Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент, доцент кафедры УМФ,
к.ф.-м.н., доцент

_____/ Д.Е. Шафранов
«____»_____20___г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой,
д.ф.-м.н., доцент

_____/ А.А.Замышляева
«____»_____2018 г.

Разработка телеграм-бота для анализа и обработки информации о криптовалютах

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.03.02.2018.51.ПЗ ВКР

Руководитель работы, доцент
кафедры МиКМ, к.ф.-м.н., доцент

_____/ М.А Сагадеева
«____»_____2018 г.

Автор работы
студент группы ЕТ-412

_____/ С.С. Лепесев
«____»_____2018 г.

Нормоконтролер, к.э.н., доцент

_____/ Д.А. Дрозин
«____»_____2018 г.

Челябинск,
2018

АННОТАЦИЯ

Лепесев С.С. Разработка Телеграм-бота для анализа и обработки информации о криптовалютах. – Челябинск: ЮУрГУ, ЕТ-412, 47 с., 42 ил., 4 табл., библиогр. список – 15 наим., 2 прил.

Данная работа посвящена разработке Телеграм-бота для анализа и обработки информации о криптовалютах.

В работе выполнен обзор наиболее используемых ботов для криптовалют.

Разработана математическая модель, база данных и алгоритм работы приложения. Разработаны команды для взаимодействия с Телеграм-ботом. Реализован и отлажен сайт для введения некоторых объёмных данных.

Программа реализована на языке программирования C#, веб-интерфейс написан с помощью языков html, ccs, jquery. В программе используется СУБД MongoDB. Сам бот написан с помощью ASP.NET Core. Текст программы приведён в приложении.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В последние годы все больше набирает популярность различные программ-боты, которые имеют огромное значение в выполнении однообразной и повторяемой работы с максимальной возможной скоростью. Им находят применение в тех случаях, когда реакции человека недостаточно, например, игровые боты, боты для интернет-бирж [1].

Одним из удобных способов размещения бота, является приложение Телеграм [2]. Это кроссплатформенный мессенджер, в котором можно обмениваться сообщениями и файлами разных форматов. Телеграм имеет удобное API и библиотеки для разных языков программирования, с помощью которых можно без труда разработать бота для различных нужд.

Боты для интернет-бирж, сейчас очень стремительно появляются на рынке, так как упрощают анализ валютных пар, благодаря чему торгуют даже без участия человека. В их задачу входят решения по покупке и продаже валюты. Процесс происходит по набору стратегий, которые разработаны опытными “трейдерами” и финансовыми аналитиками. На данный момент существует два вида таких биржевых ботов – торговые и арбитражные [3]. В основном боты, на данный момент, для интернет-бирж на данный момент торгую цифровыми валютами, также называемыми как криптовалюты.

- ✓ Торговые боты торгуют в рамках одной биржи. Получают доход по средству скупки и продажи валют [4].
- ✓ Арбитражные боты торгуют в рамках нескольких бирж. Получают доходы скупая на одной бирже и перепродавая дороже на другой [5].

Криптовалюта – разновидность цифровой валюты, контроль и создание которой основано на криптографических методах. Функционирование такой валюты происходит на технологии блокчейн, направленный ациклический граф, и т.д. Все транзакции обычно не шифруются и находятся в открытом доступе, один из способов предотвращения подделки валюты [6].

Программ-бот для анализа и обработки информации о криптовалюте на интернет-биржах сейчас получает большую популярность, так как упрощает торговлю “трейдеру”. Эта программа позволяет анализировать и обрабатывать рынок криптовалюты с высокой скоростью. Это нужно для того, чтобы человек просматривал большее количество валютных пар за короткий промежуток времени, всегда был в курсе последних изменений на рынке и получал всю необходимую на данный момент информацию [6].

Анализ валютных пар – это набор методик, совмещающих в себе определенные правила и закономерности, графические и аналитические характеристики, для прогнозирования цены валютной пары в будущем. Видов анализа валютных пар существует большое множество. Самые популярные из них – это технический и фундаментальный виды анализа, остальные – это подвиды их методик. Трейдеры разных уровней спорят о том, какой из видов анализа валютных пар лучше. Но чаще всего

использование того или иного вида анализа в чистом виде заменяется их совмещением. Не нужно забывать, что в основе любых событий на рынке лежат события, подвластные фундаментальному виду анализа [7].

Один из самых популярных способов анализа валютных пар – это экспоненциальное скользящее среднее (EMA). EMA – является частным случаем взвешенного скользящего среднего. Цель данного сглаживания передача большего веса последним значениям цен, меньшего веса старым [8].

Важное условие для работы программ бота для торговых бирж, это торговое API.

API (Application Programming Interface) – набор готовых классов, методов, свойств, предоставляемых, в нашем случае, биржей для взаимодействия во внешних программных продуктах. Он позволяет получить нам данные исходя из настроек пользователя, очень быстро и продуктивно.

Разработка данного Телеграм-бота поможет трейдерам и майнерам в следующем:

- просматривать цены на интересующие криптовалюты;
- увидеть объемы торгов;
- отследить процент скачка и изменение цены;
- понять срок окупаемости в зависимости от личного оборудования;
- узнать изменение позиции валюты в маркете;
- получать краткий анализ криптовалютных пар.

Целью данной работы является разработка Телеграм-бота для сбора, обработки и анализа информации о криптовалютах.

Для достижения данной цели нужно решить следующие **задачи**:

- выполнить обзор существующих методов обработки и технического анализа данных криптовалютных пар;
- разработать и реализовать алгоритмы работы приложения Телеграм-бот;
- разработать базу данных настроек пользователей;
- реализовать взаимодействие с командами Телеграм-бота;
- выполнить проверку работы приложения Телеграм-бота;
- разработать вспомогательную программу, реализующую выбранный метод анализа информации.

1. МЕТОДЫ РАЗРАБОТКИ ТЕЛЕГРАМ-БОТА, АНАЛИЗ И ОБРАБОТКА ИНФОРМАЦИИ КРИПТОВАЛЮТНЫХ ПАР

1.1 Информация о криптовалютах

1.1.1 История создания криптовалют

Криптовалюта появилась именно тогда, когда человечество в ней нуждалось. Нестабильность, инфляция, волна кризисов, которые прошли по многим странам, поспособствовали тому, что обществу были необходимы независимые денежные средства.

В 2008 году впервые заговорили о криптовалюте, хоть и среди только IT-специалистов. Первые упоминания о ней, это статья, о денежной валюте которая основана на принципах криптографии. Сатоши Накамото предложил свою концепцию крипто-денег. И через год уже была запущена сеть под названием Bitcoin и виртуальные кошельки, что положило основу криптовалютам. Основополагающей технологией в создании криптовалюты легла технология блокчейн [7].

Январь 2009 – появление биткоина. Запушена сеть bitcoin, и был сгенерирован первый стартовый денежный блок. Через несколько дней была совершена первая транзакция.

27 ноября 2010 года впервые появились майнинг пулы, известный как Slush’s Pool. Хоть у него был и не эффективный алгоритм, который быстро взломали, но это дало начало созданию пулов, потому что тогда все поняли, что совместный майнинг лучше, чем индивидуальный.

В начале 2011 года начинают появляться интернет-биржи, которые работают с криптовалютой. Первая из них биржа MtGox, на которой можно было обменять биткойны на реальные деньги.

Летом 2011 года был представлен принцип подтверждения владения как ответ на неравное распределение голосов. Вместо получения вознаграждения за расшифровку блока, комиссия, после проведения транзакций, распределялась пропорционально между владельцами этой валюты. Через год была выпущена криптовалюта PPCoin, которая была полностью основана на этой системе.

В 2013 году общая мощность всех вычислительных систем обогнала в 256 раз топ-500 самых мощных суперкомпьютеров, вместе взятых во всем мире.

К 2015 году количество различных криптовалют растет и достигает больше 100, использующих разные или похожие алгоритмы шифрования.

В 2017 году на майнинг криптовалют уходило больше электроэнергии чем потребляли в Сирии, Кипре, Камбодже или Брунее

На сегодняшний день существует более 900 криптовалют. Сейчас майнинг из развлечения для людей малого круга, превратился в способ хорошо заработать. Огромное количество бирж, криптовалют, трейдеров, обменников только доказывают это. На данный момент эту валюту можно

выводить на банковские карты, счета в банках и электронных платежных систем. Сейчас люди готовы пользоваться ими в полную силу [5].

1.1.2 Технологии работы криптовалют

Блокчейн – встроенная по определенным правилам непрерывная цепочка блоков, или также связный список, содержащие разную информацию. Чаще всего копии цепочек этих блоков содержаться на множестве разных компьютерах и не зависимы друг от друга.

Впервые этот термин появился как название полностью распределенной базы данных и реализовывался в системе Биткоин, благодаря чему блокчейн часто относят к транзакциям в криптовалютах, однако эта технология может быть распространена на любые связные между собой блоки.

Блок транзакций – специальная структура для формирования групп транзакций в системе Биткоин и похожих. Транзакция считается подтвержденной и завершенной если, проверены ее подписи и формат, и когда сама транзакция объединена с несколькими другими в группу и записана в специальную структуру – блок.

Блок состоит из заголовка и списка транзакций. Заголовок включает в себя хэш, хэш предыдущего блока, все хэши транзакций, и дополнительную информацию.

Созданный блок будет принят пользователями тогда, когда числовое значение хэша не более определенного числа. Так как результат хэш-функции SHA-256 является необратимым, на данный момент нет алгоритма для расшифровки зашифрованного сообщения, кроме случайного перебора. Если хэш не удовлетворяет условию, то его пересчитывают. Как только найден вариант, то блок рассылается по подключённым узлам, которые проверяет блок. Если все верно, блок добавляется в цепочку и следуюший блок добавит в себя его хэш.

Блоки формируются множеством майнеров. Удовлетворяющие некоторым критериям блоки отправляются в сеть, включаясь во все копии распределенной базы блоков. Однако часто происходит, когда новые блоки в разных частях сети, называю предыдущим один и тот же блок, и так получается, что цепочка может ветвиться. Таким образом можно проследить историю владения, с которой можно ознакомиться на специальных сайтах.

Пока транзакция не включена в блок, система думает, что количество валюты на адресе остается не именным. В этот момент есть возможность отправить одну и ту же единицу валюты разным получателем, но как только транзакция войдет в блок, все остальные автоматически отменят. Таким образом, попадание одной транзакции в блок – это гарантия ее достоверности вне зависимости от других транзакций с этими же единицами валюты.

Так как вычислительные мощности непостоянны, то за требование к хэшам отвечает специальный параметр называемый “сложность”. Этот

параметр пересчитывается раз в 2016 созданных и подтвержденных блоков, и так чтобы поддерживать среднюю скорость формирования блокчейна на уровне 2016 блоков в 2 недели. На практике получается, что при увеличении мощности сети – параметр сложность возрастает, и наоборот, при уменьшении мощности – сложность падает.

Майнинг – деятельность по созданию новых блоков в блокчейне для обеспечения функционирования криптовалютных платформ. За создание структурной единицы предусмотрено вознаграждение за счет новых единиц криптовалюты или комиссионных сборов. Майнинг сводится к серии вычисления параметров для нахождения хэша. Разные криптовалюты используют разные способы вычесления, но они всегда достаточно длительны по времени для нахождения правильного варианта, и быстры для проверки на соответствие. Такие вычесления используются для обеспечения защиты от повторного расходования тех же единиц, а вознаграждения позволяет стимулировать людей тратить свои вычислительные мощности для поддержания работы всей сети. Вероятность получения награды для майнеров, приблизительно равно, соотношению его вычислительной мощности к мощности всей рабочей сети [6].

Особенность задачи майнинга это использовать максимальное распараллеливание вычислений. Поэтому, в силу специфики, хорошо подошли графические процессоры, которые в сотни раз производительнее CPU.

Для уменьшения влияния фактора удачи и более стабильного получения вознаграждения майнеры стали объединять свои вычислительные мощности в пулы. Главной задачей пулов это возможность использовать максимальное распараллеливание процессов, тогда каждый участник пула находит свой вариант хэша. Пул, с точки зрения криптосистемы, это мощный майнер, который получает также получает вознаграждение [9].

Существуют 3 вида пула:

- Proportional – Награда, после того как блок будет найден, делиться пропорционально между всеми;
- PPS – Вознаграждается каждый полученный стандартный вариант;
- Score – Оценочная система вознаграждения стандартных вариантов.

Выпуск новых единиц валюты обычно происходит по заранее опубликованным правилам и никак не зависит от какого-либо регулирующего органа. Очень часто вознаграждение получает именно тот, кто сформировал блок в блокчейне. Вознаграждение может быть всегда неизменным, но для предотвращения инфляции награда постепенно падает.

1.2 Торговля криптовалютой на биржах. Существующие решения

Все обменные операции и спекуляции на различных интернет-биржах похожи на любую другую. Для получения прибыли нужно купить подешевле и продать дороже. Так получается, что все методы анализа и обработки данных становятся такими же, как и на площадках торговли ценными бумагами или на форексе [10].

Главные составляющие, которыми оперирует участник торговли на интернет-биржах криптовалют:

- график;
- ордера sell, buy для покупки и продажи;
- история недавних сделок
- объёмы торгов, проводимых на биржах;
- процент изменения стоимости валюты на рынке;
- различные вспомогательные графики и линии на нем (EMA (Рисунок 1.1), линии Боллинджера (Рисунок 1.2)).

Рисунок 1.1 – График btc/usdt с линиями EMA

Рисунок 1.2 – График btc/usdt с линиями Боллинджера

Стакан котировок – отображение желания торговать по определенному курсу криптовалюты. Можно основываясь на него делать анализ и принимать

решения по ведению торговли. На этих основаниях в стакане котировок с большим объёмом, можно прогнозировать дальнейшее поведение цены [11].

Еще один из способов анализа рынка криптовалют это разделение ордеров на пассивные и агрессивные. Ордер называется пассивным, если он выставлен на торги не по текущему курсу, а на уровне, который предполагает сильным сопротивлением или поддержкой. Эти уровни формируют уровни поддержки/сопротивления. В эти моменты, когда курс валюты пойдет к этому уровню, можно понять сможет она его пробить или после тестирования развернется в обратном направлении. Агрессивные ордера предназначены для торговли на бирже по текущему уровню. Так же их называют моментальными, потому что они исполняются при наличии ответного предложения. Эти ордера в основном формируют и поддерживают движение на графике пары валют [8].

Один из популярных способов анализа криптовалюты это Уровни сопротивления и поддержки для цены. Постоянно меняющаяся цена формирует график, который не имеет прямолинейного движения. На нем существуют пики и провалы, которые формируют максимумы и минимумы, также называемые локальными экстремумами. На значительном интервале времени крайние экстремумы называют значимыми и абсолютными. Если мы проведем последовательную линию по ряду максимумов и минимумов, то она покажет уровень сопротивления и поддержки. Это уровни и значения котировок, где курс валют явно почувствует преграду, и совершит разворот.

Сильные уровни поддержки появляются в местах большого скопления списка ордеров на покупку. Уровень сопротивления определяется в местах большого списка ордеров на продажу. Получается главная цель большинства трейдеров заключается в покупке валюты во время падения курса и продаже, когда она достигла высокие значения.

Линии сопротивления и поддержки можно представить в виде канала, и получается очень выгодно торговать на границах этого канала. Одна из основных задач анализа криптовалютных пар — это определение значимых уровней для ценового графика [7].

Одно из главных определений с чем сталкиваются трейдеры это – тренд и трендовое движение цены. Важно здесь знать это направление тренда. Тренд из себя представляет канал, который составлен из параллельных уровней поддержки/сопротивления.

Наклон уровней — это направление трендового движения. Если он направлен вверх, то его называют восходящим. Это говорит о том, что торговля валютой ведется с преобладанием покупок. При нисходящем все наоборот. Также существует боковой тренд – это движение, когда уровни поддержки/сопротивления расположены горизонтально и видно примерное равенство между объемом продаж и покупок.

Важно оценить интерес участников к той или иной валюте. В этом помогает история совершения сделок и общие объемы торгов на бирже к определенной паре [12].

Важно использовать объем торгов в анализе пар криптовалют. Объем торгов – это общее количество единиц торгуемой криптовалютой, которая перешла другим людям при исполнении ордеров за определенный промежуток времени. В анализе важным показателем является объем, который возник при достижении какого-либо уровня. Это является возможным сильным сигналом, для начала торговли в ожидании изменения цен.

На различных интернет-биржах для ведения торговли валютой используются понятия вертикального и горизонтального объема. Больше всего распространён вертикальный объем.

В этом случае формируется список с данными о размере всех заключенных сделках за выбранный период времени и показывается в виде столбиков на графике с привязкой ко времени. Наиболее хорошим вариантом является использование объема – это определение окончание коррекционного движения в составе тренда. Когда сильно возрастает объем при цене, идущей против основного тренда, может говорить об его окончании. Этот момент позволяет торговать в направлении тренда.

На данный момент можно выделить 2 класса ботов для торговли криптовалютой.

- ✓ Торговый бот торгует в рамках одной биржи. Получают доход по средству скупки и продажи валют.
- ✓ Арбитражный бот – это программа, которая постоянно просматривает и анализирует пары валют на разных биржах для дальнейшей покупки или продажи.

1.2.1 Торговый бот

Торговые боты – функционирую по особым алгоритмам, которые создаются на основе анализа закономерностей. После определения закономерностей происходит проверка алгоритма на основе исторических данных, и если прибыль устраивает, то создаются правила стратегии для торговли бота на бирже криптовалют. Некоторые боты дополнительно имеют индикаторы для более гибкой настройки стратегии, которые позволяют проанализировать текущую ситуацию на рынке.

Плюсы:

- Может эффективно торговать как одной, так и несколькими валютными парами.
- Большой выбор ботов этого класса.
- Большое количество стратегий, которые помогут подстроится под любую ситуацию на рынке.

Минусы:

- Бот строго привязан к одной единственной бирже
- Почти все боты платные, а те что предоставляются бесплатно, почти не приносят прибыли, что делает их бесполезными.

Примеры торговых ботов:

- Cloudbot 2 бот для торговли биткоинами. Он использует две несложные стратегии: long – покупает биткоин за вторую валюту в паре и продает дороже, заработок осуществляется во второй валюте. Short – покупает за биткойны вторую валюту в паре, и продает, заработок происходит в биткоинах.

- 1b bot использует уже готовые стратегии на валюте биткоин. В стратегиях используются около 150 переменных, что позволяет воссоздать любую ситуацию на бирже.
- Bot-2 облачный бот. Его преимущество в том, что он работает круглосуточно. Пользователю достаточно указать желаемую чистую прибыль и выбрать любую из предоставленных площадок.
- BTC Robot бот для торговли разными криптовалютными парами. Также включает в себя мониторинг курсов валют.
- Cryptotrader облачный бот, который позволяет создать свои стратегии торговли, но также есть уже существующие. Поддерживает большое количество крипто-бирж.
- PHP Trader мало известный бот, написанный на языке php. В ней собраны наиболее мощные торговые алгоритмы. На данный момент она торгует биткоинами и ефириумом. Работает только с биржей Coinbase.

1.2.2 Арбитражный бот

Арбитражный бот – выполняет такие же действия, но на нескольких биржах одновременно. То есть купить валюту можно на одной платформе, а продать на другой, по более выгодной цене.

Плюсы:

- Торгует в рамках нескольких бирж, что позволяет получает еще большую прибыль.
- Очень хорошо торгует одной парой валют.

Минусы:

- Плохо торгует несколькими криптовалютными парами.
- Почти все боты платные.

Примеры арбитражных ботов:

BTC ROBOT – один из самых первых арбитражных торговых ботов. Этот бот достаточно прост в настройке, но как показывает практика, приносит не самый высоких доход.

GEKKO – арбитражный бот с открытым исходным кодом. Использование данного бота, может быть простым, потому что он не имеет серьезных стратегий.

ZENBOT – продвинутый арбитражный бот с открытым исходным кодом. Для использования доступны множество функций таких как умная аналитика бирж или плагины. Это отличное решения для продвинутых пользователей.

TRADEWAVE – облачная платформа для создания арбитражных ботов, имеет продвинутую систему торговых стратегий.

1.3 Обоснование выбора средств разработки

Для написания Телеграм-бота для анализа и обработки информации о криптовалютах нужен мощный и очень гибкий язык программирования. Потому что, постоянные изменения в API бирж, в самих биржах, различных расчетах, количествах разных криптовалют и бирж, и новых возможностях в написании ботов, с подвигнут на доработку и расширения функциональности этого приложения.

1.3.1 Язык C#, платформа .Net Core 2.0, технология ASP.NET Core

На сегодняшний день C# является одним из самых гибких и мощных языков программирования. На нем пишутся совершенно различные программы: от небольших десктопных приложений до крупных веб-сервисов и веб-порталов, которые обслуживают миллионы людей в день. Этот язык является полностью объектно-ориентированным. Поэтому этот подход позволяет решить задачи по построению гибких, но в тоже время сложных, приложений.

Язык C# был специально создан для работы с платформой .Net. И после выхода фреймворка .Net Core, теперь приложении написанные на нем, можно без проблем запускать не только на операционной системе Windows, а также и на Linux и Macintosh. Платформа .Net Core обладает следующими особенностями:

- Поддержка нескольких языков. Основа платформы — это общеязыковая среда выполнения (CLR), из-за чего .NET поддерживает несколько языков программирования: C#, VB.NET, C++, F#. При компиляции код на любом из этих языков компилируется в сборку на общем для них языке CIL. Это, грубо говоря, ассемблер для платформы .NET. Благодаря чему мы можем комбинировать библиотеки, написанные на разных языках программирования. [4]
- Кроссплатформенность. .Net Core является полностью кроссплатформенным, с некоторыми допущениями. Для полной работы всех его библиотек (FCL) необходимо на компьютер установить фреймворк. Также с помощью него можно разрабатывать мобильные приложения под операционные системы iOS и Android. [1]
- Мощная библиотека классов (FCL). .Net представляет единую для всех языков библиотеку классов. И при написании любого приложения, веб-сервис или десктопное приложения, так или иначе мы задействуем FCL.
- Разнообразные технологии. Общеязыковая среда выполнения (CLR) и библиотека классов (FCL) являются основой для всех технологий, которые можно задействовать для разработки различных приложений.

Например, для работы с базами данных в FCL предназначена технология ADO.NET. Для разработки графических приложений с богатым и красивым интерфейсом WPF. А для написания веб-сайтов или веб-сервисов ASP.NET.

Платформа ASP.Net Core, новая предоставленная технология от компании Microsoft, разработана для создания различный родов веб-приложений, веб-сервисов и т.д. Это технология является кроссплатформенной, что позволяет развернуть приложение на основных популярных системах. [1]

Благодаря модульности платформы .Net Core все необходимые библиотеки могут быть загружены через пакетный менеджер Nuget, в котором содержатся десятки тысяч классов и библиотек, упрощающих разработку приложений.

Можно выделить несколько ключевых отличий от прошлых версий ASP.NET:

- Новый легковесный и модульный конвейер HTTP-запросов;
- Возможность развертывать приложение как на IIS, так и в рамках своего собственного процесса;
- Использование платформы .NET Core и ее функциональности;
- Распространение пакетов платформы через NuGet;
- Интегрированная поддержка для создания и использования пакетов NuGet;
- Единый стек веб-разработки, сочетающий Web UI и Web API;
- Конфигурация для упрощенного использования в облаке;
- Встроенная поддержка для внедрения зависимостей;
- Расширяемость;
- Кроссплатформенность: возможность разработки и развертывания приложений ASP.NET на Windows, Mac и Linux;
- Развитие как open source, открытость к изменениям [1].

