_password_form)})


class StockGrowthView(View):
        """StockGrowth view."""

        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def get(self: object, request: object) -> object:
        """Get request stockgrowth view."""
        return render(request, template_name='index.html')

        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def post(self: object, request: object) -> object:
        """Post request stockgrowth view."""
        return MarketPlaceHandler.post_request_stocks(request)


class EditProfileView(View):
        """Editpofile view."""

        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def get(self: object, request: object) -> object:
        """Get request editprofile view."""
        return render(request, template_name='index.html')

        @method_decorator(csrf_protect)
        @method_decorator(login_required(login_url='login'))
        def post(self: object, request: object) -> object:
        """Post request editprofile view."""
        edit_profile_info = EditProfileInfo(request.POST)

        if edit_profile_info.is_valid():
        return AuthenticationHandler.edit_pofile_handler(
        request, edit_profile_info)

        return JsonResponse(
        {'message': AuthenticationHandler.form_erors(
        edit_profile_info)})
```

Marketplace/services/marketplace.py

```
"""This module contain functions for marketplace."""

import json
import math

from django.http import JsonResponse
from django.contrib.auth import login
from django.contrib.auth.models import User
from django.contrib.auth.hashers import make_password, check_password
from django.http import JsonResponse
from django.shortcuts import render
from django.template.loader import render_to_string
from django.template.loader import render_to_string
from ..models import Stock, MyStock, Notification
from account.models import Profile

from django.core.paginator import Paginator


class MarketPlaceHandler:
        """The class that handles the main processes of the site."""

        @staticmethod
        def get_stocks(page: int) -> object:
        """Getting specific promotions based on page number."""
        return Paginator(Stock.objects.all(), 8).page(page)

        @staticmethod
        def get_stocks_pages() -> list:
        """Getting pages depending on the number of promotions."""
        return [i + 1 for i in range(math.ceil(len(Stock.objects.all()) / 8))]

        @staticmethod
        def get_my_stocks(user: object) -> object:
        """A function that returns the user's stocks."""
        return MyStock.objects.filter(user=user)

        @staticmethod
        def buy_stock(request: object) -> str:
        """Stock purchase processing function."""
        user = request.user
        stock_id = request.POST['stock_id'].split('-')[-1]
        stock = Stock.objects.get(id=stock_id)
        profile = Profile.objects.get(user=user)

        if stock.cost <= profile.balance:
        profile.balance -= stock.cost

        if MyStock.objects.filter(stock=stock).exists():
        mystock = MyStock.objects.get(stock=stock)
        mystock.count += 1
        else:
```

```
mystock = MyStock(user=user, stock=stock, count=1)

mystock.save()
profile.deals_amount += 1
profile.save()
else:
notification = Notification(
user=user, cost=stock.cost,
message=f'Unsuccessful purchase {stock.name}')
notification.save()
return 'Insufficient funds'

profile.dividend_income = sum([
mystock.stock.dividend_income * mystock.count
for mystock in MyStock.objects.filter(user=request.user)])

profile.save()

notification = Notification(
user=user, cost=stock.cost,
message=f'Buy {stock.name}')
notification.save()

return 'ok'

@staticmethod
def post_buy_stock(request: object) -> object:
"""Balance update after purchase."""
return JsonResponse({
'message': MarketPlaceHandler.buy_stock(request),
'profile': f'{request.user}, {request.user.profile.balance:.2f}$',
'template': render_to_string(
request=request, template_name='marketplace/stocks.html')})

@staticmethod
def post_display_stocks(request: object) -> object:
"""Displaying stocks on the page."""
page = request.POST['active_page']
return JsonResponse({'message': 'ok', 'template': render_to_string(
request=request, template_name='marketplace/stocks.html',
context={
            'stocks': MarketPlaceHandler.get_stocks(page),
            'stocks_pages': MarketPlaceHandler.get_stocks_pages()})})

@staticmethod
def sell_stock(request: object) -> str:
"""Sale of stocks."""
profile = Profile.objects.get(user=request.user)
act, stock_id = request.POST['stock_id'].split('-')
mystock = MyStock.objects.get(stock=stock_id)

if act == 'sellall':
profile.balance += (mystock.stock.cost * mystock.count)
```

```
notification = Notification(
user=request.user, cost=mystock.stock.cost * mystock.count,
message=f'Sell {mystock.stock.name}. Count: {mystock.count}')

mystock.delete()
else:
profile.balance += mystock.stock.cost

notification = Notification(
user=request.user, cost=mystock.stock.cost * mystock.count,
message=f'Sell {mystock.stock.name}. Count: 1')

mystock.count -= 1
mystock.save()

if mystock.count == 0:
mystock.delete()

profile.dividend_income = sum([
mystock.stock.dividend_income * mystock.count
for mystock in MyStock.objects.filter(user=request.user)])

profile.deals_amount += 1
profile.save()
notification.save()

return 'ok'