1.3.2 HTML и JQuery

Для ввода большого объёма данных, было решено разработать вспомогательный сайт, для этого понадобились такие средства разработки как html, jquery [2].

HTML – стандартизированный язык разметки документов. Это язык интерпретируется браузерами в форматированный текст и отображается в браузере или на экране мобильных телефонов.

HTML – теговый язык разметки. Любой документ, представленный на этом языке, представляет из себя набор дескрипторов. Эти элементы как могут быть и пустыми, не содержать никакого текста, так и иметь атрибуты, например, размер шрифта [3].

JQuery – сегодня веб-программирование и разработка веб-сайтов невозможно представить без языка javascript. Но в настоящее время, все чаще используется не “голый” javascript, а его библиотеки и разные фреймворки. Одна из таких библиотек, на данный момент самая популярная, это JQuery.

JQuery не просто библиотека, она объединяет в себе целую экосистему, построенную на базовой библиотеки:

- jquery.ui предназначена для создания визуальных эффектов и интерфейсов;
- jquery.mobile используемая для разработки мобильных сайтов и т.д.

Преимущества работы с библиотекой JQuery:

- Упрощение работы с кодом. JQuery предлагает простой и элегантный синтаксис для взаимодействия элементами на веб-странице;
- Расширяемость. Весь код открыт для просмотра и изменения и, если что-то в библиотеки не устраивает, можно ее модифицировать. Также можно писать свои плагины для Jquery;
- Кроссбраузерность. JQuery имеет поддержку всех известных браузеров [3].

1.3.3 Система управления базой данных MongoDB

MongoDB – это новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи реляционным базам данных. В отличии от реляционных баз данных MongoDB предлагает документ-но ориентируемую модель, из-за чего она работает быстрее, обладает лучшей масштабируемостью. Вся система MongoDB может представлять не только одну БД, хранящуюся на одном физической сервере. Но ее функциональность позволяет расположить несколько БД на разных физических серверах, и они с легкостью могут обмениваться данными и сохранять целостность.

Формат данных MongoDB. На данный момент самый популярный формат передачи данных по сети является JSON. Он с легкостью описывает сложные структуры данных. Способ хранения данных в MongoDB похож в этом плане на JSON. Для хранения данных в этой БД используется BSON (binary JSON).

BSON позволяет работать с данными намного быстрее: быстрее выполняется поиск и обработка данных. Хотя он по сравнению с JSON имеет небольшой недостаток — это больший вес данных. Хотя скорость работы с BSON с лихвой окупает этот недостаток.

1.3.4 Система контроля версий Git и Source Tree

Система контроля версий – программное обеспечение для облегчения роботы с постоянно изменяющейся информацией. Эта система позволяет хранить несколько версий одного и того же документа и при необходимости можно вернутся к более ранним версиям, определять, кто и когда сделал изменения, и т.д.

Ядро Git представляет собой набор команд для командной строки с параметром. Все настройки хранятся в текстовых файлах. Такая система

позволяет сделать Git легко портируемым на любую платформу и легко интегрировать в любую систему.

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы. При импорте нового репозитория автоматически создается копия, соответствующая последнему состоянию репозитория на сервере.

Source Tree бесплатный git клиент для Windows. Позволяет графически увидеть представления веток в системе контроля версий. Так же упрощает работу с основными командами git: commit, push, merge, pull (Рисунок 1.3).

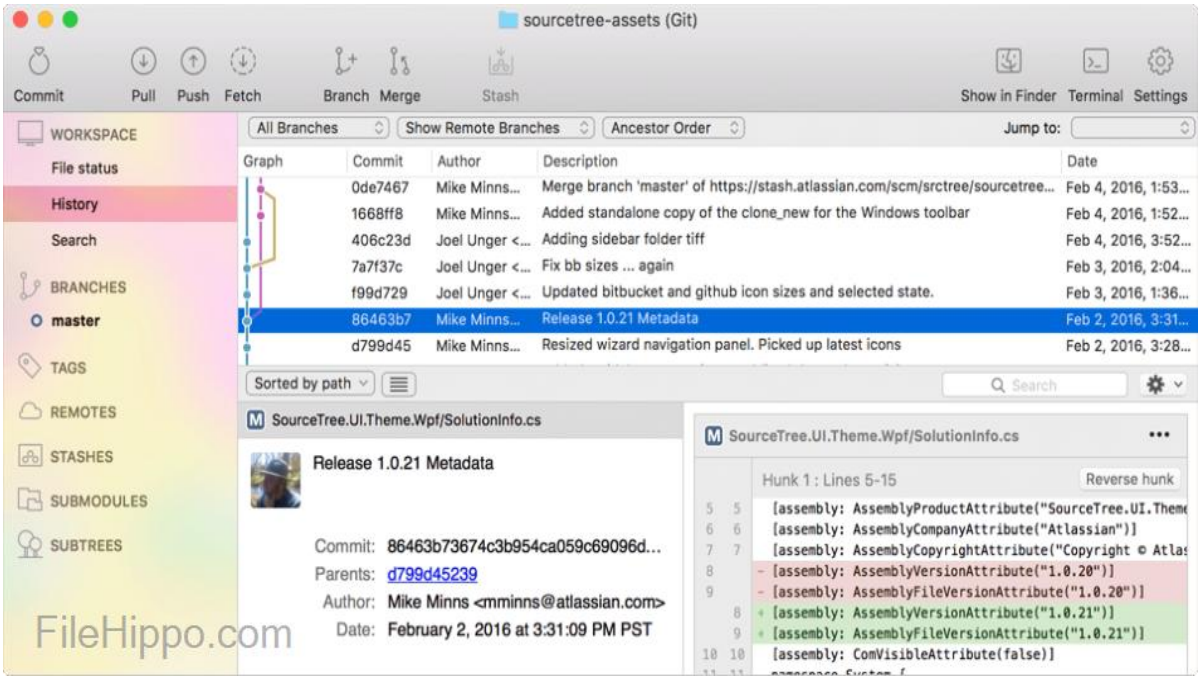


Рисунок 1.3 – Система контроля версий Source Tree

1.4 Выводы по разделу

Из обзора ясно, что нужна система, которая сможет собрать в себе все преимущества как арбитражных, так и торговых ботов. И к тому же будет служить информационной системой, для того чтобы пользователь мог сам контролировать рынок криптовалют. Поэтому актуально разработать аналог, который сможет нести в себе все эти преимущества, а также быть бесплатной альтернативой всем существующий ботам.

В качестве основной технологии выберем, язык C# и серверную технологию ASP.NET Core, так как это мощный механизм для построения крупных приложений, которые в будущем просто поддерживать и дополнять различной логикой.

2. РАЗРАБОТКА ПРИЛОЖЕНИЯ

2.1 Написание алгоритма программы

Для разработки приложения Телеграм-бот для анализа и обработки информации о криптовалютах, следует выделить отдельные модули, которые должны реализовывать определенную часть функций и выполнять следующие действия:

- Собирать информацию о криптовалютах
- Организовывать хранения данных
- Производить автоматический анализ данных
- Выполнять определенные действия на команды пользователей
- Уведомлять пользователей, в зависимости от произошедшего

Веб-сервис разрабатывается под операционные системы, Linux, MacOS, Windows. Для запуска необходимо установленный фреймворк версии .Net Core 2. Для старта сервиса необходимо ввести API ключ бота, ввести логин и пароль от БД, и строку подключения к БД. Основной алгоритм работы всего приложения расположен на рисунке 2.1.

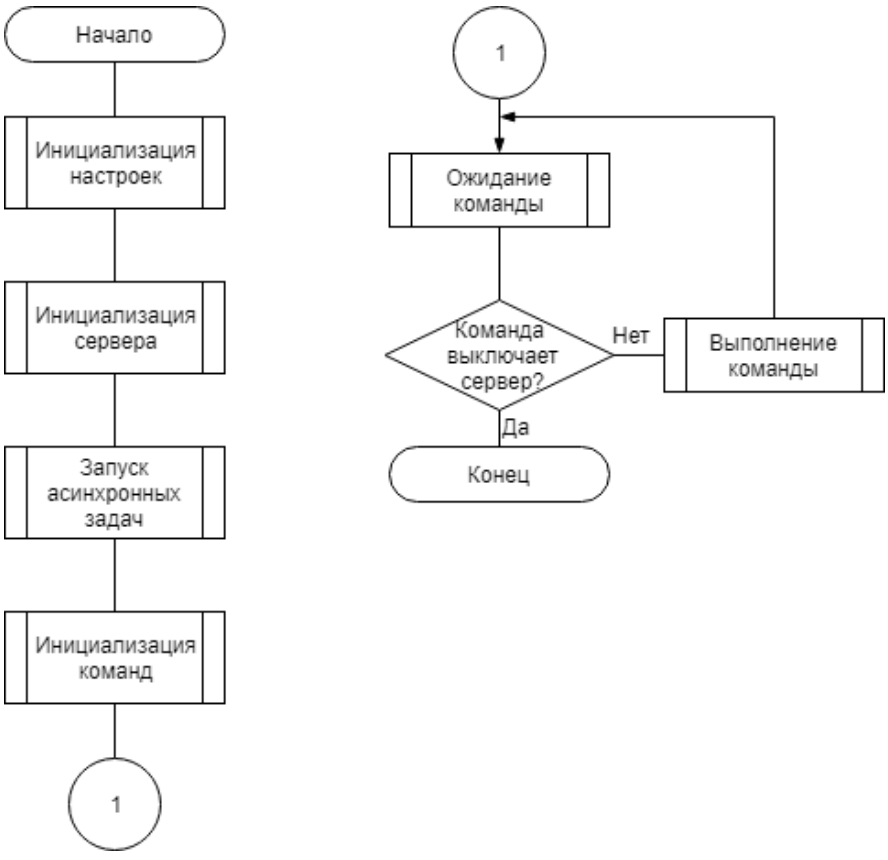


Рисунок 2.1 – Основной алгоритм

2.1.1 Алгоритм запуска приложения

После запуска приложения выполняется предварительная настройка и инициализация всех компонентов программы.

Далее для начала работы с ботом нужно открыть диалог с ним в Телеграме, и написать команду “/login password”, где password это пароль для получения доступа к боту. После чего происходит инициализации настроек для запуска алгоритма таймеров. Этот алгоритм показан на рисунке 2.2.

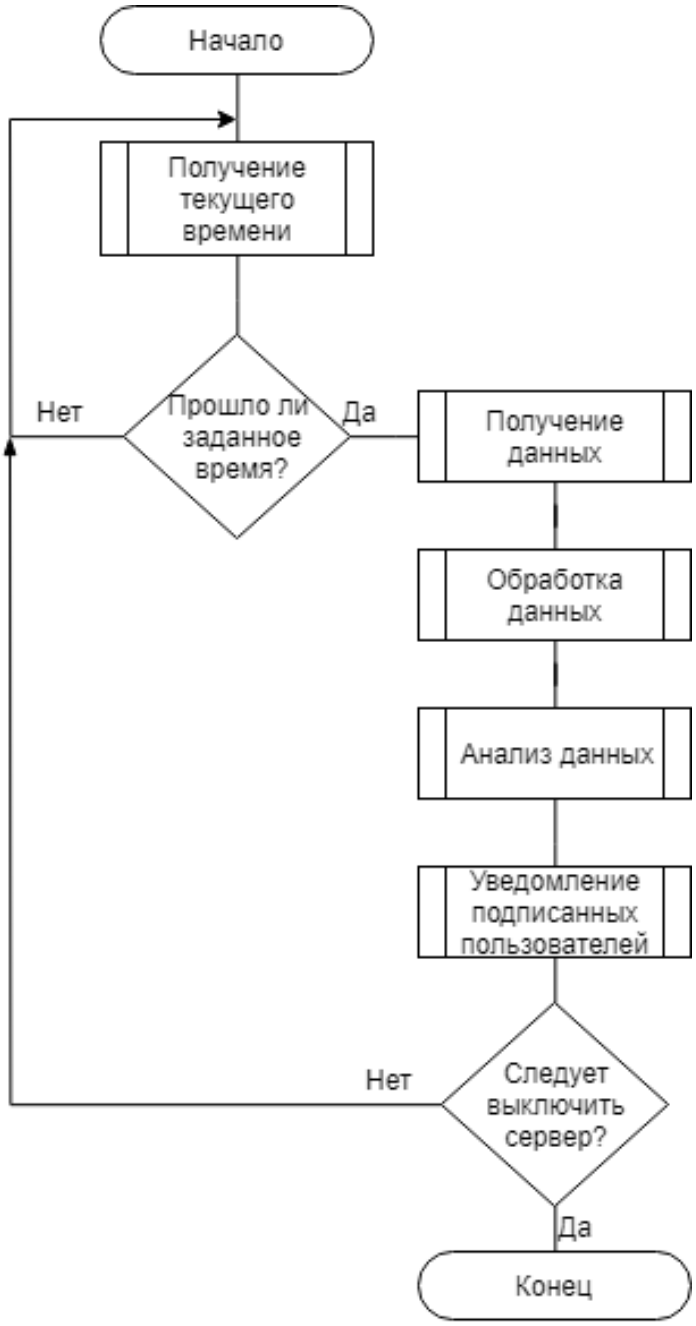


Рисунок 2.2 – Алгоритм обработки таймеров

2.1.2 Алгоритм обработки команд

При введении команды /help можно получить весь список команд доступных для пользователя. Для получения курса и процента изменения определенной криптовалюты нужно ввести в чат аббревиатуру этой валюты. Алгоритм обработки команд приведен в рисунке 2.3.

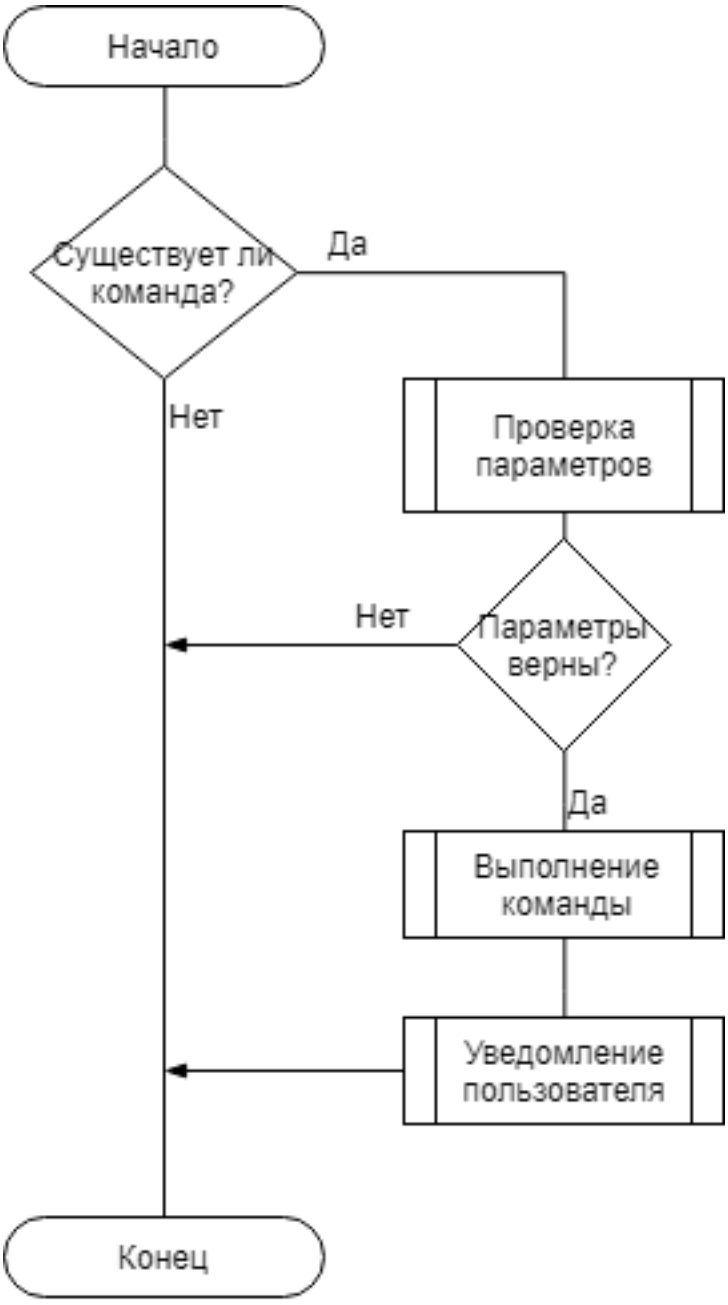


Рисунок 2.3 Алгоритм обработки команд

2.2 Проектирование базы данных

При проектировании базы данных нужно выполнить 3 основных этапа:

1. Концептуальное проектирование – формулирование, анализ и выделение основных требований к данным
2. Логическое проектирование – преобразование концептуальных требований в структурах данных.
3. Физическое проектирование – определение особенностей хранения данных, методов доступа и т.д. и реализация этих особенностей на основе одной из существующих БД.

Перед началом проектирования базы данных необходимо сформировать понятия о предметах, фактах и событиях, которыми будет оперировать данная система.

Для того чтобы привести основные понятия предметной области к той или иной модели данных, необходимо заменить их информационными представлениями. Одним из наиболее удобных инструментом унифицированного представления данных, не зависимо от реализующего его программного обеспечения, является модель “сущность-связь” (Entity-Relationship Model), или ER-модель. Она является наиболее известным представителем класса семантических (концептуальных, инфологических) моделей предметной области и предназначена для логического представления данных. Она определяет значение данных в контексте их взаимосвязи с другими данными. ER-модель обычно представляется в графической форме.

Основными элементами ER-модели являются объекты (сущности), атрибуты объектов и связи между ними. В свою очередь, связь между сущностями характеризуется типом связи и классом принадлежности. Класс может быть обязательным и необязательным. Если каждый экземпляр сущности участвует в связи, то класс принадлежности и – обязательный, иначе необязательный.

2.2.1 Представление сущностей в виде элементов не реляционной базы данных

Для преобразования ER-диаграмм к схеме базы данных, сущности и отдельные отношения были преобразованы в таблицы.

Таблица 2.1 хранит основную информацию об “Алертах”.

Таблица 2.1

Имя	Тип	Размер	Комментарий
Id	строка	255	Идентификатор пользователя
UserId	длинное целое	20	Идентификатор пользователя в Телеграме
NameCurrency	строка	255	Название валюты
PercentNotify	строка	255	Процент при котором мы будем уведомлять
NotifyCost	число с плавающей точкой	20	Цена при которой будем уведомлять
Cost	число с плавающей точкой	20	Текущая цена

Таблица 2.2 хранит информацию о пользователе.

Таблица 2.2

Имя	Тип	Размер	Комментарий
Id	строка	255	Идентификатор пользователя
TelegramUserId	длинное целое	20	Идентификатор пользователя в Телеграме
NotifyBittrix	логическая	1	Настройка уведомления пользователя с биржи Bittrix
NotifyPoloniex	логическая	1	Настройка уведомления пользователя

Окончание таблицы 2.2

Имя	Тип	Размер	Коментарий
NotifyCoinMarket	логическая	1	Настройка уведомления пользователя с биржи CoinMarket
TimeToCheckBittrix	число с плавающей точкой	20	Интервал проверки для биржы Bittrix
PoloniexPercentVolume	число с плавающей точкой	20	Настройка уведомления по проценту изменения
PoloniexPercentChangeCost	число с плавающей точкой	20	Настройка уведомления по проценту изменения
BittrixPercentVolume	число с плавающей точкой	20	Настройка уведомления по проценту изменения
BittrixPercentChangeCost	число с плавающей точкой	20	Настройка уведомления по проценту изменения
S9Cost	число с плавающей точкой	20	Цена майнера S9
L3PlusCost	число с плавающей точкой	20	Цена майнера L3Plus
D3Cost	число с плавающей точкой	20	Цена майнера D3
B8Cost	число с плавающей точкой	20	Цена майнера B8
Nvidia1070	число с плавающей точкой	20	Цена майнера Nvidia1070
Nvidia1080Ti	число с плавающей точкой	20	Цена майнера Nvidia1080Ti

Таблица 2.3 хранит информацию о асиках.

Таблица 2.3

Имя	Тип	Размер	Комментарий
Id	строка	255	Идентификатор пользователя
TelegramUserId	длинное целое	20	Идентификатор пользователя
Sha256_HR	число с плавающей точкой	20	HashRate алгоритма
Sha256_P	число с плавающей точкой	20	Power алгоритма
Scrypt_HR	число с плавающей точкой	20	HashRate алгоритма
Scrypt_P	число с плавающей точкой	20	Power алгоритма
X11_HR	число с плавающей точкой	20	HashRate алгоритма
X11_P	число с плавающей точкой	20	Power алгоритма
Quark_HR	число с плавающей точкой	20	HashRate алгоритма
Quark_P	число с плавающей точкой	20	Power алгоритма
Qubit_HR	число с плавающей точкой	20	HashRate алгоритма
Qubit_P	число с плавающей точкой	20	Power алгоритма
Cost	число с плавающей точкой	20	Цена оборудования

2.2.2 Вывод по проектированию БД

В данном разделе была разработана база данных.

Первоначально было выполнено концептуальное программирование, в ходе которого сначала были выделены сущности и связи в системе, а потом на основе них была построена ER-диаграмма.

Далее был осуществлен переход к даталогическому проектированию – построена схема базы данных и было приведено описание всех таблиц и атрибутов, находящихся в ней.

2.3 Разработка команд для бота и методов работ с ними

После открытия диалога с ботом нужно ввести команду /login ”password” для начала работы с ним, иначе ни одна из команд работать не будет. После того как вы залогинитесь ваше Телеграм id и базовые настройки будут добавлены в бд. И также вам теперь стали доступны все команды для работы с ботом.

1. Команда /main показывает главную информацию о главных криптовалютах с сайта poloniex. Информацию обновляется каждые 5 минут. Показывает текущий курс и процент изменения цены за текущий день по определенной криптовалюте (рисунок 2.4 и 2.5).

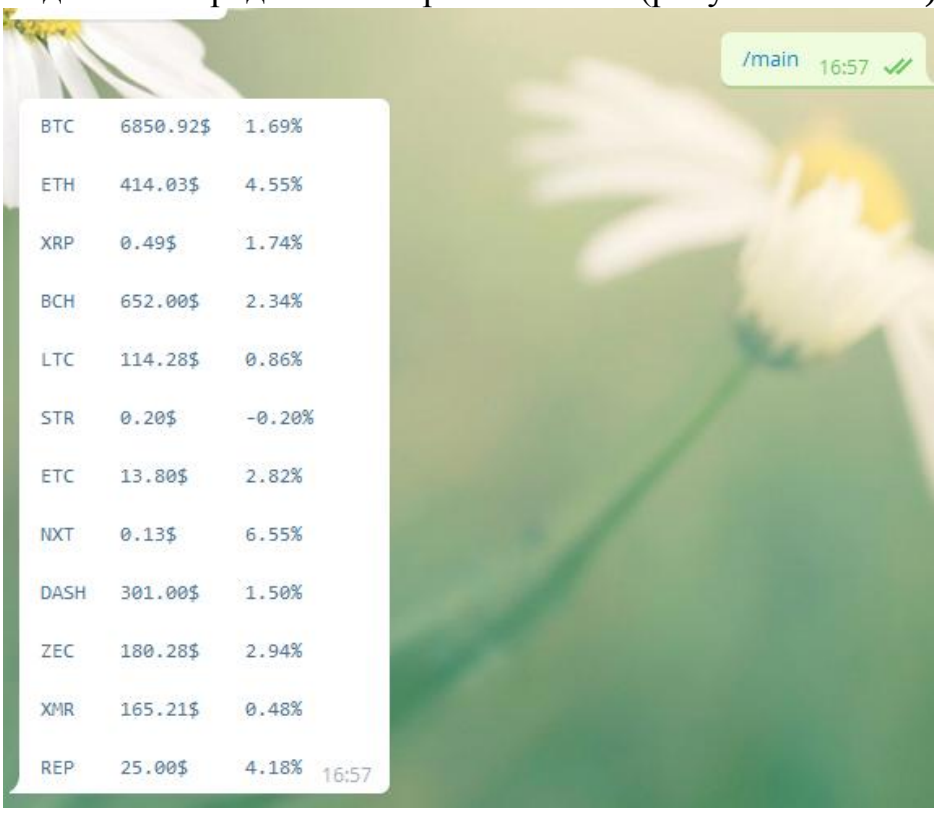


Рисунок 2.4 – Команда /main

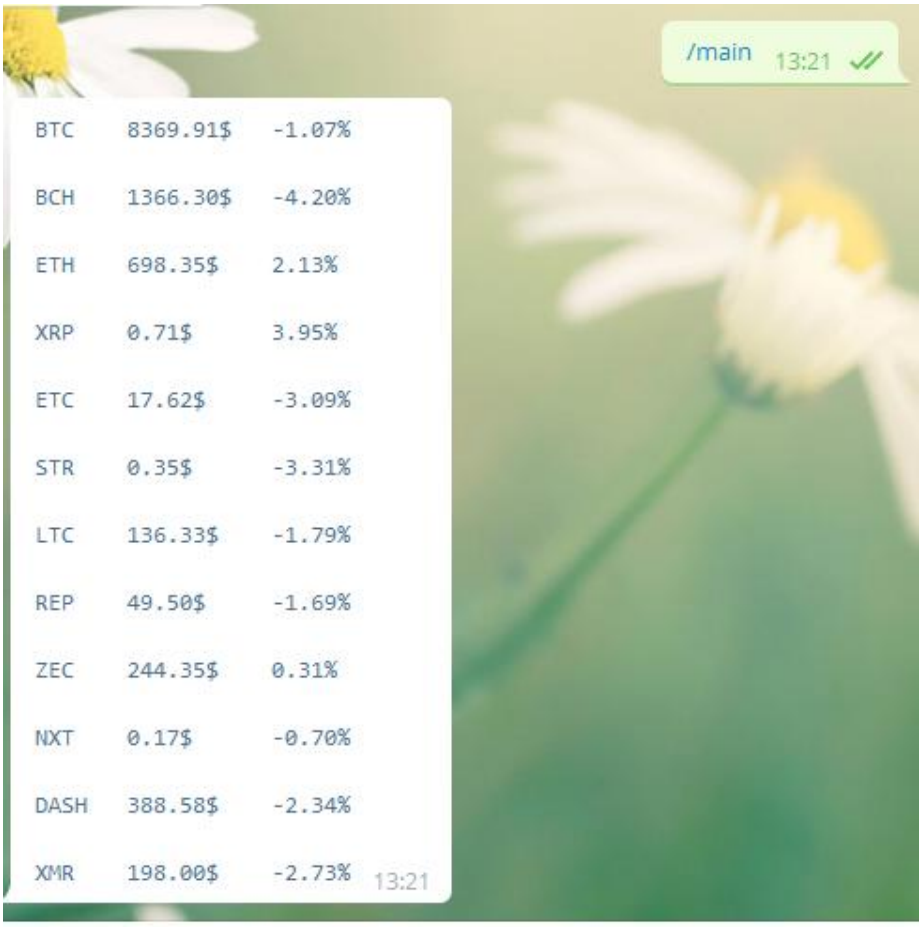


Рисунок 2.5 – Команда /main

2. Команды /bittrixtimecompare и /poloniextimecompare служат для настройки сравнения промежутка времени. Предназначены для биржевых сайтов Bittrix и Poloniex (Рисунок 2.6, 2.7).

Рисунок 2.6 – Команда /bittrixtimecompare

Рисунок 2.7 – Команда /poloniextimecompare

3. Команда /gettimesettings предназначена для просмотра настроек по командам /bittrixtimecompare и /poloniextimecompare (Рисунок 2.8).

Рисунок 2.8 – Команда /gettimesettings

4. Команды /bittrixcostchange и /poloniexcostchange служат для изменения параметров настроек уведомления о изменения цены криптовалюты в процентах. Чем меньше это значение, тем чаще будут приходить уведомления о изменения курсе. Работает как в сторону увеличения курса, так и уменьшения (Рисунок 2.9, 2.10).

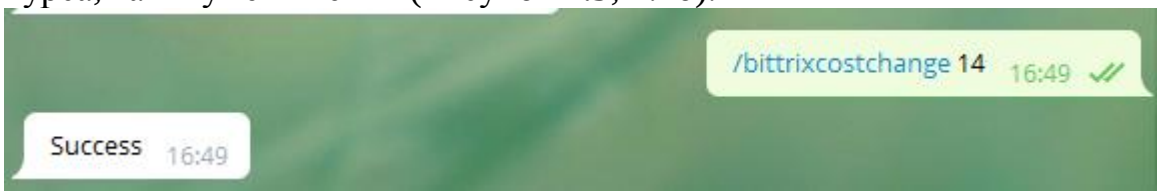


Рисунок 2.9 – Команда /bittrixcostchange

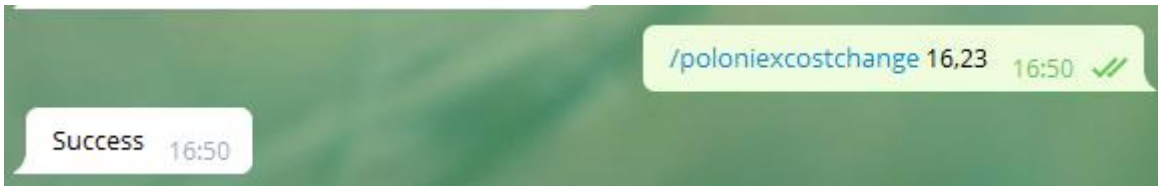


Рисунок 2.10 – Команда /poloniexcostchange

5. Команды /bittrixvolumchange и /poloniexvolumchange предназначены для изменения параметров настроек уведомления о изменения объема торгов какой-либо криптовалюты в процентах. Чем меньше это значение, тем чаще будут приходить уведомления о изменения курсе. Работает как в сторону увеличения курса, так и уменьшения (Рисунок 2.11, 2.111).

Рисунок 2.11 – Команда /bittrixvolumchange.

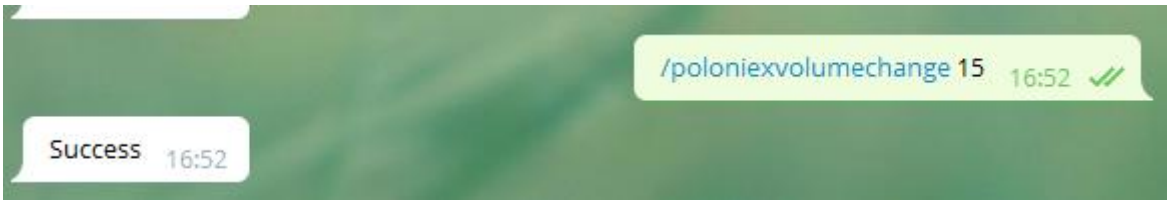


Рисунок 2.111 – Команда /poloniexvolumechange

6. Команда /getpercent используется для просмотра установленных настроек по командам /bittrixvolumchange, /poloniexvolumchange, /bittrixcostchange и /poloniexcostchange (Рисунок 2.12).

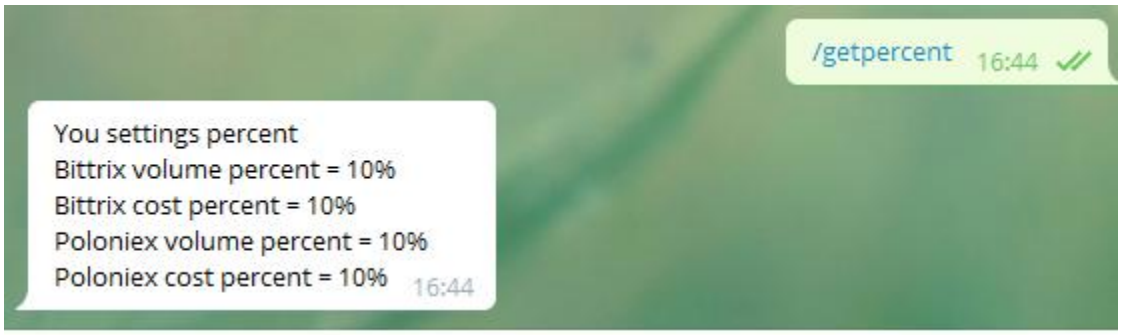


Рисунок 2.12 – Команда /getpercent

7. Команда /alert предназначена для установки специального уведомления пользователя о изменении выбранной валюты либо на какой-то процент, либо достижение определенной цены или ее снижения (Рисунок 2.13).

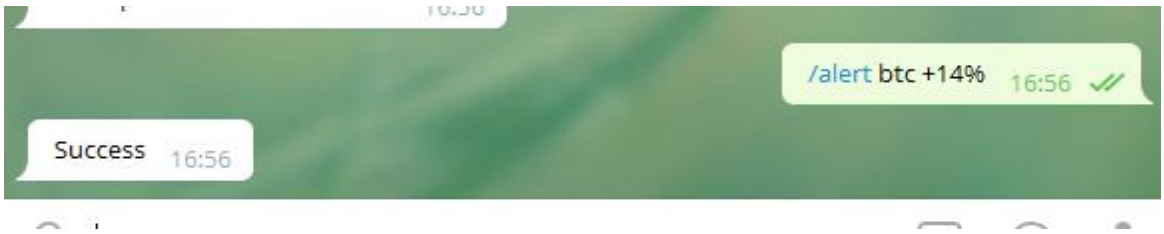


Рисунок 2.13 – Команда /alert

8. Команда /getalert предназначена для просмотра выбранной или выбранных настроек, установленные командой /alert (Рисунок 2.14).

Рисунок 2.14 – Команда /getalert

9. Команды /1070, /1080ti, b8, d3, l3, s9 предназначены для установки цены закупочного оборудования для майнеров: на видиокартах Nvidia 1070, на видеокартах Nvidia 1080Ti, B8, D3, L3Plus, S9. Цена закупочного оборудования устанавливается в долларах (Рисунок 2.15, 2.16, 2.17, 2.18, 2.19, 2.10).

Рисунок 2.15 – Команда /1070

Рисунок 2.16 – Команда /1080ti

Рисунок 2.17 – Команда /b8

Рисунок 2.18 – Команда /d3

Рисунок 2.19 – Команда /l3

Рисунок 2.20 – Команда /s9

10. Команда /hw предназначена для вывода установленных настроек по информации о цене закупочного оборудования (Рисунок 2.21).

Рисунок 2.21 – Команда /hw

11. Команда /profit выводит пользователю приблизительный срок окупаемости оборудования с учетом настроек цены закупочного оборудования, текущего курса и других параметров (Рисунок 2.22 и 2.221).

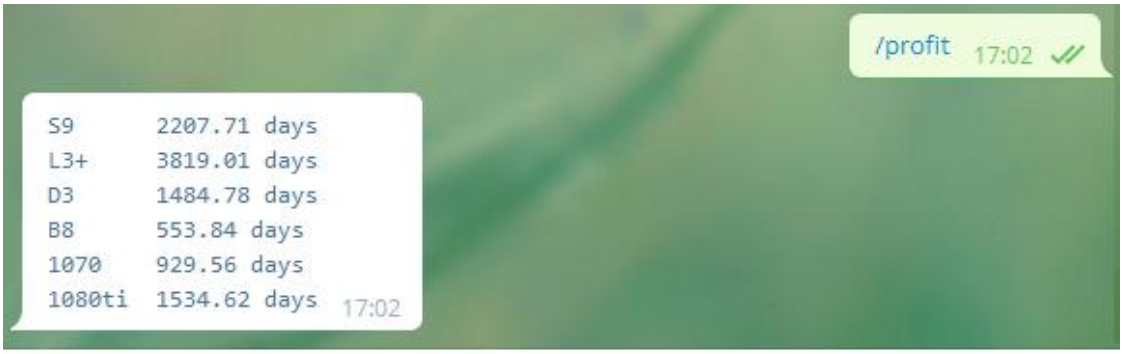
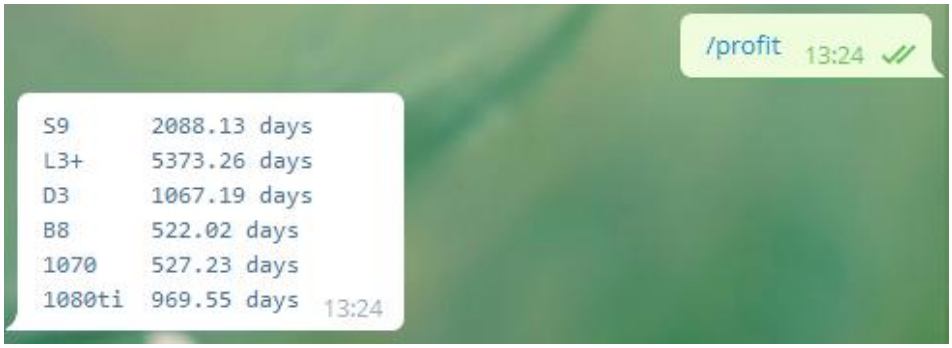


Рисунок 2.22 – Команда /profit



Команда 2.221 – Команда /profit

12. Команды /notifycoinmarket, /notifypoloniex, /notifybittrix предназначены для настройки показа уведомлений. При вводе одной из этих команд пользователю выводится две кнопки true и false где он может выбрать уведомлять о изменения на этих биржах или наоборот отменить уведомления (Рисунок 2.23, 2.24, 2.25).

Рисунок 2.23 – Команда /notifybittrix

Рисунок 2.24 – Команда /notifycoinmarket

Рисунок 2.25 – Команда /notifypoloniex

2.4 Разработка алгоритма уведомления о изменении курса и объема для валютных пар

После введения команды /login вам автоматически применяются предустановленные настройки, по которым бот сам понимает, когда и при каком изменении нужно вас уведомить о изменении курса и объёма валютной пары. На рисунках 2.26, 2.27, 2.28, 2.29 показано как работают уведомления при базовой настройке в 10%.

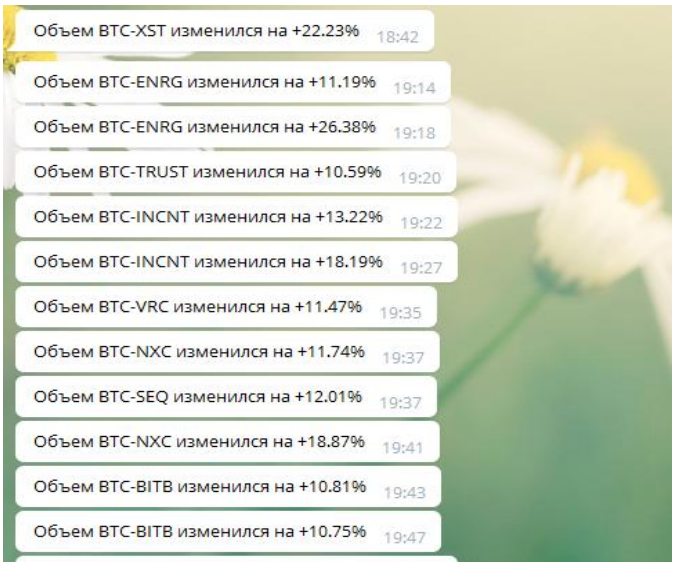


Рисунок 2.26 – Уведомление о изменении объёма

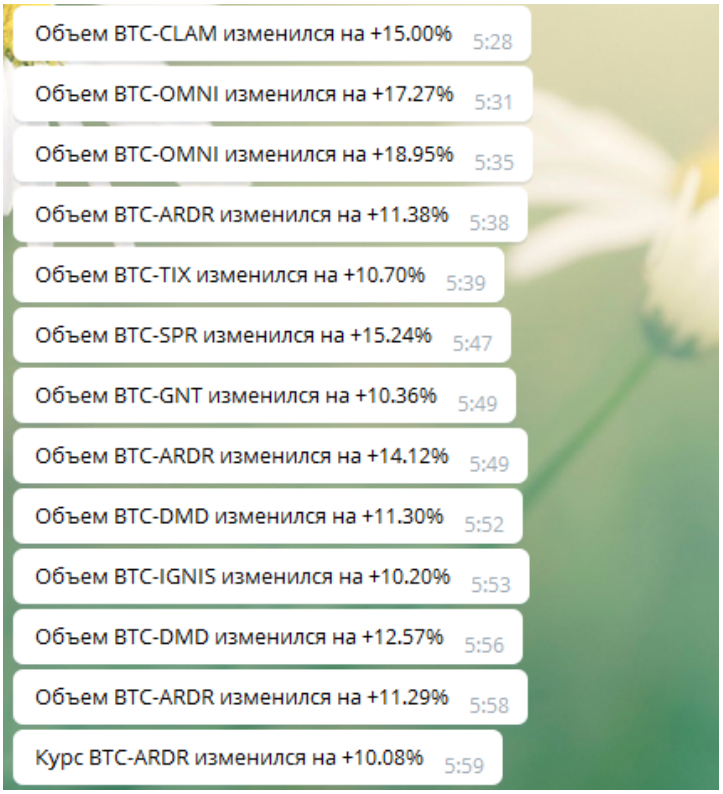


Рисунок 2.27 – Уведомление о изменении объёма

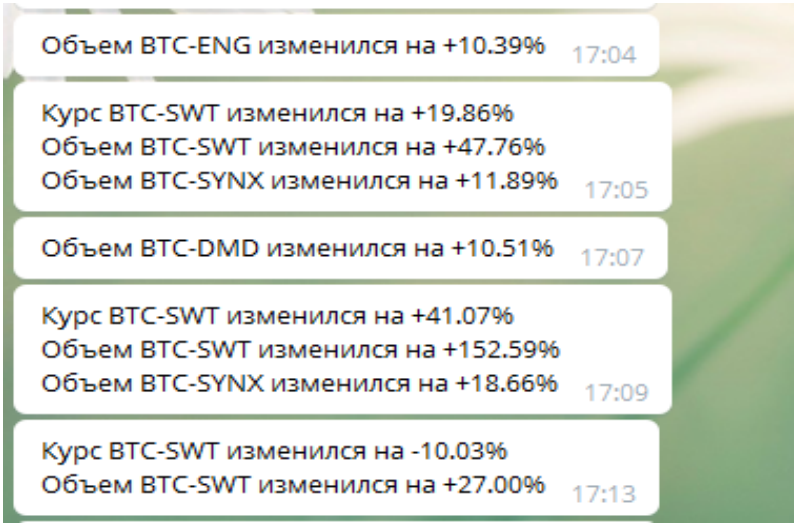


Рисунок 2.28 – Уведомление о изменении курса

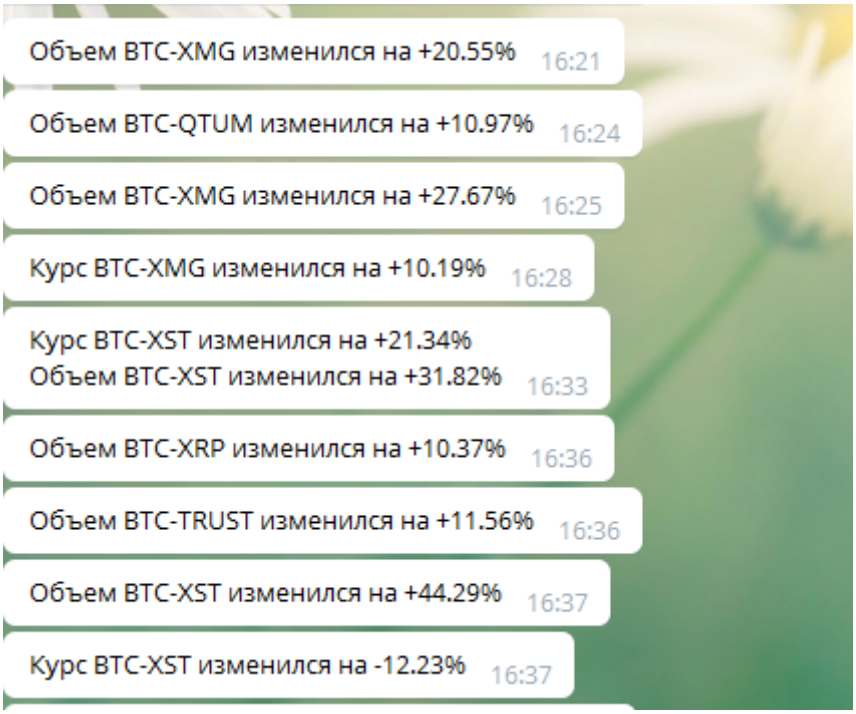


Рисунок 2.29 – Уведомление о изменении курса

2.5 Выводы по разделу

В результате разработаны алгоритмы роботы различных команд программ-бота, а также и всего веб-сервиса в целом.

Также был разработан удобный механизм взаимодействия с командами и различными функциями бота. Приложение состоит из множества различных команд и функций. Можно выделить несколько видов команд:

- редактирование значений алгоритма уведомления
- редактирование значений пользовательских настроек
- установки значений
- получения информации о валютах с разных бирж

Все эти команды предназначены для удобного использования и просмотра информации о почти всех криптовалютах в мире. С помощью этого бота можно быстро получить подробную информацию о криптовалюте, а именно текущую стоимость, текущий объём, процент изменения за день, и примерный анализ этой валюты.

3. АНАЛИЗ ДАННЫХ

3.1 Обзор основных методик анализа данных

Технический анализ – наиболее распространенный вид анализа валютного рынка. Он дарит ощущение полного контроля в прогнозах торгующего человека. Суть его заключается в анализе прошлого, так трейдеры анализируют прошедшие котировки и уровни цены, и на основе этого делают выводы о будущем [12].

Методология состоит из анализа ценовых фигур, которые состоят из прошедших котировок валютной пары, анализа линий сопротивления/поддержки (линии, которые цене пройти наиболее тяжело), а также использования специальных математических индикаторов, которые сигнализируют о состоянии на валютном рынке.

Так как наше приложение свободно манипулирует большим количеством текущих и прошлых данных, этот вид анализа наиболее подходящий.

3.1.1 Линии Боллинджера

Технический индикатор Линии Боллинджера – обычно наносится непосредственно на ценовой график [13]. Он представляет собой две линии, которые ограничивают динамику цены торгового инструмента сверху и снизу и находятся на удаленных от цены уровнях. Линии Боллинджера строятся в виде границ полосы вдоль скользящей средней, которая подразумевает текущий тренд. При их построении руководствуются следующим правилом: около 95% цен должны находиться внутри полосы индикатора, а 5% — за ее пределами.

Полосы Боллинджера представляют собой три линии:

1. Средняя линия ML (1) (обычное скользящее среднее) рассчитывается по формуле:

$$ML = SUM (CLOSE, N) / N = SMA (CLOSE, N) \quad (1)$$

где:

— SUM (…, N) — сумма за N периодов;

— CLOSE — цена закрытия;

— N — количество периодов, используемых для расчета;

— SMA — простая скользящая средняя.

2. Верхняя линия TL (2) (средняя линия ML, смещенная вверх на определенное число D стандартных отклонений StdDev) рассчитывается по формуле:

$$TL = ML + (D * StdDev) \quad (2)$$

3. Нижняя линия BL (3) (средняя линия ML, смещенная вниз на число стандартных D отклонений StdDev) рассчитывается по формуле

$$BL = ML — (D * StdDev) \quad (3)$$

StdDev (4) – стандартное отклонение рассчитывается как:

$$StdDev = SQRT (SUM ((CLOSE — SMA (CLOSE, N))^2, N)/N) \quad (4)$$

где SQRT — квадратный корень.