@staticmethod
def post_sell_stock(request: object) -> object:
"""Refreshing the page after purchase."""
return JsonResponse({
'message': MarketPlaceHandler.sell_stock(request),
'template': render_to_string(
request=request, template_name='marketplace/profile.html',
context={'user': request.user, 'mystocks':
            MarketPlaceHandler.get_my_stocks(request.user)})})

@staticmethod
def get_notifications(request: object) -> object:
"""Display notifications."""
return Notification.objects.filter(
user=request.user).order_by('-datetime')

@staticmethod
def get_values_stocks(request: object) -> list:
"""Getting information about stocks."""
res = []
for stock in Stock.objects.all():
res.append({'name': stock.name, 'values': stock.string_to_json()})
return res

@staticmethod
def post_request_stocks(request: object) -> object:
```

```
        """Displaying information about stocks on a profile."""
        return JsonResponse({
        'message': 'ok',
        'content': MarketPlaceHandler.get_values_stocks(request)})
```

static/js/marketplace/googleCharts.js

```
function getStockValues() {
        fetch('stockgrowth',
        {
        method: 'POST',
        headers: {'content-type': 'application/x-www-form-urlencoded', 'X-CSRFToken': getCookie('csrftoken') },
        })
        .then(response => {
        if (response.status !== 200) {
        return Promise.reject();
        }
        return response.json();
        })
        .then(response => {
        if (response['message'] === 'ok') {
        content_response = response['content'];
        }
        })
        .catch(() => console.log('error response'));
        return content_response;
}

function getWindowSizes() {
        if (window.innerWidth >= 992) {
        return {width: 900, height: 400, fontTitle: 24, fontAxis: 22, fontValue: 16}
        } else if (window.innerWidth >= 768) {
        return {width: 500, height: 250, fontTitle: 14, fontAxis: 12, fontValue: 10}
        } else if (window.innerWidth >= 426) {
        return {width: 280, height: 125, fontTitle: 8, fontAxis: 7, fontValue: 4}
        } else {
        return {width: 250, height: 160, fontTitle: 8.5, fontAxis: 7, fontValue: 6}
        }
}

function setUpCharts() {
        var charts = [];

        getStockValues().forEach(element => {
        charts.push({
        title: `${element.name} Stock Chart`,
        chartID: `${element.name}_chart`,
        values: element.values
        })
        });

        page = document.getElementById('stockgrowth-charts');
```

```
	let resultHtml = [];

	for (var chart in charts) {
	resultHtml.push(`<div id=${charts[chart].chartID} class="stockgrowth-container"></div>`);
	}
	page.innerHTML = resultHtml.join('');

	setOnCallBack(charts);
}

function setOnCallBack(charts) {
	google.charts.setOnLoadCallback(function() {
	drawChart(charts);
	});
}

function drawChart(charts) {
	for (chart in charts) {
	var lastElelement = charts[chart].values.pop();
	charts[chart].values.push(lastElelement);

	var newDate = new Date();
	var sizes = getWindowSizes();
	var options = {
	width: sizes.width,
	height: sizes.height,
	title: `${charts[chart].title}.\nDate ${('0' + newDate.getDate()).slice(-2)}.${('0' +
newDate.getMonth()).slice(-2)}.${newDate.getFullYear()} ${('0' +
newDate.getHours()).slice(-2)}:${lastElelement[0]}. Price ${lastElelement[1].toFixed(2)}$`,
	titleTextStyle: {
	color: '#FFFFFF',
	fontSize: sizes.fontTitle,
	bold: false
	},
	hAxis: {
	title: 'Time',
	gridlines: {
		color: '#C4C4C4'
	},
	titleTextStyle: {
		color: '#FFFFFF',
		fontSize: sizes.fontAxis
	},
	textStyle: {
		color: '#FFFFFF',
		fontSize: sizes.fontValue
	},
	slantedText: true,
	slantedTextAngle: 45
	},
	vAxis: {
	title: 'Cost',
	minValue: 0,
	titleTextStyle: {
```

```
                color: '#FFFFFF',
                fontSize: sizes.fontAxis
        },
        textStyle: {
                color: '#FFFFFF',
                fontSize: sizes.fontValue
        },
        gridlines: {
                count: 8,
        },
        },
        legend : { position: 'none'},
        colors: ['#9C2394', '#2F048C'],
        backgroundColor: {
        fill: '#232323',
        fillOpacity: 0.9
        },
        };

        var data = new google.visualization.DataTable();
        var areaChart = new google.visualization.AreaChart(document.getElementById(charts[chart].chartID));

        data.addColumn('string', 'Date');
        data.addColumn('number', 'Cost');
        data.addRows(charts[chart].values);

        areaChart.draw(data, options);
        }
}
```