3.1.2 Уровни Фибоначи

Рисунок 3.1 – Уровни Фибоначи

Именно эти закономерности играют важную роль в применении технического анализа на рынке Форекс [14].

Причем следует подчеркнуть, что эти числа служат для определения возможного коррекционного хода цены, а также эти числа являются расширяющими.

Все числа сразу же для удобства высчитаны в процентах. В итоге мы получили два вида коэффициентов, играющих важную роль при определении уровней Фибонччи.

Рисунок 3.2 – Уровни Фибоначи

Следующим нашим шагом будет нахождение уровней Фибоначчи с помощью графики.

Если рассмотреть по внимательней график нашей торговли, то как раз можно и проследить так называемые коррекции — движения в обратном направлении. Когда линия на графике идет вниз, то это обозначает откат, нисходящий тренд, когда линия идет вверх — это соответственно рост.

Так вот эти изменения движения на противоположные идущему и называются коррекцией предыдущего движения.

В техническом анализе используются две крайние точки: рост и падение.

Уровни Фибоначчи (5) формул:

$$C = B – (B -) \text{ x } N\% \quad (5)$$

где B – Самая высокая точка на графике;

(B-) – Самая низкая точка на графике;

N – откат в процентах.

3.2 Экспоненциальное скользящее среднее (EMA)

Экспоненциальное скользящее среднее (англ. Exponential Moving Average, EMA) является частным случаем взвешенного скользящего среднего и применяется в техническом анализе как самостоятельная методика, так и в качестве составляющей части других индикаторов. Целью такого сглаживания является передача большего веса последним значениям цен, и меньшего веса более ранним [15].

В общем виде формула для расчета значения экспоненциального скользящего среднего в период времени t (EMAt) (6) может быть записана следующим образом:

$$EMA(t) = \alpha * P(t) + (1-\alpha) * EMA(t-1) \quad (6)$$

α – весовой коэффициент в интервале от 0 до 1, отражающий скорость старения прошлых данных: чем выше его значение, тем больший удельный вес имеют новые наблюдения случайной величины, и тем меньший старые;

P(t) – значение случайной величины в период времени t;

EMA(t-1) – значение экспоненциального скользящего среднего в период времени (t-1).

Для расчета оптимального значения α коэффициента не существует какой-либо математической формулы, а находится оно, как правило, методом подбора. Критерием отбора в данном случае выступает минимизация среднеквадратической ошибки отклонения фактического значения случайной величины от прогнозного. На практике это выглядит следующим образом:

1. Выбирается несколько значений α коэффициента.
2. Рассчитывается среднеквадратическая ошибка для каждого значения α.
3. Лучшим считается значение α, при котором среднеквадратическая ошибка будет минимальной.

Однако в практике технического анализа на реальном рынке такой подход не применим, поскольку статистический ряд постоянно дополняется новыми

значениями цен. Это делает невозможным одновременно зафиксировать α коэффициент и соблюсти критерий минимизации среднеквадратической ошибки. С этой целью для расчета α коэффициента (7) используется следующая формула:

$$= 2/(N+1) \quad (7)$$

где N – интервал сглаживания.

Следует понимать, что экспоненциальное сглаживание в этом случае не будет удовлетворять критерию минимизации среднеквадратической ошибки.

Также для расчета значения экспоненциального скользящего среднего в период времени t необходимо знать его значение в предыдущем периоде времени (t-1). При этом, в качестве первого значения берется простое скользящее среднее (англ. Simple Moving Average) с тем же самым интервалом сглаживания.

Пример расчета. Рассмотрим методику расчет экспоненциального скользящего среднего на примере данных о цене акций Компании XYZ за последние пятнадцать периодов, которые представлены в таблице.

№	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P	5,3	6,7	7,9	7,1	5,2	4,1	3,5	5,4	7,3	9,4	8,0	6,6	7,9	9,2	7,6
EMA				6,8	6,1	5,3	4,6	4,9	5,9	7,3	7,6	7,2	7,5	8,2	7,9

Таблица 1 – Данные о акциях

Предположим, что интервал сглаживания равен 4. В этом случае α коэффициент будет равен 0,4.

В качестве первоначального значения экспоненциального скользящего среднего используем простое скользящее среднее с интервалом сглаживание 4, которое составляет 6,8.

Следующее значение EMA, рассчитанное по приведенные выше формуле, составит 6,1.

Последующие значения рассчитываются аналогичным образом и графически представлены на рисунке 3.1.

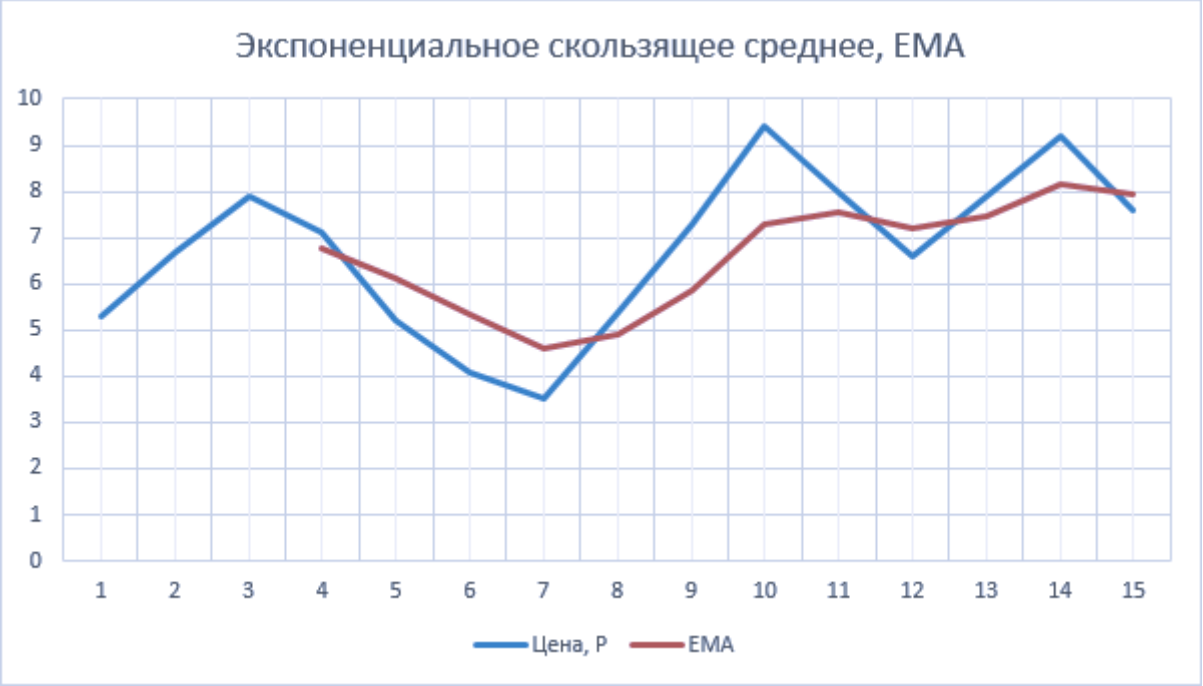


Рисунок 3.1 – График экспоненциальное скользящее среднее

Как можно видеть на графике, экспоненциальное сглаживание позволяет сгладить наиболее резкие отклонения цен и установить направление сложившегося на рынке тренда.

3.3 Разработка программы для анализа данных

Была разработана программа для получение данных с биржи криптовалют. Она получает данные по валютной паре BTC/USDT за 2017 год в формате JSON, после чего записывает их в текстовый файл. После чего программа выводит эти данные на график. Дальше над этими данными производится анализ данных (EMA), и также они выводятся на графике после чего можно посмотреть и сделать вывод о валюте BTC.

3.3.1 Блок схема программы для анализа данных

На рисунке 3.2 приведена блок-схема программы.

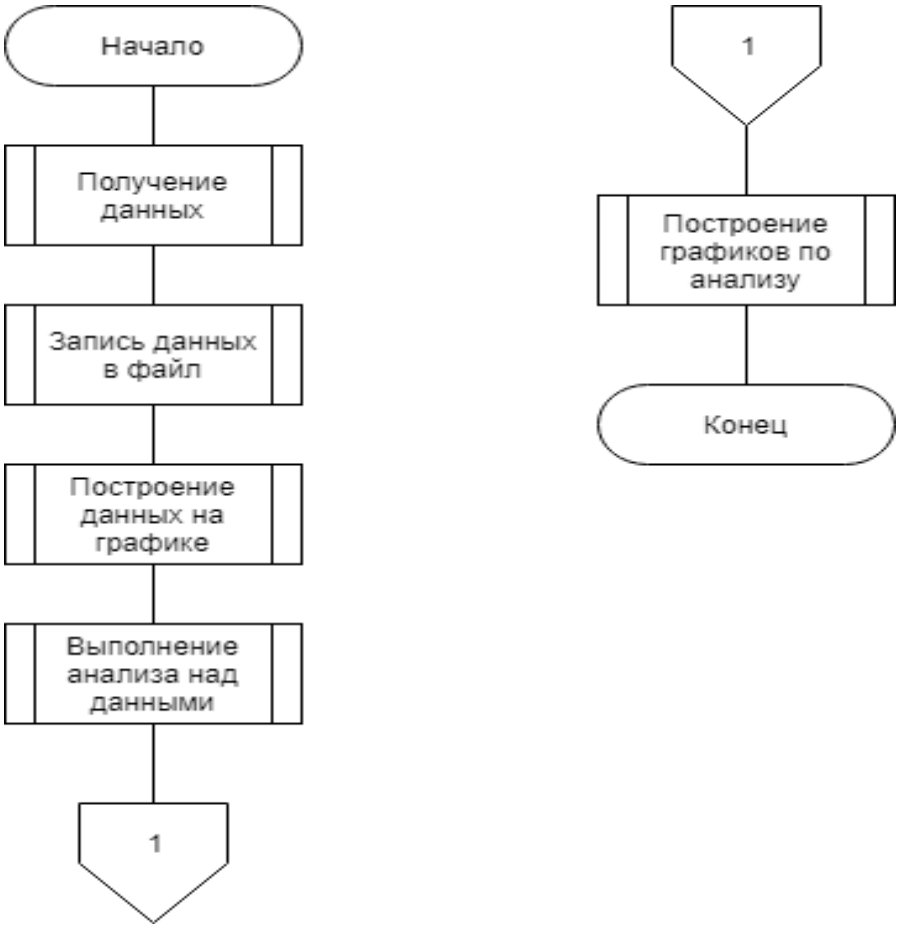


Рисунок 3.2 – Алгоритм работы программы по анализу данных

3.3.2 Пример работы программы

На рисунке 3.3 и 3.4 приведен пример работы программы.

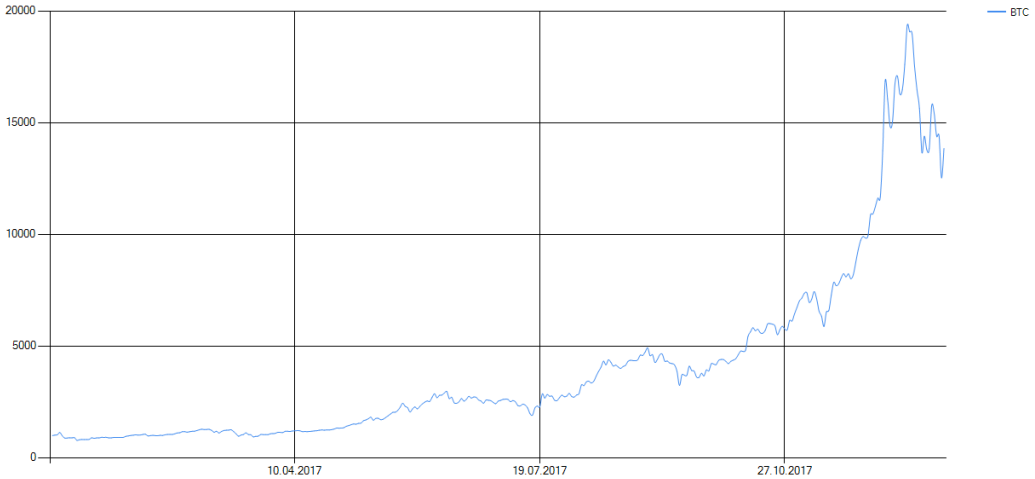


Рисунок 3.3 – Пример работы программы

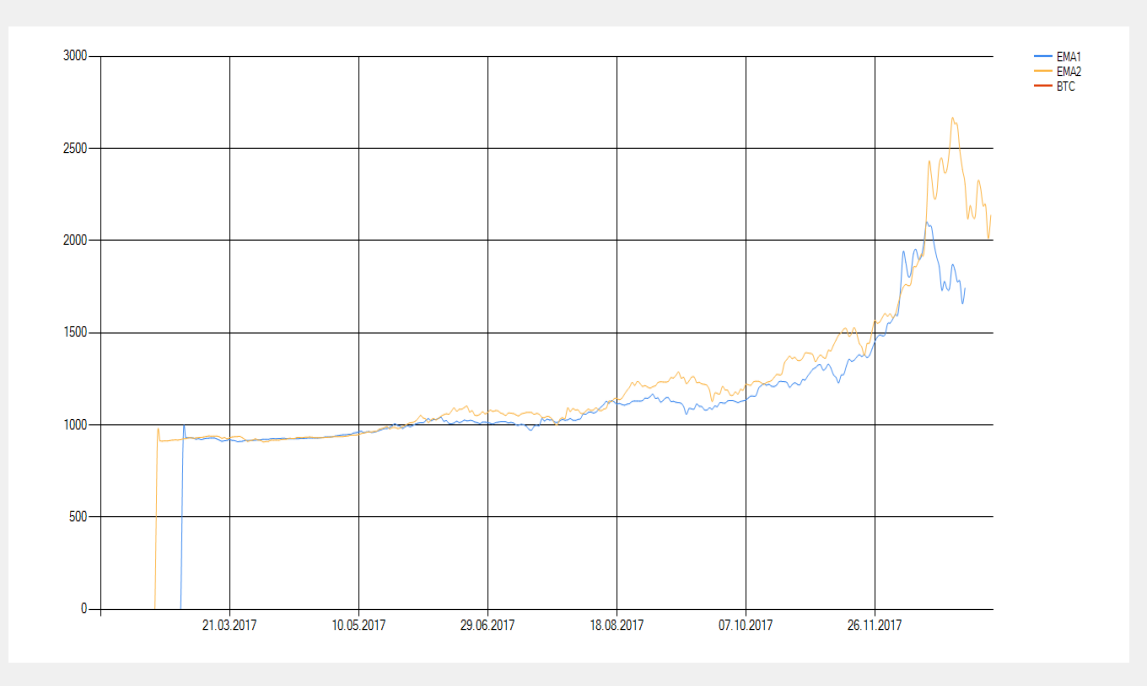


Рисунок 3.4 – Пример работы программы

3.4 Анализ котировок пары BTC/USDT за 2017 год

Приведем данные, собранные спроектированной программой за 2017 год, и построим график, где X это дата, а Y это цена валютной пары (Рисунок 3.5).

Рисунок 3.5 – График валютной пары BTC/USDT

Далее для анализа пары, построим график EMA и выделим особенности на этом графике (Рисунок 3.6).

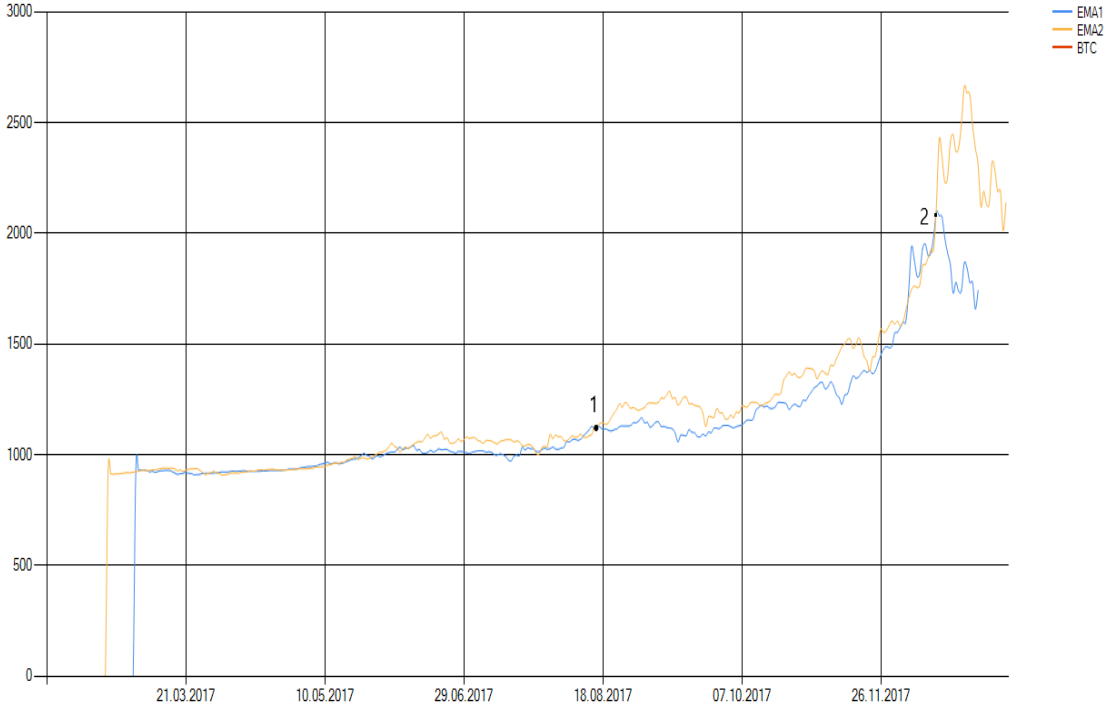


Рисунок 3.6 – Графики EMA

На графике можно увидеть 2 линии.

EMA1 – построен с α = 30.

EMA2 – построен с α = 20.

В точках 1 и 2 на видно пересечение графика EMA1 сверху EMA2, этот сигнал говорит о том, что график валютной пары пойдет вверх. Что и видно на рисунке 3.6.

Вывод: Из графиков мы видим, что этот способ хоть и является не точным, но вполне подходит для краткого обзора валютной пары. Благодаря описанной методике мы можем сделать небольшой анализ валютной пары, что будет полезно для трейдеров, и даст некоторое понимание состояния валюты на основе предыдущих данных.

3.5 Выводы по разделу

Экспоненциальное скользящее среднее (англ. Exponential Moving Average, EMA) является частным случаем взвешенного скользящего среднего и применяется в техническом анализе как самостоятельная методика, так и в анализе валютных пар является не заменимым способом узнать движение валютных пар на бирже. Эту методику удобно использовать на любом количестве данных, так как возможность выбора интервала сглаживания дает большую гибкость в ее использовании. Еще одним полезным свойством этого инструмента является его способность демонстрировать силу тренда, которая определяется по степени наклона кривой этого среднего. Чем круче угол, тем более сильным и ярко выраженным является ценовое движение.

ЗАКЛЮЧЕНИЕ

Данная работа посвящена актуальной теме разработка телеграм-бота для сбора информации о криптовалютах, а также по простому анализу различных криптовалютных пар.

Целью работы являлось разработка Телеграм-бота для сбора, обработки и анализа информации о криптовалютах.

В соответствии с целью был проведен обзор всех имеющихся решений в первой главе. Также классифицировали различные программ-боты из области торговли. Ознакомились с методами создания криптовалют, способы ее шифрования, и почему она получила такое распространение. Так же были приведены обоснования выбора средств разработки и вспомогательных инструментов.

Во второй главе был написан алгоритм программы, приведена блок схема работы программ бота и вспомогательной программы. Так же была спроектирована вспомогательная программа, которая строила графики по курсу и анализу валюты. Была спроектирована и разработана база данных, так же приведена ее ER-диаграмма. Разработаны команды для бота и способ взаимодействия с ними.

В третьей главе были приведены способы анализа валютных пар. Выполнен обзор способов анализа и также приведены примеры их работы. Выбран способ EMA была приведена математическая составляющая этого алгоритма и приведен пример его работы. Так же этот алгоритм был проверен на работе с экспериментальными данными, на которых было выяснено что этот способ анализа подходит для использования в боте.

В результате работы получен законченный программный продукт, который может использовать абсолютно любой человек.

Разработанный Телеграм-бот позволяет:

- просматривать изменение валютных пар;
- вычислять сроки окупаемости для разного оборудования;
- моментально узнать текущую цену любой валютной пары;
- получить прогноз по поводу будущего движения валютной пары.

В ходе работы были решены следующие задачи:

- выполнен обзор существующих методов обработки и технического анализа данных криптовалютных пар;
- разработаны и реализованы алгоритмы работы приложения Телеграм-бот;
- разработана база данных настроек пользователей;
- реализовано взаимодействие с командами Телеграм-бота;
- выполнена проверка работы приложения Телеграм-бота;
- разработана вспомогательная программа, реализующая выбранный метод анализа информации о криптовалютных парах.

Таким образом, все поставленные задачи полностью решены и цель достигнута. В дальнейшем планируется реализовать дополнительные способы анализа криптовалютных пар, а также портировать бота на разные платформы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Руководство по ASP.NET Core 2.0 / Metanit – Дата обновления: 15.11.2017. URL: https://metanit.com/sharp/aspnet5/ (дата обращения: 21.04.2018).
2. Руководство по JQuery / Metanit – Дата обновления: 15.11.2017. URL: https://metanit.com/web/jquery/ (дата обращения: 21.04.2018).
3. Руководство по HTML (дата обращения: 21.04.2018) https://metanit.com/web/html5/ (дата обращения: 21.04.2018).
4. Джеффри Рихтер, CLR via C# – М.: Русская редакция 2008г – 636С.
5. Бот для биржи криптовалют / Делаем успешный бизнес – Дата обновления: 04.10.2017. URL: https://delen.ru/onlayn-biznes/bot-dlya-birzhi-kriptovalyut.html (дата обращения: 23.04.2018).
6. Топ 6 арбитражных ботов bitcoin / Коин майнинг – Дата обновления: 04.03.2017. URL: https://coinsmining.ru/топ-6-арбитражных-ботов/ (дата обращения: 29.04.2018).
7. Фридрих фон Хайек, Denationalisation of Money: An Analysis of the Theory and Practice of Concurrent Currencies. — London: Institute of Economic Affairs, 1976.
8. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. — 2008. — 9 с.
9. Кирилл Сарханянц, Ольга Шестопал, Роман Рожков Много денег из ничего // Газета «Коммерсантъ», № 102/П (5133), 17.06.2013
10. Боты для торговли криптовалютой / CryptoNet – Дата обновления: 22.05.2017. URL: https://cryptonet.biz/ru/boty-dlya-torgovli-kriptovalyutoj-stoit-li-ispolzovat-na-birzhah/ (дата обращения: 28.04.2018).
11. Анализ валютных пар / Экономика – Дата обновления: 01.01.2017. URL: http://economic-definition.com/Currency_Pairs/Analiz_valyutnyh_par_Analysis_of_currency_pairs__eto.html (дата обращения: 01.05.2018).
12. Экспоненциальное скользящее среднее / Dewin Forex – Дата обновления: 22.12.2017. URL: http://dewinforex.com/ru/indikatory-foreks/indikator-ema-skolziashchie-srednie.html (дата обращения: 04.05.2018).
13. Стивен Б. Акелис, Технический анализ от А до Я.: Русская редакция 2006г – 376С.
14. Уровни (линии) Фибоначчи / Идеи профитной торговли – Дата обновления: 15.07.2016. URL: http://www.tevola.ru/trading/method-analiz/tekhnicheskij-analiz/urovni-fibonachchi.html (дата обращения: 06.05.2018).
15. Индикатор EMA — подробное описание / Информация о форексе – Дата обновления: 29.09.2016. URL: http://infofx.ru/torgovye-metody/indikator-ema-podrobnoe-opisanie-skachat-standartnyj-algoritm-dlya-torgovli-na-foreks/ (дата обращения: 14.05.2018).

ПРИЛОЖЕНИЕ 1

BittriexTask.cs

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using CryptoBot.Services;
using CryptoBot.Services.Stores;
using Microsoft.Extensions.Logging;

namespace CryptoBot.BackgroundTasks
{
    public class BittriexTask : IScheduledTask
    {
        private readonly HttpClientWrapper _client;
        private readonly CurrencyService _currencyService;
        private readonly IRepository<Alert> _alertRepository;
        private readonly ILogger<BittriexTask> _logger;
        private readonly Notifyer _notifyer;
        private readonly IRepository<User> _userRepository;

        public BittriexTask(
            IRepository<User> userRepository,
            Notifyer notifyer,
            HttpClientWrapper client,
            ILogger<BittriexTask> logger,
            CurrencyService currencyService,
            IRepository<Alert> alertRepository)
        {
            _userRepository = userRepository;
            _notifyer = notifyer;
            _client = client;
            _logger = logger;
            _currencyService = currencyService;
            _alertRepository = alertRepository;
        }

        public string Schedule => (10 * 1000).ToString();

        public async Task Invoke(CancellationToken cancellationToken)
        {
            _logger.LogInformation("Invoked timerbittrex");
            try
            {
                _logger.LogInformation(NoticeStore.NoticesBittrix.Count.ToString());
                _logger.LogInformation(CurrencyInfoStore.Bittrixes.Count.ToString());

                NoticeStore.ClearBittrix();

                var users = _userRepository.GetItems(x => x.NotifyBittrix).ToList();
                var currencyPairs = await _client.GetBittrix();

                CurrencyInfoStore.AddBittrix(currencyPairs);

                var notifications =
                    _currencyService.CreateNotifications(users,
CurrencyInfoStore.Bittrixes, 0, _alertRepository);
```

```
                await _notifyer.Notify(notifications);
            }
            catch (Exception e)
            {
                _logger.LogError("Timer bittriex", e.Message);
            }
        }
    }
    }
```

CoinMarketTask.cs

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using CryptoBot.Services;
using CryptoBot.Services.Stores;
using Microsoft.Extensions.Logging;

namespace CryptoBot.BackgroundTasks
{
    public class CoinMarketTask : IScheduledTask
    {
        private readonly HttpClientWrapper _client;
        private readonly CurrencyService _currencyService;
        private readonly ILogger<CoinMarketTask> _logger;
        private readonly IRepository<Alert> _alertRepository;
        private readonly Notifyer _notifyer;
        private readonly IRepository<User> _userRepository;

        public CoinMarketTask(IRepository<User> userRepository, Notifyer notifyer,
HttpClientWrapper client,
            CurrencyService currencyService,
            ILogger<CoinMarketTask> logger,
            IRepository<Alert> alertRepository )
        {
            _userRepository = userRepository;
            _notifyer = notifyer;
            _client = client;
            _currencyService = currencyService;
            _logger = logger;
            _alertRepository = alertRepository;
        }

        public string Schedule => (5 * 60 * 1000).ToString();

        public async Task Invoke(CancellationToken cancellationToken)
        {
            _logger.LogInformation("Invoked timercoinmarket");
            try
            {
                _logger.LogInformation(NoticeStore.NoticesCoinMarket.Count.ToString());
                _logger.LogInformation(CurrencyInfoStore.CoinMarkets.Count.ToString());

                NoticeStore.ClearCoinMarket();

                //_alertRepository.DeleteMany(x=> x.NotifyCost == null &&
string.IsNullOrEmpty(x.PercentNotify));
```

```
                var users = _userRepository.GetItems(z => z.NotifyCoinMarket).ToList();
//+

                var currencyPairs = await _client.GetCoinMarket();

                CurrencyInfoStore.AddCoinMarket(currencyPairs);

                var notifications = _currencyService
                    .CreateNotifications(users, CurrencyInfoStore.CoinMarkets, 2,
_alertRepository);

                await _notifyer.Notify(notifications);
            }
            catch (Exception e)
            {
                _logger.LogError("Timer coinmarket", e.Message);
            }
        }
    }
    }
```

PoloniexTask.cs

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using CryptoBot.Services;
using CryptoBot.Services.Stores;
using Microsoft.Extensions.Logging;

namespace CryptoBot.BackgroundTasks
{
    public class PoloniexTask : IScheduledTask
    {
        private readonly HttpClientWrapper _client;
        private readonly CurrencyService _currencyService;
        private readonly ILogger<PoloniexTask> _logger;
        private readonly Notifyer _notifyer;
        private readonly IRepository<User> _userRepository;
        private readonly IRepository<Alert> _alertRepository;

        public PoloniexTask(IRepository<User> userRepository,
                            IRepository<Alert> alertRepository,
                            Notifyer notifyer,
                            HttpClientWrapper client,
                            CurrencyService currencyService,
                            ILogger<PoloniexTask> logger)
        {
            _userRepository = userRepository;
            _alertRepository = alertRepository;
            _notifyer = notifyer;
            _client = client;
            _currencyService = currencyService;
            _logger = logger;
        }

        public string Schedule => (5 * 60 * 1000).ToString();

        public async Task Invoke(CancellationToken cancellationToken)
        {
```

```
            _logger.LogInformation("Invoked timerpoloniex");
            try
            {
                _logger.LogInformation(NoticeStore.NoticesPoloniex.Count.ToString());
                _logger.LogInformation(CurrencyInfoStore.Poloniexes.Count.ToString());

                NoticeStore.ClearPoloniex();

                var users = _userRepository.GetItems(x => x.NotifyPoloniex).ToList();

                var currencyPairs = await _client.GetPoloniex();

                CurrencyInfoStore.AddPoloniex(currencyPairs);

                var notifications = _currencyService
                    .CreateNotifications(users, CurrencyInfoStore.Poloniexes, 1,
_alertRepository);

                await _notifyer.Notify(notifications);
            }
            catch (Exception e)
            {
                _logger.LogError("Timer poloniex", e.Message);
            }
        }
    }
    }
```

DeleteAlertCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using Telegram.Bot;
using Telegram.Bot.Types;

namespace CryptoBot.Commands.Alerts
{
    public class DeleteAlertCommand:ITelegramCommand
    {
        public string Name => "deletealert";
        public bool IsProtected => true;
        private IRepository<Alert> _alertRepository;
        private ITelegramBotClient _client;

        public DeleteAlertCommand(IRepository<Alert> repository, ITelegramBotClient
client)
        {
            _client = client;
            _alertRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var splitStr = message.Text.Split(' ');
            if (splitStr.Length != 2)
            {
                await _client.SendTextMessageAsync(message.Chat.Id, $"Not successful
\nExample /{Name} BTC");
                return;
            }
            var currency = splitStr[1].ToUpper();
```

```
            var alert = _alertRepository.GetItem(z => z.UserId == message.Chat.Id &&
currency == z.NameCurrency);
            if (alert == null)
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "Alert not found");
                return;
            }
            _alertRepository.Delete(z=>z.UserId == message.Chat.Id && currency ==
z.NameCurrency);
            await _client.SendTextMessageAsync(message.Chat.Id, "Success");
        }
    }
    }
```

GetAlertCommand.cs

```
using System;
using System.Linq;
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using Telegram.Bot;
using Telegram.Bot.Types;

namespace CryptoBot.Commands.Alerts
{
    public class GetAlertCommand:ITelegramCommand
    {
        public string Name => "getalerts";
        public bool IsProtected => true;
        private IRepository<Alert> _alertRepository;
        private ITelegramBotClient _client;

        public GetAlertCommand(IRepository<Alert> repository, ITelegramBotClient client)
        {
            _alertRepository = repository;
            _client = client;
        }

        public async Task ExecuteAsync(Message message)
        {
            var allAllert = _alertRepository.GetItems(z => z.UserId == message.Chat.Id);
            if (allAllert == null || !allAllert.Any())
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "No one installed
Alert");
                return;
            }
            var report = String.Join(Environment.NewLine, allAllert);
            await _client.SendTextMessageAsync(message.Chat.Id, report);
        }
    }
    }
```

SetAlertCommand.cs

```
using System;
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
```

```
using CryptoBot.Services.Stores;
using Microsoft.Extensions.Logging;
using Telegram.Bot;
using Telegram.Bot.Types;

namespace CryptoBot.Commands.Alerts
{
    public class SetAlertCommand : ITelegramCommand
    {
        public string Name => "alert";
        public bool IsProtected => true;
        private IRepository<Alert> _alertRepository;
        private ITelegramBotClient _client;
        private ILogger<SetAlertCommand> _logger;

        public SetAlertCommand(IRepository<Alert> repository, ITelegramBotClient client,
ILogger<SetAlertCommand> logger)
        {
            _client = client;
            _alertRepository = repository;
            _logger = logger;
        }

        public async Task ExecuteAsync(Message message)
        {
            try
            {
                var splitStr = message.Text.Split(' ');
                var currency = splitStr[1].ToUpper();
                var value = splitStr[2].Replace(',','.');
                string sign = "";
                double valueParsed;
                if (value.Contains("+") || value.Contains("-"))
                {
                    valueParsed = double.Parse(value.Substring(1, value.Length -
2),CultureInfo.InvariantCulture);
                    sign += value[0];
                }
                else if (value.Contains("%"))
                {
                    valueParsed = double.Parse(value.Substring(0, value.Length -
1),CultureInfo.InvariantCulture);
                }
                else
                    valueParsed = double.Parse(value.Substring(0, value.Length),
CultureInfo.InvariantCulture);

                var alert = _alertRepository.GetItem(z => z.UserId == message.Chat.Id &&
currency == z.NameCurrency);

                if (value.Contains("%"))
                {
                    var lastCoinMarcet = CurrencyInfoStore.GetLastCoinMarket()
                        .FirstOrDefault(z => z.FirstCurrency == currency);
                    if (lastCoinMarcet == null)
                    {
                        await _client.SendTextMessageAsync(message.Chat.Id, "Currency not
found");
                        return;
                    }
                    if (alert == null)
                        _alertRepository.Create(new Alert
                        {
                            NameCurrency = currency,
```

```
                            PercentNotify = sign +
valueParsed.ToString(CultureInfo.InvariantCulture),
                            UserId = message.Chat.Id,
                            Cost = lastCoinMarcet.Last
                        });
                    else
                    {
                        alert.PercentNotify = sign +
valueParsed.ToString(CultureInfo.InvariantCulture);
                        alert.Cost = lastCoinMarcet.Last;
                        _alertRepository.Update(alert, z => z.UserId == message.Chat.Id
&& currency == z.NameCurrency);
                    }
                }
                else
                {
                    if (alert == null)
                        _alertRepository.Create(new Alert
                        {
                            NameCurrency = currency,
                            NotifyCost = valueParsed,
                            UserId = message.Chat.Id
                        });
                    else
                    {
                        alert.NotifyCost = valueParsed;
                        _alertRepository.Update(alert, z => z.UserId == message.Chat.Id
&& currency == z.NameCurrency);
                    }
                }
                await _client.SendTextMessageAsync(message.Chat.Id, "Success");
            }
            catch (Exception e)
            {
                await NotSuccess(message);
                _logger.LogError("Exeption in SetAlertCommand",e.Message);
            }
        }

        public async Task NotSuccess(Message message)
        {
            await _client.SendTextMessageAsync(message.Chat.Id, $"Not successful
\nExample /{Name} BTC 243/+10%");
        }
    }
    }
```

SetAlertCostCommand.cs

```
using System;
using System.Linq;
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using Telegram.Bot;
using Telegram.Bot.Types;

namespace CryptoBot.Commands.Alerts
{
    public class SetAlertCostCommand : ITelegramCommand
    {
        public string Name => "setalertcost";
        public bool IsProtected => true;
```

```
        private ITelegramBotClient _client;
        private IRepository<Alert> _alertRepository;

        public SetAlertCostCommand(ITelegramBotClient client, IRepository<Alert>
alertRepository)
        {
            _client = client;
            _alertRepository = alertRepository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var allAlert = _alertRepository.GetItems(z => z.UserId == message.Chat.Id);
            var splitStr = message.Text.TrimEnd().Split(' ');
            string cost;
            try
            {
                cost = TakeNumber(splitStr[2]);

            }
            catch (Exception e)
            {
                await _client.SendTextMessageAsync(message.Chat.Id,
                    "Not successful \nExample /setalertcost BTC 243");
                return;
            }
            if (splitStr.Length != 3 || splitStr[1].Length > 9 || cost == null)
            {
                await _client.SendTextMessageAsync(message.Chat.Id,
                    "Not successful \nExample /setalertcost BTC 243");
                return;
            }
            string pair = splitStr[1];//здесь делай ToUpper()
            var alert = allAlert?.FirstOrDefault(z => string.Equals(z.NameCurrency,
pair));
            if (allAlert == null || alert == null)
                _alertRepository.Create(new Alert{NameCurrency = pair,NotifyCost =
double.Parse(cost), UserId = message.Chat.Id});
            else
            {
                alert.NotifyCost = double.Parse(cost);
                _alertRepository.Update(alert,z=>z.UserId==message.Chat.Id &&
                                                string.Equals(z.NameCurrency, pair));
            }
            await _client.SendTextMessageAsync(message.Chat.Id, "Success");
        }

        private string TakeNumber(string str)
        {
            return new string(str.Where(char.IsDigit).ToArray());
        }
    }
    }
```

SetAlertPercentCommand.cs

```
using System;
using System.Linq;
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using Telegram.Bot;
```

```
using Telegram.Bot.Types;

namespace CryptoBot.Commands.Alerts
{
    public class SetAlertPercentCommand : ITelegramCommand
    {
        public string Name => "setalertpercent";
        public bool IsProtected => true;

        private ITelegramBotClient _client;
        private IRepository<Alert> _alertRepository;

        public SetAlertPercentCommand(ITelegramBotClient client,IRepository<Alert>
alertRepository)
        {
            _client = client;
            _alertRepository = alertRepository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var allAlerts = _alertRepository.GetItems(z => z.UserId == message.Chat.Id);
            var splitStr = message.Text.TrimEnd().Split(' ');
            string percent;
            try
            {
                percent = TakeNumber(splitStr[2]);
            }
            catch (Exception e)
            {
                await _client.SendTextMessageAsync(message.Chat.Id,
                    "Not successful \nExample /setalertpercent BTC +/-/nothing 10%");
                return;
            }
            if (splitStr.Length != 3 || splitStr[1].Length > 9 || percent == null)
            {
                await _client.SendTextMessageAsync(message.Chat.Id,
                    "Not successful \nExample /setalertpercent BTC +/-/nothing 10%");
                return;
            }
            string pair = splitStr[1];
            string percentForAlert;
            var alert = allAlerts?.FirstOrDefault(z => string.Equals(z.NameCurrency,
pair));
            if (splitStr[2][0] == '+' || splitStr[2][0] == '-')
            {
                percentForAlert = splitStr[2][0] + percent;
            }
            else
                percentForAlert = percent;
            if (allAlerts == null || alert == null)
                _alertRepository.Create(new Alert { NameCurrency = pair, PercentNotify =
percentForAlert, UserId = message.Chat.Id });
            else
            {
                alert.PercentNotify = percentForAlert;
                _alertRepository.Update(alert,z=>z.UserId == message.Chat.Id &&
                                                string.Equals(z.NameCurrency, pair));
            }
            await _client.SendTextMessageAsync(message.Chat.Id, "Success");
        }

        private string TakeNumber(string str)
        {
```

```
            return new string(str.Where(char.IsDigit).ToArray());
        }
    }
    }
```

GetAllPercentCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands.GetSettings
{
    public class GetAllPercentCommand:ITelegramCommand
    {
        public string Name => "getpercent";
        public bool IsProtected => true;

        private readonly ITelegramBotClient _client;
        private readonly IRepository<User> _userRepository;

        public GetAllPercentCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var user = _userRepository.GetItem(z => z.TelegramUserId == message.Chat.Id);
            string report = "You settings percent\n";
            report += $"Bittrix volume percent = {user.BittrixPercentVolume}%\n";
            report += $"Bittrix cost percent = {user.BittrixPercentChangeCost}%\n";
            report += $"Poloniex volume percent = {user.PoloniexPercentVolume}%\n";
            report += $"Poloniex cost percent = {user.PoloniexPercentChangeCost}%";
            await _client.SendTextMessageAsync(message.Chat.Id, report);
        }
    }
    }
```

GetTimeToCompareCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Microsoft.Azure.KeyVault.Models;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands.GetSettings
{
    public class GetTimeToCompareCommand:ITelegramCommand
    {
        public string Name => "gettimesettings";
        public bool IsProtected => true;

        private readonly ITelegramBotClient _client;
        private readonly IRepository<User> _userRepository;
```

```
        public GetTimeToCompareCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var user = _userRepository.GetItem(z => z.TelegramUserId == message.Chat.Id);
            string report = "Time to compare currency\n";
            report += $"Bittrix time = {user.TimeToCheckBittrix} min\n";
            report += $"Poloniex time = {user.TimeToCheckPoloniex} min";
            await _client.SendTextMessageAsync(message.Chat.Id, report);
        }
    }
    }
```

HardwareSettingsCommand.cs

```
using System;
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands.GetSettings
{
    public class HardwareSettingsCommand : ITelegramCommand
    {
        public string Name => "hw";
        public bool IsProtected => true;

        private ITelegramBotClient _client;
        private IRepository<User> _userRepository;

        public HardwareSettingsCommand(ITelegramBotClient client, IRepository<User> user)
        {
            _client = client;
            _userRepository = user;
        }

        public async Task ExecuteAsync(Message message)
        {
            string report = "`";
            var user = _userRepository.GetItem(z => z.TelegramUserId == message.Chat.Id);
            report += $"D3       {user.D3Cost}$" + Environment.NewLine;
            report += $"B8       {user.B8Cost}$" + Environment.NewLine;
            report += $"S9       {user.S9Cost}$" + Environment.NewLine;
            report += $"L3+      {user.L3PlusCost}$" + Environment.NewLine;
            report += $"Nv1070   {user.Nvidia1070}$" + Environment.NewLine;
            report += $"Nv1080Ti {user.Nvidia1080Ti}$`";
            await _client.SendTextMessageAsync(message.Chat.Id, report,
ParseMode.Markdown);
        }
    }
    }
```

B8Command.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;


namespace CryptoBot.Commands.HardwareCost
{
    public class B8Command : ITelegramCommand
    {
        public string Name => "b8";
        public bool IsProtected => true;

        private ITelegramBotClient _client;
        private IRepository<Models.DataBase.User> _user;

        public B8Command(ITelegramBotClient client, IRepository<Models.DataBase.User>
user)
        {
            _client = client;
            _user = user;
        }

        public async Task ExecuteAsync(Message message)
        {
            var cost = message.Text.Replace($"/{Name}", string.Empty).Trim();
            if (double.TryParse(cost, out var costResult))
            {
                var hardWare = _user.GetItem(z => z.TelegramUserId == message.Chat.Id);
                hardWare.B8Cost = costResult;
                _user.Update(hardWare, z => z.TelegramUserId == message.Chat.Id);
                await _client.SendTextMessageAsync(message.Chat.Id, "Success");
            }
            else
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "Invalid cost");
            }
        }
    }
    }
```

D3Command.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;


namespace CryptoBot.Commands.HardwareCost
{
    public class D3Command : ITelegramCommand
    {
        public string Name => "d3";
        public bool IsProtected => true;

        private ITelegramBotClient _client;
        private IRepository<Models.DataBase.User> _user;

        public D3Command(ITelegramBotClient client, IRepository<Models.DataBase.User>
user)
        {
            _client = client;
```

```
            _user = user;
        }

        public async Task ExecuteAsync(Message message)
        {
            var cost = message.Text.Replace($"/{Name}", string.Empty).Trim();
            if (double.TryParse(cost, out var costResult))
            {
                var hardWare = _user.GetItem(z => z.TelegramUserId == message.Chat.Id);
                hardWare.D3Cost = costResult;
                _user.Update(hardWare, z => z.TelegramUserId == message.Chat.Id);
                await _client.SendTextMessageAsync(message.Chat.Id, "Success");
            }
            else
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "Invalid cost");
            }
        }
    }
    }
```

L3PlusCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;


namespace CryptoBot.Commands.HardwareCost
{
    public class L3PlusCommand : ITelegramCommand
    {
        public string Name => "l3";
        public bool IsProtected => true;

        private ITelegramBotClient _client;
        private IRepository<Models.DataBase.User> _user;

        public L3PlusCommand(ITelegramBotClient client, IRepository<Models.DataBase.User>
user)
        {
            _client = client;
            _user = user;
        }

        public async Task ExecuteAsync(Message message)
        {
            var cost = message.Text.Replace($"/{Name}", string.Empty).Trim();
            if (double.TryParse(cost, out var costResult))
            {
                var hardWare = _user.GetItem(z => z.TelegramUserId == message.Chat.Id);
                hardWare.L3PlusCost = costResult;
                _user.Update(hardWare, z => z.TelegramUserId == message.Chat.Id);
                await _client.SendTextMessageAsync(message.Chat.Id, "Success");
            }
            else
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "Invalid cost");
            }
        }
    }
    }
```

Nvidia1070.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;


namespace CryptoBot.Commands.HardwareCost
{
    public class Nvidia1070 : ITelegramCommand
    {
        public string Name => "1070";
        public bool IsProtected => true;

        private ITelegramBotClient _client;
        private IRepository<Models.DataBase.User> _user;

        public Nvidia1070(ITelegramBotClient client, IRepository<Models.DataBase.User>
user)
        {
            _client = client;
            _user = user;
        }

        public async Task ExecuteAsync(Message message)
        {
            var cost = message.Text.Replace($"/{Name}", string.Empty).Trim();
            if (double.TryParse(cost, out var costResult))
            {
                var hardWare = _user.GetItem(z => z.TelegramUserId == message.Chat.Id);
                hardWare.Nvidia1070 = costResult;
                _user.Update(hardWare, z => z.TelegramUserId == message.Chat.Id);
                await _client.SendTextMessageAsync(message.Chat.Id, "Success");
            }
            else
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "Invalid cost");
            }
        }
    }
    }
```

Nvidia1080Ti.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;


namespace CryptoBot.Commands.HardwareCost
{
    public class Nvidia1080Ti : ITelegramCommand
    {
        public string Name => "1080ti";
        public bool IsProtected => true;

        private ITelegramBotClient _client;
        private IRepository<Models.DataBase.User> _user;
```

```
        public Nvidia1080Ti(ITelegramBotClient client, IRepository<Models.DataBase.User>
user)
        {
            _client = client;
            _user = user;
        }

        public async Task ExecuteAsync(Message message)
        {
            var cost = message.Text.Replace($"/{Name}", string.Empty).Trim();
            if (double.TryParse(cost, out var costResult))
            {
                var hardWare = _user.GetItem(z => z.TelegramUserId == message.Chat.Id);
                hardWare.Nvidia1080Ti = costResult;
                _user.Update(hardWare, z => z.TelegramUserId == message.Chat.Id);
                await _client.SendTextMessageAsync(message.Chat.Id, "Success");
            }
            else
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "Invalid cost");
            }
        }
    }
    }
```

S9Command.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;


namespace CryptoBot.Commands.HardwareCost
{
    public class S9Command:ITelegramCommand
    {
        public string Name => "s9";
        public bool IsProtected => true;

        private ITelegramBotClient _client;
        private IRepository<Models.DataBase.User> _user;

        public S9Command(ITelegramBotClient client, IRepository<Models.DataBase.User>
user)
        {
            _client = client;
            _user = user;
        }

        public async Task ExecuteAsync(Message message)
        {
            var cost = message.Text.Replace($"/{Name}",string.Empty).Trim();
            if (double.TryParse(cost, out var costResult))
            {
                var hardWare = _user.GetItem(z => z.TelegramUserId == message.Chat.Id);
                hardWare.S9Cost = costResult;
                _user.Update(hardWare,z=>z.TelegramUserId == message.Chat.Id);
                await _client.SendTextMessageAsync(message.Chat.Id, "Success");
            }
            else
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "Invalid cost");
```

```
            }
        }
    }
    }
```

NotifyBittrixCommand.cs

```
using System.Threading.Tasks;
using Telegram.Bot;
using Telegram.Bot.Types;
using Telegram.Bot.Types.InlineKeyboardButtons;
using Telegram.Bot.Types.ReplyMarkups;

namespace CryptoBot.Commands
{
    public class NotifyBittrixCommand:ITelegramCommand
    {
        private static ITelegramBotClient _client;

        public string Name => "notifybittrix";
        public bool IsProtected => true;

        public NotifyBittrixCommand(ITelegramBotClient client)
        {
            _client = client;
        }

        public async Task ExecuteAsync(Message message)
        {
            var keyboard = new InlineKeyboardMarkup(new InlineKeyboardButton[][]
            {
                new[] // first row
                {
                    new
InlineKeyboardCallbackButton("false",$"/{NotifyBittrixFalseCommand.NameCommand}"),
                    new
InlineKeyboardCallbackButton("true",$"/{NotifyBittrixTrueCommand.NameCommand}")
                }
            });
            await _client.SendTextMessageAsync(message.Chat.Id, "Показываем?",
replyMarkup: keyboard);
        }
    }
    }
```

NotifyBittrixFalseCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands
{
    public class NotifyBittrixFalseCommand:ITelegramCommand
    {
        public string Name => NameCommand;
        public static string NameCommand => "notbitFalse";
        public bool IsProtected => true;

        private ITelegramBotClient _client;
```

```
        private IRepository<User> _userRepository;

        public NotifyBittrixFalseCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var user = _userRepository.GetItem(z => z.TelegramUserId == message.Chat.Id);
            user.NotifyBittrix = false;
            _userRepository.Update(user,z=>z.TelegramUserId == message.Chat.Id);
            await _client.SendTextMessageAsync(message.Chat.Id, "Success");

        }
    }
    }
```

NotifyBittrixTrueCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands
{
    public class NotifyBittrixTrueCommand:ITelegramCommand
    {
        public string Name => NameCommand;
        public static string NameCommand => "notbitTrue";
        public bool IsProtected => false;
        private ITelegramBotClient _client;
        private IRepository<User> _userRepository;

        public NotifyBittrixTrueCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var user = _userRepository.GetItem(z => z.TelegramUserId == message.Chat.Id);
            user.NotifyBittrix = true;
            _userRepository.Update(user,z=>z.TelegramUserId==message.Chat.Id);
            await _client.SendTextMessageAsync(message.Chat.Id, "Success");
        }
    }
    }
```

NotifyCoinMarketCommand.cs

```
using System.Threading.Tasks;
using Telegram.Bot;
using Telegram.Bot.Types;
using Telegram.Bot.Types.InlineKeyboardButtons;
```

```
using Telegram.Bot.Types.ReplyMarkups;

namespace CryptoBot.Commands
{
    public class NotifyCoinMarketCommand:ITelegramCommand
    {
        public string Name => "notifycoinmarket";
        public bool IsProtected => true;
        private ITelegramBotClient _client;

        public NotifyCoinMarketCommand(ITelegramBotClient client)
        {
            _client = client;
        }

        public async Task ExecuteAsync(Message message)
        {
            var keyboard = new InlineKeyboardMarkup(new InlineKeyboardButton[][]
            {
                new[] // first row
                {
                    new
InlineKeyboardCallbackButton("true",$"/{NotifyCoinMarketTrueCommand.NameCommand}"),
                    new
InlineKeyboardCallbackButton("false",$"/{NotifyCoinMarketFalseCommand.NameCommand}")
                }
            });
            await _client.SendTextMessageAsync(message.Chat.Id, "Показываем?",
replyMarkup: keyboard);
        }
    }
    }
```

NotifyCoinMarketFalseCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands
{
    public class NotifyCoinMarketFalseCommand:ITelegramCommand
    {
        public static string NameCommand => "notcoinmarketfalse";
        public string Name => NameCommand;
        public bool IsProtected => true;
        private IRepository<User> _userRepository;
        private ITelegramBotClient _client;

        public NotifyCoinMarketFalseCommand(IRepository<User> repository,
ITelegramBotClient client)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var user = _userRepository.GetItem(z => z.TelegramUserId == message.Chat.Id);
            user.NotifyCoinMarket = false;
            _userRepository.Update(user, z => z.TelegramUserId== message.Chat.Id);
```

```
            await _client.SendTextMessageAsync(message.Chat.Id, "Success");
        }
    }
    }
```

NotifyCoinMarketTrueCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands
{
    public class NotifyCoinMarketTrueCommand:ITelegramCommand
    {
        public string Name => NameCommand;
        public static string NameCommand = "notcoinmarkettrue";
        public bool IsProtected => true;

        private IRepository<User> _userRepository;
        private ITelegramBotClient _client;

        public NotifyCoinMarketTrueCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var user = _userRepository.GetItem(z => z.TelegramUserId == message.Chat.Id);
            user.NotifyCoinMarket = true;
            _userRepository.Update(user,z=>z.TelegramUserId==message.Chat.Id);
            await _client.SendTextMessageAsync(message.Chat.Id, "Success");
        }
    }
    }
```

NotifyPoloniexCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;
using Telegram.Bot.Types.InlineKeyboardButtons;
using Telegram.Bot.Types.ReplyMarkups;

namespace CryptoBot.Commands
{
    public class NotifyPoloniexCommand:ITelegramCommand
    {
        private static ITelegramBotClient _client;

        public string Name => "notifypoloniex";
        public bool IsProtected => true;

        public NotifyPoloniexCommand(ITelegramBotClient client)
        {
```

```
            _client = client;
        }

        public async Task ExecuteAsync(Message message)
        {

            var keyboard = new InlineKeyboardMarkup(new InlineKeyboardButton[][]
            {
                new[] // first row
                {
                    new
InlineKeyboardCallbackButton("false",$"/{NotifyPoloniexFalseCommand.NameCommand}"),
                    new
InlineKeyboardCallbackButton("true",$"/{NotifyPoloniexTrueCommand.NameCommand}")
                }
            });
            await _client.SendTextMessageAsync(message.Chat.Id, "Показываем?",
replyMarkup: keyboard);

        }
    }
    }
```

NotifyPoloniexFalseCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands
{
    public class NotifyPoloniexFalseCommand:ITelegramCommand
    {
        public string Name => NameCommand;
        public static string NameCommand => "notpolFalse";
        public bool IsProtected => true;
        private ITelegramBotClient _client;
        private IRepository<User> _userRepository;

        public NotifyPoloniexFalseCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var user = _userRepository.GetItem(z => z.TelegramUserId == message.Chat.Id);
            user.NotifyPoloniex = false;
            _userRepository.Update(user,z=>z.TelegramUserId == message.Chat.Id);
            await _client.SendTextMessageAsync(message.Chat.Id, "Success");
        }
    }
    }
```

NotifyPoloniexTrueCommand.cs

```
using System;
using System.Threading.Tasks;
```

```
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands
{
    public class NotifyPoloniexTrueCommand:ITelegramCommand
    {
        public string Name => NameCommand;
        public static string NameCommand => "notpolTrue";
        public bool IsProtected => true;
        private ITelegramBotClient _client;
        private IRepository<User> _userRepository;

        public NotifyPoloniexTrueCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var user = _userRepository.GetItem(z => z.TelegramUserId == message.Chat.Id);
            user.NotifyPoloniex = true;
            _userRepository.Update(user,z=>z.TelegramUserId == message.Chat.Id);
            await _client.SendTextMessageAsync(message.Chat.Id, "Success");

        }
    }
    }
```

BittrixCostChangeCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands.PercentChange
{
    public class BittrixCostChangeCommand:ITelegramCommand
    {
        public string Name => "bittrixcostchange";
        public bool IsProtected => true;
        private readonly ITelegramBotClient _client;
        private readonly IRepository<User> _userRepository;

        public BittrixCostChangeCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var percentStr = message.Text.TrimEnd().Split(' ');
            if(percentStr.Length > 1)
                if (double.TryParse(percentStr[1], out var percent))
                {
```

```
                    var user = _userRepository.GetItem(z => z.TelegramUserId ==
message.Chat.Id);
                    user.BittrixPercentChangeCost = percent;
                    _userRepository.Update(user,z=>z.TelegramUserId == message.Chat.Id);
                    await _client.SendTextMessageAsync(message.Chat.Id, "Success");
                    return;
                }
            await _client.SendTextMessageAsync(message.Chat.Id, "Not successful \nExample
/bittrixCostChange 14,88");
        }
    }
    }
```

BittrixVolumeChangeCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands.PercentChange
{
    public class BittrixVolumeChangeCommand:ITelegramCommand
    {
        public string Name => "bittrixvolumechange";
        public bool IsProtected => true;
        private readonly ITelegramBotClient _client;
        private readonly IRepository<User> _userRepository;

        public BittrixVolumeChangeCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var percentStr = message.Text.TrimEnd().Split(' ');
            if(percentStr.Length >1)
                if (double.TryParse(percentStr[1], out var percent))
                {
                    var user = _userRepository.GetItem(z => z.TelegramUserId ==
message.Chat.Id);
                    user.BittrixPercentVolume = percent;
                    _userRepository.Update(user,z=>z.TelegramUserId == message.Chat.Id);
                    await _client.SendTextMessageAsync(message.Chat.Id, "Success");
                    return;
                }
            await _client.SendTextMessageAsync(message.Chat.Id, "Not successful \nExample
/bittrixVolumeChange 14,88");

        }
    }
    }
```

PoloniexCostChangeCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
```

```
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands.PercentChange
{
    public class PoloniexCostChangeCommand:ITelegramCommand
    {
        public string Name => "poloniexcostchange";
        public bool IsProtected => true;
        private readonly ITelegramBotClient _client;
        private readonly IRepository<User> _userRepository;

        public PoloniexCostChangeCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var percentStr = message.Text.TrimEnd().Split(' ');
            if (percentStr.Length > 1)
                if (double.TryParse(percentStr[1], out var percent))
                {
                    var user = _userRepository.GetItem(z => z.TelegramUserId ==
message.Chat.Id);
                    user.PoloniexPercentChangeCost = percent;
                    _userRepository.Update(user,z=>z.TelegramUserId==message.Chat.Id);
                    await _client.SendTextMessageAsync(message.Chat.Id, "Success");
                    return;
                }
            await _client.SendTextMessageAsync(message.Chat.Id, "Not successful \nExample
/poloniexCostChange 14,88");
        }
    }
    }
```

PoloniexVolumeChangeCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands.PercentChange
{
    public class PoloniexVolumeChangeCommand : ITelegramCommand
    {
        public string Name => "poloniexvolumechange";
        public bool IsProtected => true;
        private readonly ITelegramBotClient _client;
        private readonly IRepository<User> _userRepository;

        public PoloniexVolumeChangeCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
```

```
        {
            var percentStr = message.Text.TrimEnd().Split(' ');
            if (percentStr.Length > 1)
                if (double.TryParse(percentStr[1], out var percent))
                {
                    var user = _userRepository.GetItem(z => z.TelegramUserId ==
message.Chat.Id);
                    user.PoloniexPercentVolume = percent;
                    _userRepository.Update(user,z=>z.TelegramUserId==message.Chat.Id);
                    await _client.SendTextMessageAsync(message.Chat.Id, "Success");
                    return;
                }
            await _client.SendTextMessageAsync(message.Chat.Id, "Not successful \nExample
/poloniexVolumeChange 14,88");
        }
    }
    }
```

TimeToCompareBittrixCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands.TimeToCompare
{
    public class TimeToCompareBittrixCommand:ITelegramCommand
    {
        public string Name => "bittrixtimecompare";
        public bool IsProtected => true;

        private readonly ITelegramBotClient _client;
        private readonly IRepository<User> _userRepository;

        public TimeToCompareBittrixCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var timeStr = message.Text.TrimEnd().Split(' ');
            if (timeStr.Length > 1)
                if (double.TryParse(timeStr[1], out var time))
                {
                    var user = _userRepository.GetItem(z => z.TelegramUserId ==
message.Chat.Id);
                    user.TimeToCheckBittrix = time;
                    _userRepository.Update(user,z=>z.TelegramUserId==message.Chat.Id);
                    await _client.SendTextMessageAsync(message.Chat.Id, "Success");
                    return;
                }
            await _client.SendTextMessageAsync(message.Chat.Id, "Not successful \nExample
/bittrixTimeCompare 7");
        }
    }
    }
```

TimeToComparePoloniexCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands.TimeToCompare
{
    public class TimeToComparePoloniexCommand:ITelegramCommand
    {
        public string Name => "poloniextimecompare";
        public bool IsProtected => true;

        private readonly ITelegramBotClient _client;
        private readonly IRepository<User> _userRepository;

        public TimeToComparePoloniexCommand(ITelegramBotClient client, IRepository<User>
repository)
        {
            _client = client;
            _userRepository = repository;
        }

        public async Task ExecuteAsync(Message message)
        {
            var timeStr = message.Text.TrimEnd().Split(' ');
            if(timeStr.Length>1)
                if (double.TryParse(timeStr[1], out var time))
                {
                    var user = _userRepository.GetItem(z => z.TelegramUserId ==
message.Chat.Id);
                    user.TimeToCheckPoloniex = time;
                    _userRepository.Update(user,z=>z.TelegramUserId==message.Chat.Id);
                    await _client.SendTextMessageAsync(message.Chat.Id, "Success");
                    return;
                }
            await _client.SendTextMessageAsync(message.Chat.Id, "Not successful \nExample
/poloniexTimeCompare 7");
        }
    }
    }
```

UnconfirmedBtcCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Services;
using Telegram.Bot;
using Telegram.Bot.Types;

namespace CryptoBot.Commands.UnconfirmedTransactions
{
    public class UnconfirmedBtcCommand : ITelegramCommand
    {
        private readonly ITelegramBotClient _client;
        private readonly HttpClientWrapper _httpClientWrapper;
        public string Name => "unbtc";
        public bool IsProtected  => true;

        public UnconfirmedBtcCommand(ITelegramBotClient client, HttpClientWrapper
httpClientWrapper)
```

```
        {
            _client = client;
            _httpClientWrapper = httpClientWrapper;
        }

        public async Task ExecuteAsync(Message message)
        {
            var count = await _httpClientWrapper.GetUnconfirmedBtc();
            if (!string.IsNullOrEmpty(count))
                await _client.SendTextMessageAsync(message.Chat.Id, count);

        }
    }
    }
```

AsicCommand.cs

```
using System;
using System.Threading.Tasks;
using CryptoBot.Data;
using Telegram.Bot;
using Telegram.Bot.Types;

namespace CryptoBot.Commands.Webs
{
    public class AsicCommand:ITelegramCommand
    {
        public string Name => "asic";
        public bool IsProtected => true;

        private ITelegramBotClient _client;

        public AsicCommand(ITelegramBotClient client)
        {
            _client = client;
        }

        public async Task ExecuteAsync(Message message)
        {
            var url = Environment.GetEnvironmentVariable("HEROKU_URL");
            await _client.SendTextMessageAsync(message.Chat.Id,
$"{url}/asic/settings?={message.Chat.Id}");
        }
    }
    }
```

GetAsicCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using Telegram.Bot;
using Telegram.Bot.Types;

namespace CryptoBot.Commands.Webs
{
    public class GetAsicCommand:ITelegramCommand
    {
        public string Name => "getasic";
        public bool IsProtected => true;
        private IRepository<Asic> _asicRepository;
        private ITelegramBotClient _client;
```

```
        public GetAsicCommand(IRepository<Asic> repository, ITelegramBotClient client)
        {
            _client = client;
            _asicRepository = repository;
        }
        public async Task ExecuteAsync(Message message)
        {
            var asic = _asicRepository.GetItem(z => z.TelegramUserId == message.Chat.Id);
            await _client.SendTextMessageAsync(message.Chat.Id,
asic.Quark_HR.ToString());
        }
    }
    }
```

HelpCommand.cs

```
using System.Threading.Tasks;
using Telegram.Bot;
using Telegram.Bot.Types;

namespace CryptoBot.Commands
{
    public class HelpCommand : ITelegramCommand
    {
        private readonly ITelegramBotClient _client;

        public HelpCommand(ITelegramBotClient client)
        {
            _client = client;
        }

        public string Name => "help";
        public bool IsProtected => false;

        public async Task ExecuteAsync(Message message)
        {
            await _client.SendTextMessageAsync(message.Chat.Id, "help");
        }
    }
    }
```

InformationCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Services.Stores;
using Telegram.Bot;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;

namespace CryptoBot.Commands
{
    public class InformationCommand:ITelegramCommand
    {
        public static string NameCommand = "Information";
        public string Name => NameCommand;
        public bool IsProtected => true;

        private ITelegramBotClient _client;

        public InformationCommand(ITelegramBotClient client)
        {
```

```
            _client = client;
        }

        public async Task ExecuteAsync(Message message)
        {
            var currencyPairs = CurrencyInfoStore.GetLastCoinMarket();
            if (currencyPairs == null)
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "Some problemes");
                return;
            }
            foreach (var pair in currencyPairs)
            {
                if (string.Equals(pair.FirstCurrency,message.Text.ToUpper()))
                {
                    await _client.SendTextMessageAsync(message.Chat.Id,
pair.GetInfo(),ParseMode.Markdown);
                    return;
                }
            }
            await _client.SendTextMessageAsync(message.Chat.Id, "Currency not find");
        }
    }
    }
```

ITelegramCommand.cs

```
using System.Threading.Tasks;
using Telegram.Bot.Types;

namespace CryptoBot.Commands
{
    public interface ITelegramCommand
    {
        string Name { get; }
        bool IsProtected { get; }
        Task ExecuteAsync(Message message);
    }
    }
```

LoginCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Commands
{
    public class LoginCommand : ITelegramCommand
    {
        private readonly ITelegramBotClient _client;
        private readonly IRepository<User> _userRepository;
        private readonly IRepository<Models.DataBase.HardwareCost> _hwRepository;

        public LoginCommand(ITelegramBotClient client, IRepository<User> userRepository,
IRepository<Models.DataBase.HardwareCost> hwRepository)
        {
            _client = client;
            _userRepository = userRepository;
```

```
            _hwRepository = hwRepository;
        }

        public string Name => "login";
        public bool IsProtected => false;

        public async Task ExecuteAsync(Message message)
        {
            // security
            if (_userRepository.GetItem(x => x.TelegramUserId == message.Chat.Id) !=
null)
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "Invalid command!");
                return;
            }

            if (message.Text.Replace($"/{Name}", string.Empty).Trim() == "password")
            {
                _userRepository.Create(new User
                {
                    TelegramUserId = message.Chat.Id
                });

                //_hwRepository.Create(new Models.DataBase.HardwareCost()
                //{
                //    TelegramUserId = message.Chat.Id
                //});


                await _client.SendTextMessageAsync(message.Chat.Id, "Welcome!");

            }
            else
            {
                await _client.SendTextMessageAsync(message.Chat.Id, "Unknown password");
            }

        }
    }
    }
```

MainCommand.cs

```
using System.Threading.Tasks;
using CryptoBot.Services.Stores;
using Telegram.Bot;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;

namespace CryptoBot.Commands
{
    public class MainCommand:ITelegramCommand
    {
        public string Name => "main";
        public bool IsProtected => true;
        private ITelegramBotClient _client;

        public MainCommand(ITelegramBotClient client)
        {
            _client = client;
        }

        public async Task ExecuteAsync(Message message)
```

```
        {
            var report = CurrencyInfoStore.GetLastPoloneix();
            await _client.SendTextMessageAsync(message.Chat.Id,
report,ParseMode.Markdown);
        }
    }
    }
```

ProfitCommand.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using CryptoBot.Data;
using CryptoBot.Models;
using CryptoBot.Services;
using CryptoBot.Services.Stores;
using Telegram.Bot;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;

namespace CryptoBot.Commands
{
    public class ProfitCommand : ITelegramCommand
    {
        public string Name => "profit";
        public bool IsProtected => true;
        private ITelegramBotClient _client;
        private readonly ProfitMineService _profitMineService;
        private readonly IRepository<Models.DataBase.User> _hardwareRepository;

        public ProfitCommand(ITelegramBotClient client,ProfitMineService
profitMineService, IRepository<Models.DataBase.User> hardwareRepository)
        {
            _client = client;
            _profitMineService = profitMineService;
            _hardwareRepository = hardwareRepository;
        }

        public async Task ExecuteAsync(Message message)
        {
            //испавить этот костыль
            //if (_hardwareRepository.GetItem(x => x.TelegramUserId == message.Chat.Id)
== null)
            //{
            //    _hardwareRepository.Create(new Models.DataBase.HardwareCost()
            //    {
            //        TelegramUserId = message.Chat.Id
            //    });
            //    return;
            //}

            var hardwareCost = _hardwareRepository.GetItem(x => x.TelegramUserId ==
message.Chat.Id);
            _profitMineService.SetCost(hardwareCost);

            List<ReportProfit> report = new List<ReportProfit>();

            var s9 = await _profitMineService.GetProfit(ProfitMinerStore.minerS9);
            report.Add(new ReportProfit()
            {
                Name = "S9",
```

```
                Profit = s9
            });
            var l3 = await _profitMineService.GetProfit(ProfitMinerStore.minerL3);
            report.Add(new ReportProfit()
            {
                Name = "L3+",
                Profit = l3
            });
            var d3 = await _profitMineService.GetProfit(ProfitMinerStore.minerD3);
            report.Add(new ReportProfit()
            {
                Name = "D3",
                Profit = d3
            });

            var b8 = await _profitMineService.GetProfit(ProfitMinerStore.minerB8);
            report.Add(new ReportProfit()
            {
                Name = "B8",
                Profit = b8
            });
            var n70 = await _profitMineService.GetProfit(ProfitMinerStore.miner1070);
            report.Add(new ReportProfit()
            {
                Name = "1070",
                Profit = n70
            });
            var n80ti = await _profitMineService.GetProfit(ProfitMinerStore.miner1080ti);
            report.Add(new ReportProfit()
            {
                Name = "1080ti",
                Profit = n80ti
            });
            await _client.SendTextMessageAsync(message.Chat.Id,
'``'+string.Join(Environment.NewLine,report.Select(x=>x.ToString()))+'``',
ParseMode.Markdown);
        }
    }
    }
```

AsicController.cs

```
using CryptoBot.Data;
using CryptoBot.Models.DataBase;
using Microsoft.AspNetCore.Mvc;

namespace CryptoBot.Controllers
{
       /// <summary>
       /// Expecting url as site/?userId=123
       /// </summary>
       public class AsicController : Controller
       {
              private readonly IRepository<Asic> _asicRepository;

              public AsicController(IRepository<Asic> repository)
              {
                     _asicRepository = repository;
              }

              [HttpPost]
              public IActionResult Settings(Asic asic, [FromQuery]long userId)
              {
```

```
                    var oldAsic = _asicRepository.GetItem(z => z.TelegramUserId ==
asic.TelegramUserId);

                    if (oldAsic == null)
                    {
                        _asicRepository.Create(asic);
                    }
                    else
                    {
                        _asicRepository.Update(asic, z => z.TelegramUserId ==
asic.TelegramUserId);
                    }
                    return Ok();
                }
            }
        }
```

HealthCheckController.cs

```
using Microsoft.AspNetCore.Mvc;

namespace CryptoBot.Controllers
{
    [Route("api/[controller]")]
    public class HealthCheckController : ControllerBase
    {
        public IActionResult Check()
        {
            return Ok(new
            {
                Status = "ok"
            });
        }
    }
    }
```

WebhookController.cs

```
using System.Threading.Tasks;
using CryptoBot.Services;
using Microsoft.AspNetCore.Mvc;
using Telegram.Bot.Types;

namespace CryptoBot.Controllers
{
    [Route("api/[controller]")]
    public class WebhookController : ControllerBase
    {
        private readonly TelegramCommandService _telegramCommandService;

        public WebhookController(TelegramCommandService telegramCommandService)
        {
            _telegramCommandService = telegramCommandService;
        }

        [HttpPost]
        public async Task<IActionResult> Process([FromBody] Update update)
        {
            if (update.Message != null)
                await _telegramCommandService.Process(update.Message);
            else if(update.CallbackQuery != null)
                await _telegramCommandService.Process(update);
```

```
            return Ok();
        }
    }
    }
```

DataContext.cs

```
using CryptoBot.Models.DataBase;
using MongoDB.Bson;
using MongoDB.Bson.Serialization;
using MongoDB.Bson.Serialization.IdGenerators;
using MongoDB.Bson.Serialization.Serializers;
using MongoDB.Driver;

namespace CryptoBot.Data
{
    public class DataContext
    {
        private readonly IMongoDatabase _database;

        public DataContext(AppSettings settings)
        {

            var client = new MongoClient(settings.Db.ConnectionString);

            BsonClassMap.RegisterClassMap<User>(cm =>
            {
                cm.AutoMap();
                cm.MapIdMember(c => c.Id)
                    .SetIdGenerator(StringObjectIdGenerator.Instance)
                    .SetSerializer(new StringSerializer(BsonType.ObjectId));
            });

            BsonClassMap.RegisterClassMap<Alert>(cm =>
            {
                cm.AutoMap();
                cm.MapIdMember(c => c.Id)
                    .SetIdGenerator(StringObjectIdGenerator.Instance)
                    .SetSerializer(new StringSerializer(BsonType.ObjectId));
            });

            BsonClassMap.RegisterClassMap<Asic>(cm =>
            {
                cm.AutoMap();
                cm.MapIdMember(c => c.Id)
                    .SetIdGenerator(StringObjectIdGenerator.Instance)
                    .SetSerializer(new StringSerializer(BsonType.ObjectId));
            });

            BsonClassMap.RegisterClassMap<HardwareCost>(cm =>
            {
                cm.AutoMap();
                cm.MapIdMember(c => c.Id)
                    .SetIdGenerator(StringObjectIdGenerator.Instance)
                    .SetSerializer(new StringSerializer(BsonType.ObjectId));
            });

            _database = client.GetDatabase(settings.Db.DatabaseName);
        }

        public IMongoCollection<T> GetItems<T>() =>
_database.GetCollection<T>(typeof(T).Name);
    }
```

```
    }
```

IRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq.Expressions;

namespace CryptoBot.Data
{
    public interface IRepository<T>
    {
        IEnumerable<T> GetItems(Expression<Func<T, bool>> filter = null);
        T GetItem(Expression<Func<T, bool>> filter);
        void Create(T item);
        void Update(T item, Expression<Func<T, bool>> filter);
        void Delete(Expression<Func<T, bool>> filter);
        void DeleteMany(Expression<Func<T, bool>> filter);
    }
    }
```

MongoDbRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using MongoDB.Driver;

namespace CryptoBot.Data
{
    public class MongoDbRepository<T> : IRepository<T> where T : class
    {
        private readonly DataContext _db;

        public MongoDbRepository(DataContext dataContext)
        {
            _db = dataContext;
        }

        public IEnumerable<T> GetItems(Expression<Func<T, bool>> filter = null)
        {
            if (filter == null)
                return _db.GetItems<T>().Find(Builders<T>.Filter.Empty).ToList();

            return _db.GetItems<T>().Find(filter).ToList();
        }

        public void Create(T item)
        {
            _db.GetItems<T>().InsertOne(item);
        }


        public void Delete(Expression<Func<T, bool>> filter)
        {
            _db.GetItems<T>().FindOneAndDelete(filter);
        }

        public void DeleteMany(Expression<Func<T, bool>> filter)
        {
            _db.GetItems<T>().DeleteMany(filter);
        }
```

```
        public T GetItem(Expression<Func<T, bool>> filter)
        {
            return _db.GetItems<T>().Find(filter).FirstOrDefault();
        }

        public void Update(T item, Expression<Func<T, bool>> filter)
        {
            _db.GetItems<T>().ReplaceOne(filter, item);
        }
    }
    }
```

Bittrix.cs

```
namespace CryptoBot
{
    public class Bittrix
    {
        public bool success { get; set; }
        public string message { get; set; }
        public CurrencyData[] result { get; set; }
    }

    public class CurrencyData
    {
        public string MarketName { get; set; }
        public float High { get; set; }
        public float Low { get; set; }
        public float Volume { get; set; }
        public float Last { get; set; }
        public float BaseVolume { get; set; }
        //public DateTime TimeStamp { get; set; }
        //public float Bid { get; set; }
        /*public float Ask { get; set; }
        public int OpenBuyOrders { get; set; }
        public int OpenSellOrders { get; set; }
        public float PrevDay { get; set; }
        public DateTime Created { get; set; }*/
    }


    }
```

CoinMarket.cs

```
using Newtonsoft.Json;

namespace CryptoBot.Models.API.JSON
{
    public class CoinMarket
    {
        [JsonProperty("available_supply")]
        public string AvailableSupply { get; set; }

        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("last_updated")]
        public string LastUpdated { get; set; }

        [JsonProperty("market_cap_usd")]
        public string MarketCapUsd { get; set; }
```

```
        [JsonProperty("max_supply")]
        public string MaxSupply { get; set; }

        [JsonProperty("name")]
        public string Name { get; set; }

        [JsonProperty("percent_change_1h")]
        public string PercentChange1h { get; set; }//

        [JsonProperty("percent_change_24h")]
        public string PercentChange24h { get; set; }//

        [JsonProperty("percent_change_7d")]
        public string PercentChange7d { get; set; }//

        [JsonProperty("price_btc")]
        public string PriceBtc { get; set; }

        [JsonProperty("price_usd")]
        public string PriceUsd { get; set; }//

        [JsonProperty("rank")]
        public string Rank { get; set; }

        [JsonProperty("symbol")]
        public string Symbol { get; set; }//

        [JsonProperty("24h_volume_usd")]
        public string The24hVolumeUsd { get; set; }

        [JsonProperty("total_supply")]
        public string TotalSupply { get; set; }

    }
    }
```

Poloniex.cs

```
using Newtonsoft.Json;

namespace CryptoBot.Models.API.JSON
{
    public class Poloniex
    {
        [JsonProperty("baseVolume")]
        public string BaseVolume { get; set; }

        [JsonProperty("high24hr")]
        public string High24Hr { get; set; }

        [JsonProperty("last")]
        public string Last { get; set; }

        [JsonProperty("low24hr")]
        public string Low24Hr { get; set; }

        [JsonProperty("percentChange")]
        public string PercentChange { get; set; }

        [JsonProperty("quoteVolume")]
        public string QuoteVolume { get; set; }

        [JsonIgnore]
```

```
        public string CurrencyPair { get; set; }
    }
}
```

ProfitMine.cs

```
using Newtonsoft.Json;

namespace CryptoBot.Models.API.JSON
{
    public class ProfitMine
    {
        public int id { get; set; }

        [JsonProperty("name")]
        public string Name { get; set; }

        [JsonProperty("tag")]
        public string Tag { get; set; }

        [JsonProperty("algorithm")]
        public string Algorithm { get; set; }

        [JsonProperty("profit")]
        public string Profit { get; set; }

        [JsonProperty("lagging")]
        public string Lagging { get; set; }
    }

    }
```

CurrencyPair.cs

```
namespace CryptoBot.Models.API
{
    public class CurrencyPair
    {
        public string CurrencyPairName { get; set; }
        public string Name { get; set; }
        public double High { get; set; }
        public double Low { get; set; }
        public double Volume { get; set; }
        public double Last { get; set; }
        public double BaseVolume { get; set; }
        public double PercentChange { get; set; }
        public string FirstCurrency { get; set; }
        public string SecondCurrency { get; set; }

        public string GetInfo()
        {
            return $"{Name}({CurrencyPairName})\n"+
                $"*{Last:F2}$*\n"
                +$"_{PercentChange}%_";
        }
    }
    }
```

Alert.cs

```
namespace CryptoBot.Models.DataBase
{
```

```
    public class Alert
    {
        public string Id { get; set; }
        public long UserId { get; set; }
        public string NameCurrency { get; set; }
        public string PercentNotify { get; set; }
        public double? NotifyCost { get; set; } = null;
        public double? Cost { get; set; } = null;

        public string GetCostReport()
        {
            return NotifyCost == null ? "" : $"{NotifyCost}$";
        }

        public string GetPercentReport()
        {
            return PercentNotify == null ? "" : $"{PercentNotify}%";
        }

        public override string ToString()
        {
            return $"{NameCurrency} {GetCostReport()} {GetPercentReport()}";
        }
    }
    }
```

AllNotify.cs

```
using System;

namespace CryptoBot.Models.DataBase
{
    public class AllNotify
    {
        public long Id { get; set; }
        public DateTime TimeNotify { get; set; }
        public string Report { get; set; }
    }
    }
```

Asic.cs

```

namespace CryptoBot.Models.DataBase
{
    public class Asic
    {
        public string Id { get; set; }
        public long TelegramUserId { get; set; }
        public double Sha256_HR { get; set; }
        public double Sha256_P { get; set; }
        public double Scrypt_HR { get; set; }
        public double Scrypt_P { get; set; }
        public double X11_HR { get; set; }
        public double X11_P { get; set; }
        public double Quark_HR { get; set; }
        public double Quark_P { get; set; }
        public double Qubit_HR { get; set; }
        public double Qubit_P { get; set; }
        public double Cost { get; set; }
    }
    }
```

HardwareCost.cs

```
namespace CryptoBot.Models.DataBase
{
    public class HardwareCost
    {
        public string Id { get; set; }
        public long TelegramUserId { get; set; }
        public double S9Cost { get; set; } = 1000;
        public double L3PlusCost { get; set; } = 1000;
        public double D3Cost { get; set; } = 1000;
        public double B8Cost { get; set; } = 1000;
        public double Nvidia1070 { get; set; } = 1000;
        public double Nvidia1080Ti { get; set; } = 1000;
    }
    }
```

Users.cs

```
namespace CryptoBot.Models.DataBase
{
    public class Users
    {
        public string Id { get; set; }
        public long TelegramUserId { get; set; }
        public bool Authentication { get; set; }
        public bool NotifyBittrix { get; set; }
        public bool NotifyPoloniex { get; set; }
        public double TimeToCheckBittrix { get; set; } //с каким по времени в прошлом
сравнивать
        public double TimeToCheckPoloniex { get; set; }
        public double PoloniexPercentVolume { get; set; }
        public double PoloniexPercentChangeCost { get; set; }
        public double BittrixPercentVolume { get; set; }
        public double BittrixPercentChangeCost { get; set; }
    }

    public class User
    {
        public string Id { get; set; }
        public long TelegramUserId { get; set; }
        public bool NotifyBittrix { get; set; } = true;
        public bool NotifyPoloniex { get; set; } = true;
        public bool NotifyCoinMarket { get; set; } = true;
        // с каким по времени в прошлом сравнивать
        public double TimeToCheckBittrix { get; set; } = 5;
        public double TimeToCheckPoloniex { get; set; } = 5;
        public double PoloniexPercentVolume { get; set; } = 10;
        public double PoloniexPercentChangeCost { get; set; } = 10;
        public double BittrixPercentVolume { get; set; } = 10;
        public double BittrixPercentChangeCost { get; set; } = 10;
        public double S9Cost { get; set; } = 4500;
        public double L3PlusCost { get; set; } = 1500;
        public double D3Cost { get; set; } = 2700;
        public double B8Cost { get; set; } = 10000;
        public double Nvidia1070 { get; set; } = 4500;
        public double Nvidia1080Ti { get; set; } = 9000;
    }
    }
```

MinerInfo.cs

```
using System.Globalization;
```

```
namespace CryptoBot.Models
{
    public class MinerInfo
    {
        public string Name { get; set; }
        public string Id { get; set; }
        public string HashRate { get; set; }
        public string Power { get; set; }
        public string Cost { get; set; }
        public double HardwareCost { get; set; }
        public override string ToString()
        {
            var hardwareCost =
HardwareCost.ToString(CultureInfo.InvariantCulture).Replace(',','.');
            return
$"https://whattomine.com/coins/{Id}.json?&hr={HashRate}&p={Power}&fee=0.0&cost=0.06&hcost
={hardwareCost}&commit=Calculate";

        }
    }

    }
```

Notice.cs

```
using System;

namespace CryptoBot.Models
{
    public enum TypeNotice
    {
        Volume,
        Percent,
        Cost
    }
    public class Notice
    {
        public Notice(long uerdId, string name, TypeNotice type)
        {
            UserId = uerdId;
            Name = name;
            Type = type;
        }

        public long UserId { get; }
        public string Name { get; }
        public TypeNotice Type { get; }
        public DateTime Time { get; } = DateTime.UtcNow;
    }
    }
```

Notification.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace CryptoBot.Models
{
    public class Notification
    {
```

```
        public Notification(long chatId, IEnumerable<Report> reports)
        {
            ChatId = chatId;
            Reports = reports;
        }

        public override string ToString()
        {
            return string.Join(Environment.NewLine, Reports.Select(x => x.ToString()));
        }

        public long ChatId { get; }
        public IEnumerable<Report> Reports { get; }
    }
    }
```

Report.cs

```
namespace CryptoBot.Models
{
    public class Report
    {
        public string Text { get; set; }
        public string CurrencyPair { get; set; }
        public TypeNotice Type { get; set;}
        public override string ToString()
        {
            return $"{Text}";
        }
    }
    }
```

ReportProfit.cs

```
using System;

namespace CryptoBot.Models
{
    public class ReportProfit
    {
        public string Name { get; set; }
        public string Profit { get; set; }
        public override string ToString()
        {
            return Name + new String(' ', 8 - Name.Length) + Profit + " days";
        }
    }
    }
```

CurrencyInfoStore.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using CryptoBot.Models.API;

namespace CryptoBot.Services.Stores
{
    public static class CurrencyInfoStore
    {
```

```
        public static List<IEnumerable<CurrencyPair>> Poloniexes { get; } = new
List<IEnumerable<CurrencyPair>>();

        public static List<IEnumerable<CurrencyPair>> Bittrixes { get; } = new
List<IEnumerable<CurrencyPair>>();

        public static List<IEnumerable<CurrencyPair>> CoinMarkets { get; } = new
List<IEnumerable<CurrencyPair>>();


        public static void AddBittrix(IEnumerable<CurrencyPair> bittrix)
        {
            if (Bittrixes.Count >= 200)
                Bittrixes.RemoveAt(0);

            Bittrixes.Add(bittrix);
        }

        public static void AddPoloniex(IEnumerable<CurrencyPair> poloniex)
        {
            if (Poloniexes.Count >= 10)
                Poloniexes.RemoveAt(0);

            Poloniexes.Add(poloniex);
        }

        public static void AddCoinMarket(IEnumerable<CurrencyPair> coinMarket)
        {
            if (CoinMarkets.Count >= 10)
                CoinMarkets.RemoveAt(0);
            CoinMarkets.Add(coinMarket);
        }
        public static string GetLastPoloneix()
        {
            var lengthMaxName = Poloniexes.Last().Select(x =>
x.SecondCurrency.Length).Max();
            var lengthMaxLast = Poloniexes.Last().Select(x =>
$"{x.Last:F2}".Length).Max();
            var lengthMaxPercent = Poloniexes.Last().Select(x =>
$"{x.PercentChange:F2}".Length).Max();

            string FormatItem(CurrencyPair poloniex) =>
                "`"+$"{poloniex.SecondCurrency}" + new string(' ', 3 + lengthMaxName -
poloniex.SecondCurrency.Length) +
                $"{poloniex.Last:F2}$" + new string(' ', 3 + lengthMaxLast -
$"{poloniex.Last:F2}".Length) +
                $"{(100*poloniex.PercentChange):F2}%" + new string(' ', 3 +
lengthMaxPercent - $"{poloniex.PercentChange:F2}".Length)+"`";
            //+ $"{poloniex.BaseVolume:F2}";


            return string.Join(Environment.NewLine+ Environment.NewLine,
Poloniexes.Last().OrderByDescending(z=>z.BaseVolume).Select(x => FormatItem(x)));
        }

        public static IEnumerable<CurrencyPair> GetLastCoinMarket()
        {
            return CoinMarkets.LastOrDefault();
        }
    }
    }
```

NoticeStore.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using CryptoBot.Models;

namespace CryptoBot.Services.Stores
{
    public static class NoticeStore
    {
        private static List<Notice> _noticesBittrix = new List<Notice>();
        private static List<Notice> _noticesPoloniex = new List<Notice>();
        private static List<Notice> _noticesCoinMarket = new List<Notice>();

        public static List<Notice> NoticesBittrix => _noticesBittrix;
        public static List<Notice> NoticesPoloniex => _noticesPoloniex;
        public static List<Notice> NoticesCoinMarket => _noticesCoinMarket;

        public static void AddBittrix(Notice notice)
        {
            _noticesBittrix.Add(notice);
        }

        public static void ClearBittrix()
        {
            _noticesBittrix = _noticesBittrix
                .Where(x => (DateTime.UtcNow - x.Time).TotalSeconds <= 4 * 60)
                .ToList();
        }
        public static void AddPoloniex(Notice notice)
        {
            _noticesPoloniex.Add(notice);
        }

        public static void ClearPoloniex()
        {
            _noticesPoloniex = _noticesPoloniex
                .Where(x => (DateTime.UtcNow - x.Time).TotalSeconds <= 4 * 60)
                .ToList();
            //_noticesPoloniex = new List<Notice>();
        }

        public static void AddCoinMarket(Notice notice)
        {
            _noticesCoinMarket.Add(notice);
        }

        public static void ClearCoinMarket()
        {
            _noticesCoinMarket = _noticesCoinMarket
                .Where(x => (DateTime.UtcNow - x.Time).TotalSeconds <= 4 * 60)
                .ToList();
            //_noticesCoinMarket = new List<Notice>();
        }
        public static bool IsNotifiedPoloniex(long userId, string name, TypeNotice type)
        {
            return _noticesPoloniex.Any(x => x.Name == name && x.UserId == userId &&
x.Type == type);
        }

        public static bool IsNotifiedBittrix(long userId, string name, TypeNotice type)
        {
            return _noticesBittrix.Any(x => x.Name == name && x.UserId == userId &&
x.Type == type);
```

```
        }

        public static bool IsNotifiedCoinMarket(long userId, string name, TypeNotice
type)
        {
            return _noticesCoinMarket.Any(x => x.Name == name && x.UserId == userId &&
x.Type==type);
        }
    }
    }
```

ProfitMinerStore.cs

```
using CryptoBot.Models;

namespace CryptoBot.Services.Stores
{
    public static class ProfitMinerStore
    {
        public static MinerInfo minerS9 = new MinerInfo()
        {
            HashRate = "13500",
            Name = "S9",
            Id = "1",
            Power = "1500"
        };
        public static MinerInfo minerL3 = new MinerInfo()
        {
            HashRate = "504",
            Name = "L3+",
            Id = "4",
            Power = "1100"
        };
        public static MinerInfo minerD3 = new MinerInfo()
        {
            HashRate = "19000",
            Name = "D3",
            Id = "34",
            Power = "800"
        };
        public static MinerInfo minerB8 = new MinerInfo()
        {
            HashRate = "50000",
            Name = "B8",
            Id = "1",
            Power = "6300"
        };
        public static MinerInfo miner1070 = new MinerInfo()
        {
            HashRate = "188",
            Name = "1070",
            Id = "151",
            Power = "1000"
        };

        public static MinerInfo miner1080ti = new MinerInfo()
        {
            HashRate = "4500",
            Name = "1080ti",
            Id = "166",
            Power = "1650"
        };
    }
```

```
    }
```

CurrencyPairMapper.cs

```
using System.Globalization;
using CryptoBot.Models.API;
using CryptoBot.Models.API.JSON;

namespace CryptoBot.Services
{
    public static class CurrencyPairMapper
    {
        public static CurrencyPair Map(CurrencyData currencyData)
        {
            var names = currencyData.MarketName.Trim().Split('-');

            return new CurrencyPair
            {
                BaseVolume = currencyData.BaseVolume,
                High = currencyData.High,
                Last = currencyData.Last,
                Low = currencyData.Low,
                CurrencyPairName = currencyData.MarketName,
                Volume = currencyData.Volume,
                PercentChange = 0,
                FirstCurrency = names[0],
                SecondCurrency = names[1]
            };
        }

        public static CurrencyPair Map(Poloniex poloniex)
        {
            var names = poloniex.CurrencyPair.Trim().Split('_');

            return new CurrencyPair
            {
                BaseVolume = double.Parse(poloniex.BaseVolume,
CultureInfo.InvariantCulture),
                CurrencyPairName = poloniex.CurrencyPair,
                High = double.Parse(poloniex.High24Hr, CultureInfo.InvariantCulture),
                Last = double.Parse(poloniex.Last, CultureInfo.InvariantCulture),
                Low = double.Parse(poloniex.Low24Hr, CultureInfo.InvariantCulture),
                Volume = double.Parse(poloniex.QuoteVolume,
CultureInfo.InvariantCulture),
                PercentChange = double.Parse(poloniex.PercentChange,
CultureInfo.InvariantCulture),
                FirstCurrency = names[0],
                SecondCurrency = names[1]
            };
        }

        public static CurrencyPair Map(CoinMarket coinMarket)
        {
            return new CurrencyPair
            {
                BaseVolume =
double.Parse(coinMarket.The24hVolumeUsd,CultureInfo.InvariantCulture),
                CurrencyPairName = coinMarket.Symbol,
                Last = double.Parse(coinMarket.PriceUsd, CultureInfo.InvariantCulture),
                High = 0,
                Low = 0,
                FirstCurrency = coinMarket.Symbol,
                SecondCurrency = "USD",
```

```
                Volume = 0,
                PercentChange = double.Parse(coinMarket.PercentChange24h,
CultureInfo.InvariantCulture),
                Name = coinMarket.Name
            };

        }
    }
    }
```

CurrencyService.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using CryptoBot.Data;
using CryptoBot.Models;
using CryptoBot.Models.API;
using CryptoBot.Models.DataBase;
using CryptoBot.Services.Stores;

namespace CryptoBot.Services
{
    public class CurrencyService
    {
        public List<Notification> CreateNotifications(IEnumerable<User> users,
List<IEnumerable<CurrencyPair>> currencyPairs, int type, IRepository<Alert> repository)
        {
            var notifications = new List<Notification>();

            foreach (var user in users)
            {
                var reports = new List<Report>();
                switch (type)
                {
                    case 0:
                        {
                            reports = CreateBittrixReports(user, currencyPairs);
                            break;
                        }
                    case 1:
                        {
                            reports = CreatePoloniexReports(user, currencyPairs);
                            break;
                        }
                    case 2:
                        {
                            reports = CreateCoinMarketRepots(user, currencyPairs,
repository);
                            break;
                        }
                }

                if (reports.Any())
                {
                    notifications.Add(new Notification(user.TelegramUserId, reports));
                }
            }

            return notifications;
        }
```

```
        private List<Report> CreateCoinMarketRepots(User user,
List<IEnumerable<CurrencyPair>> currencyPairs,
            IRepository<Alert> _alertRepository)
        {
            var reports = new List<Report>();
            if (currencyPairs.Count == 1)
            {
                return reports;
            }
            var alerts = _alertRepository.GetItems(x => x.UserId == user.TelegramUserId);
            var oldValues = currencyPairs[currencyPairs.Count - 2];
            var newValues = currencyPairs.Last();

            foreach (var value in newValues)
            {
                if (
                    alerts.Any(
                        x => string.Equals(x.NameCurrency, value.CurrencyPairName,
StringComparison.OrdinalIgnoreCase)))
                {
                    var alert = alerts.FirstOrDefault(x => x.NameCurrency ==
value.CurrencyPairName);
                    var percent = alert.PercentNotify;
                    var cost = alert.NotifyCost;
                    var oldCurrnecy = oldValues.FirstOrDefault(x => x.CurrencyPairName ==
value.CurrencyPairName);
                    if (!string.IsNullOrEmpty(percent))
                    {
                        var report = CheckPercent(alert, value);
                        if (report != null &&
                            !NoticeStore.IsNotifiedCoinMarket(user.TelegramUserId,
value.CurrencyPairName,
                                TypeNotice.Percent))
                        {
                            reports.Add(report);
                            alert.PercentNotify = null;
                        }
                    }
                    if (cost.HasValue)
                    {
                        var report = CheckCost(value, oldCurrnecy, cost.Value);
                        if (report != null &&
                            !NoticeStore.IsNotifiedCoinMarket(user.TelegramUserId,
value.CurrencyPairName,
                                TypeNotice.Cost))
                        {
                            reports.Add(report);
                            alert.NotifyCost = null;

                        }
                    }
                    if (alert.PercentNotify == null && alert.NotifyCost == null)
                        _alertRepository.Delete(z=>z.UserId == user.TelegramUserId &&
z.NameCurrency == value.FirstCurrency);
                    else
                        _alertRepository.Update(alert, x => x.UserId ==
user.TelegramUserId && x.NameCurrency == value.FirstCurrency);

                }
                if (reports.Any(x => x.CurrencyPair == value.CurrencyPairName && x.Type
== TypeNotice.Percent))
                    NoticeStore.AddCoinMarket(new Notice(user.TelegramUserId,
value.CurrencyPairName, TypeNotice.Percent));
```

```
                if (reports.Any(x => x.CurrencyPair == value.CurrencyPairName && x.Type
== TypeNotice.Cost))
                    NoticeStore.AddCoinMarket(new Notice(user.TelegramUserId,
value.CurrencyPairName, TypeNotice.Cost));
            }
            return reports;
        }

        private Report CheckPercent(Alert alert, CurrencyPair currency)
        {
            var percent = alert.PercentNotify;
            switch (percent[0])
            {
                case '-':
                    {
                        var p = double.Parse(percent, CultureInfo.InvariantCulture);
                        if ( alert.Cost != null && 100*(1 - currency.Last /
alert.Cost.Value) > p)
                            return new Report
                            {
                                Type = TypeNotice.Percent,
                                CurrencyPair = currency.CurrencyPairName,
                                Text = $"Курс {currency.CurrencyPairName} изменился на -
{1 - currency.Last / alert.Cost.Value:F2}%"
                            };
                        break;
                    }
                case '+':
                    {
                        var p = double.Parse(percent.Substring(1),
CultureInfo.InvariantCulture);
                        if (alert.Cost != null && 100*(currency.Last / alert.Cost.Value -
1)  > p)
                            return new Report
                            {
                                Type = TypeNotice.Percent,
                                CurrencyPair = currency.CurrencyPairName,
                                Text = $"Курс {currency.CurrencyPairName} изменился на
+{currency.Last / alert.Cost.Value - 1:F2}%"
                            };
                        break;
                    }
                default:
                    {
                        var p = double.Parse(percent, CultureInfo.InvariantCulture);
                        if (alert.Cost != null)
                        {
                            var abs = Math.Abs(currency.Last / alert.Cost.Value - 1)*100;
                            var sign = currency.Last / alert.Cost.Value - 1 > 0 ? '+' :
'-';
                            if (abs > p)
                                return new Report
                                {
                                    Type = TypeNotice.Percent,
                                    CurrencyPair = currency.CurrencyPairName,
                                    Text = $"Курс {currency.CurrencyPairName} изменился
на {sign}{abs:F2}%"
                                };
                        }
                        break;
                    }
            }
            return null;
        }
```

```
        private Report CheckCost(CurrencyPair newValue, CurrencyPair oldValue, double
cost)
        {
            if ((newValue.Last - cost) * (oldValue.Last - cost) < 0)
            {
                return new Report
                {
                    Type = TypeNotice.Cost,
                    CurrencyPair = newValue.CurrencyPairName,
                    Text = $"{newValue.CurrencyPairName} преодолел порог в {cost:F2}
usd",
                };
            }
            return null;
        }
        private List<Report> CreateBittrixReports(User user,
List<IEnumerable<CurrencyPair>> currencyPairs)
        {
            var userTimeToCheckBittrix = user.TimeToCheckBittrix;
            var userNumberChat = user.TelegramUserId;
            var userBittrixPercentChangeCost = user.BittrixPercentChangeCost;
            var userBittrixPercentVolume = user.BittrixPercentVolume;

            var reports = new List<Report>();
            int ind = (int)Math.Round(userTimeToCheckBittrix * 5);
            var oldValues = currencyPairs.Count <= ind
                ? currencyPairs[0]
                : currencyPairs[currencyPairs.Count - ind - 1];

            var newValues = currencyPairs.Last().ToList();

            foreach (var newValue in newValues)
            {
                var oldValue = oldValues
                    .Where(x => x.CurrencyPairName == newValue.CurrencyPairName)
                    .ToList();

                if (oldValue.Any())
                {
                    var percent = (newValue.Last / oldValue.First().Last - 1) * 100;
                    if (Math.Abs(percent) > userBittrixPercentChangeCost)
                    {
                        var sign = percent >= 0 ? '+' : '-';
                        var value = Math.Abs(percent);
                        if (!NoticeStore.IsNotifiedBittrix(user.TelegramUserId,
newValue.CurrencyPairName, TypeNotice.Percent))
                        {
                            reports.Add(new Report
                            {
                                Type = TypeNotice.Percent,
                                Text = $"Курс {newValue.CurrencyPairName} изменился на
{sign}{value:F2}%",
                                CurrencyPair = newValue.CurrencyPairName
                            });
                        }
                    }
                    percent = (newValue.BaseVolume / oldValue.First().BaseVolume - 1) *
100;
                    if (percent > userBittrixPercentVolume)
                    {
                        var sign = percent >= 0 ? '+' : '-';
                        var value = Math.Abs(percent);
```

```
                        if (!NoticeStore.IsNotifiedBittrix(user.TelegramUserId,
newValue.CurrencyPairName, TypeNotice.Volume))
                        {
                            reports.Add(new Report
                            {
                                Type = TypeNotice.Volume,
                                Text = $"Объем {newValue.CurrencyPairName} изменился на
{sign}{value:F2}%",
                                CurrencyPair = newValue.CurrencyPairName
                            });
                        }
                    }

                    if (reports.Any(x => x.CurrencyPair == newValue.CurrencyPairName &&
x.Type==TypeNotice.Percent))
                        NoticeStore.AddBittrix(new Notice(userNumberChat,
newValue.CurrencyPairName,TypeNotice.Percent));
                    if (reports.Any(x => x.CurrencyPair == newValue.CurrencyPairName &&
x.Type == TypeNotice.Volume))
                        NoticeStore.AddBittrix(new Notice(userNumberChat,
newValue.CurrencyPairName, TypeNotice.Volume));
                }
            }
            return reports;
        }

        private List<Report> CreatePoloniexReports(User user,
List<IEnumerable<CurrencyPair>> currencyPairs)
        {
            var userTimeToCheckPoloniex = user.TimeToCheckPoloniex;
            var userNumberChat = user.TelegramUserId;
            var userPoloniexPercentChangeCost = user.PoloniexPercentChangeCost;

            var reports = new List<Report>();
            int ind = (int)Math.Round(userTimeToCheckPoloniex / 5);
            var oldValues = currencyPairs.Count <= ind
                ? currencyPairs[0].ToList()
                : currencyPairs[currencyPairs.Count - ind - 1].ToList();

            var newValues = currencyPairs.Last().ToList();

            foreach (var newValue in newValues)
            {
                var oldValue = oldValues
                    .Where(x => x.CurrencyPairName == newValue.CurrencyPairName)
                    .ToList();
                if (oldValue.Any())
                {
                    var percent = (newValue.Last/oldValue.First().Last -1 ) * 100;
                    if (Math.Abs(percent) > userPoloniexPercentChangeCost)
                    {
                        var sign = percent >= 0 ? '+' : '-';
                        var value = Math.Abs(percent);
                        if (!NoticeStore.IsNotifiedPoloniex(user.TelegramUserId,
newValue.CurrencyPairName, TypeNotice.Percent))
                        {
                            reports.Add(new Report
                            {
                                Type = TypeNotice.Percent,
                                Text = $"Курс {newValue.CurrencyPairName} изменился на
{sign}{value:F2}%",
                                CurrencyPair = newValue.CurrencyPairName
                            });
                        }
```

```
                    }
                    if (oldValues.Count == newValues.Count)
                    {
                        var sortedOld = oldValues.OrderBy(x => x.BaseVolume).ToList();
                        var indOld = sortedOld.FindIndex(x=> x.CurrencyPairName ==
newValue.CurrencyPairName);

                        var sortedNew = newValues.OrderBy(x => x.BaseVolume).ToList();
                        var indNew = sortedNew.FindIndex(x => x.CurrencyPairName ==
newValue.CurrencyPairName);
                        if (!NoticeStore.IsNotifiedPoloniex(user.TelegramUserId,
newValue.CurrencyPairName, TypeNotice.Volume) && indNew - indOld >= 2)
                        {
                            reports.Add(new Report
                            {
                                Type = TypeNotice.Volume,
                                Text = $"{newValue.SecondCurrency} поднялся на {indNew -
indOld} поз.",
                                CurrencyPair = newValue.CurrencyPairName
                            });
                        }
                    }

                    if (reports.Any(x => x.CurrencyPair == newValue.CurrencyPairName &&
x.Type == TypeNotice.Percent))
                        NoticeStore.AddPoloniex(new Notice(userNumberChat,
newValue.CurrencyPairName, TypeNotice.Percent));
                    if (reports.Any(x => x.CurrencyPair == newValue.CurrencyPairName &&
x.Type == TypeNotice.Volume))
                        NoticeStore.AddPoloniex(new Notice(userNumberChat,
newValue.CurrencyPairName, TypeNotice.Volume));
                }
            }
            return reports;
        }
    }
}
```

HttpClientWrapper.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using CryptoBot.Models;
using CryptoBot.Models.API;
using CryptoBot.Models.API.JSON;
using Newtonsoft.Json;

namespace CryptoBot.Services
{
    public class HttpClientWrapper
    {
        private readonly HttpClient _client;

        public HttpClientWrapper()
        {
            _client = new HttpClient();
        }
```

```
        private async Task<T> GetAsync<T>(string url)
        {
            var responce = await _client.GetAsync(url);
            var content = await responce.Content.ReadAsStringAsync();

            return JsonConvert.DeserializeObject<T>(content);
        }

        public async Task<IEnumerable<CurrencyPair>> GetBittrix()
        {
            var currency = await
GetAsync<Bittrix>("https://bittrex.com/api/v1.1/public/getmarketsummaries");

            return currency.result
                .Where(z => z.MarketName.Contains("BTC-"))
                .ToArray().Select(CurrencyPairMapper.Map);
        }

        public async Task<IEnumerable<CurrencyPair>> GetPoloniex()
        {
            var url = "https://poloniex.com/public?command=returnTicker";
            var currencyPairs = await GetAsync<Dictionary<string, Poloniex>>(url);

            return currencyPairs
                .Where(z => z.Key.Contains("USDT_"))
                .Select(x =>
                {
                    x.Value.CurrencyPair = x.Key;
                    return x.Value;
                })
                .Select(CurrencyPairMapper.Map);
        }

        public async Task<IEnumerable<CurrencyPair>> GetCoinMarket()
        {
            var url = "https://api.coinmarketcap.com/v1/ticker/?limit=100";
            var currencyPairs = await GetAsync<IEnumerable<CoinMarket>>(url);

            return currencyPairs.Select(CurrencyPairMapper.Map);
        }

        public async Task<string> GetUnconfirmedBtc()
        {
            var response = await _client.GetAsync("https://blockchain.info/unconfirmed-
transactions");
            var stream = await response.Content.ReadAsStreamAsync();
            using (var reader = new StreamReader(stream))
            {
                while (!reader.EndOfStream)
                {
                    var s = reader.ReadLine();
                    if (s.Contains("Unconfirmed"))
                        return new string(s.Where(char.IsDigit).ToArray());
                }
            }
            return "";
        }

        public async Task<double> GetBreakEvenIn(MinerInfo minerInfo)
        {
            var profitMine = await GetAsync<ProfitMine>(minerInfo.ToString());
            var profit = Double.Parse(profitMine.Profit.Replace("$",""),
CultureInfo.InvariantCulture);
```

```
            return minerInfo.HardwareCost / profit;

        }
    }
    }
```

Notifyer.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using CryptoBot.Models;
using Telegram.Bot;

namespace CryptoBot.Services
{
    public class Notifyer
    {
        private readonly ITelegramBotClient _telegramBotClient;

        public Notifyer(ITelegramBotClient telegramBotClient)
        {
            _telegramBotClient = telegramBotClient;
        }

        public async Task Notify(Notification notification)
        {
            await _telegramBotClient
                .SendTextMessageAsync(notification.ChatId, notification.ToString());
        }

        public async Task Notify(IEnumerable<Notification> notifications)
        {
            foreach (var notification in notifications)
                await _telegramBotClient
                    .SendTextMessageAsync(notification.ChatId, notification.ToString());
        }
    }
    }
```

ProfitMineService.cs

```
using System.Threading.Tasks;
using CryptoBot.Models;
using CryptoBot.Models.DataBase;
using CryptoBot.Services.Stores;

namespace CryptoBot.Services
{
    public class ProfitMineService
    {
        private readonly HttpClientWrapper _client;

        public ProfitMineService(HttpClientWrapper client)
        {
            _client = client;
        }

        public void SetCost(User user)
        {
            ProfitMinerStore.miner1070.HardwareCost = user.Nvidia1070;
            ProfitMinerStore.miner1080ti.HardwareCost = user.Nvidia1080Ti;
            ProfitMinerStore.minerS9.HardwareCost = user.S9Cost;
```

```
            ProfitMinerStore.minerL3.HardwareCost = user.L3PlusCost;
            ProfitMinerStore.minerD3.HardwareCost = user.D3Cost;
            ProfitMinerStore.minerB8.HardwareCost = user.B8Cost;
        }

        public async Task<string> GetProfit(MinerInfo miner)
        {

            var profit =  await _client.GetBreakEvenIn(miner);
            return $"{profit:F2}";
        }
    }
    }
```

TelegramCommandService.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using CryptoBot.Commands;
using CryptoBot.Data;
using Microsoft.Extensions.Logging;
using Telegram.Bot;
using Telegram.Bot.Types;
using User = CryptoBot.Models.DataBase.User;

namespace CryptoBot.Services
{
    public class TelegramCommandService
    {
        private readonly IEnumerable<ITelegramCommand> _commands;
        private readonly IRepository<User> _userRepository;
        private readonly ILogger<TelegramCommandService> _logger;
        private readonly ITelegramBotClient _client;

        public TelegramCommandService(
            IEnumerable<ITelegramCommand> commands,
            IRepository<User> userRepository,
            ILogger<TelegramCommandService> logger,
            ITelegramBotClient client
        )
        {
            _commands = commands;
            _userRepository = userRepository;
            _logger = logger;
            _client = client;
        }

        public async Task Process(Message message)
        {
            var user = _userRepository
                .GetItem(x => x.TelegramUserId == message.Chat.Id);

            var command = _commands
                .SingleOrDefault(x =>  message.Text.Contains($"/{x.Name}"));



            if (command != null)
            {
                if (command.IsProtected && user == null)
                {
```

```
                    await _client.SendTextMessageAsync(message.Chat.Id, "Sign in
please");
                    return;
                }
                _logger.LogInformation("Process command", command.Name, message.Chat.Id);

                await command.ExecuteAsync(message);
            }
            else if (!message.Text.Contains("/"))
            {
                if (user == null)
                {
                    await _client.SendTextMessageAsync(message.Chat.Id, "Sign in
please");
                    return;
                }
                _logger.LogInformation("Process command", "information",
message.Chat.Id);

                _commands.FirstOrDefault(z => z.Name ==
InformationCommand.NameCommand)?.ExecuteAsync(message);
            }
            else
            {
                _logger.LogInformation("Command not found", message.Chat.Id);
                await _client.SendTextMessageAsync(message.Chat.Id, "Command not found");
            }

        }

        public async Task Process(Update update)
        {
            var callbackCommand = update.CallbackQuery.Data;
            var user = _userRepository
                .GetItems(z => z.TelegramUserId == update.CallbackQuery.Message.Chat.Id);
            var command = _commands
                .SingleOrDefault(
                    z => (user != null || z.IsProtected == false) &&
callbackCommand.Contains($"/{z.Name}"));
            if (command != null)
            {
                _logger.LogInformation("Process command", command.Name,
update.CallbackQuery.Message.Chat.Id);
                await command.ExecuteAsync(update.CallbackQuery.Message);
            }
        }

    }
    }
```

HostedService.cs

```
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Extensions.Hosting;

namespace CryptoBot
{
    // https://blogs.msdn.microsoft.com/cesardelatorre/2017/11/18/implementing-
background-tasks-in-microservices-with-ihostedservice-and-the-backgroundservice-class-
net-core-2-x/
    public abstract class HostedService : IHostedService
    {
```

```
        private readonly CancellationTokenSource _stoppingCts = new
CancellationTokenSource();
        private Task _executingTask;

        public virtual Task StartAsync(CancellationToken cancellationToken)
        {
            // Store the task we're executing
            _executingTask = ExecuteAsync(_stoppingCts.Token);

            // If the task is completed then return it,
            // this will bubble cancellation and failure to the caller
            if (_executingTask.IsCompleted)
                return _executingTask;

            // Otherwise it's running
            return Task.CompletedTask;
        }

        public virtual async Task StopAsync(CancellationToken cancellationToken)
        {
            // Stop called without start
            if (_executingTask == null)
                return;

            try
            {
                // Signal cancellation to the executing method
                _stoppingCts.Cancel();
            }
            finally
            {
                // Wait until the task completes or the stop token triggers
                await Task.WhenAny(_executingTask, Task.Delay(Timeout.Infinite,
                    cancellationToken));
            }
        }

        protected abstract Task ExecuteAsync(CancellationToken stoppingToken);

        public virtual void Dispose()
        {
            _stoppingCts.Cancel();
        }
    }
    }
```

IScheduledTask.cs

```
namespace CryptoBot
{
    public interface IScheduledTask
    {
        string Schedule { get; }
    }
    }
```

Program.cs

```
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
```

```
namespace CryptoBot
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .ConfigureServices((context, services) =>
                {
                    var appSettings = new AppSettings();
                    context.Configuration.Bind(appSettings);
                    services.AddSingleton(appSettings);
                })
                .UseStartup<Startup>()
                .Build();
    }
    }
```

SchedulerHostedService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

namespace CryptoBot
{
       public class SchedulerHostedService : HostedService
       {
           private readonly ILogger<SchedulerHostedService> _logger;
           private readonly List<SchedulerTaskWrapper> _scheduledTasks = new
List<SchedulerTaskWrapper>();
             private readonly IServiceScopeFactory _serviceScopeFactory;

             public SchedulerHostedService(
            IEnumerable<IScheduledTask> scheduledTasks,
            IServiceScopeFactory serviceScopeFactory,
            ILogger<SchedulerHostedService> logger)
             {
                 _logger = logger;
                 var referenceTime = DateTime.UtcNow;

                    foreach (var scheduledTask in scheduledTasks)
                           _scheduledTasks.Add(new SchedulerTaskWrapper
                           {
                                  Schedule = Double.Parse(scheduledTask.Schedule),
                                  Task = scheduledTask,
                                  NextRunTime = referenceTime
                           });

                    _serviceScopeFactory = serviceScopeFactory ?? throw new
ArgumentNullException(nameof(serviceScopeFactory));
        }
```

```
        public event EventHandler<UnobservedTaskExceptionEventArgs>
UnobservedTaskException;

              protected override async Task ExecuteAsync(CancellationToken
cancellationToken)
              {
                     while (!cancellationToken.IsCancellationRequested)
                     {
                            await ExecuteOnceAsync(cancellationToken);

                            await Task.Delay(TimeSpan.FromSeconds(5), cancellationToken);
                     }
              }

              private async Task ExecuteOnceAsync(CancellationToken cancellationToken)
              {
                     var taskFactory = new TaskFactory(TaskScheduler.Current);
                     var referenceTime = DateTime.UtcNow;

                     var tasksThatShouldRun = _scheduledTasks.Where(t =>
t.ShouldRun(referenceTime)).ToList();

                     foreach (var taskThatShouldRun in tasksThatShouldRun)
                     {
                            taskThatShouldRun.Increment();

                            await taskFactory.StartNew(
                                   async () =>
                                   {
                                          try
                                          {
                                                 using (var scope =
_serviceScopeFactory.CreateScope())
                                                 {
                                                        var t =
taskThatShouldRun.Task.GetType();
                                                        var method = t.GetMethod("Invoke",
BindingFlags.Instance | BindingFlags.Public);
                                                        var arguments =
method.GetParameters()
                                                               .Select(a =>
                                                                      a.ParameterType ==
typeof(CancellationToken)
                                                                             ?
cancellationToken
                                                                             :
scope.ServiceProvider.GetService(a.ParameterType))
                                                               .ToArray();


                                                        //invoke.
                                                        if
(typeof(Task).Equals(method.ReturnType))
                                                               await (Task)
method.Invoke(taskThatShouldRun.Task, arguments);
                                                        else

                     method.Invoke(taskThatShouldRun.Task, arguments);
                                                 }
                                          }
                                          catch (Exception ex)
                                          {
                                   _logger.LogError(ex, "BackgroundTask Error");
```

```
                        var args = new UnobservedTaskExceptionEventArgs(
                                                ex as AggregateException ?? new
AggregateException(ex));

                                                UnobservedTaskException?.Invoke(this,
args);

                                                if (!args.Observed)
                                                        throw;
                                        }
                                },
                                cancellationToken);
                        }
                }

                private class SchedulerTaskWrapper
                {
                        public double Schedule { get; set; }
                        public IScheduledTask Task { get; set; }

                        public DateTime LastRunTime { get; set; }
                        public DateTime NextRunTime { get; set; }

                        public void Increment()
                        {
                                LastRunTime = NextRunTime;
                                NextRunTime = LastRunTime.AddMilliseconds(Schedule) ;
                        }

                        public bool ShouldRun(DateTime currentTime)
                        {
                                return NextRunTime < currentTime && LastRunTime !=
NextRunTime;
                        }
                }
        }
    }
```

Startup.cs

```
using System;
using CryptoBot.BackgroundTasks;
using CryptoBot.Commands;
using CryptoBot.Commands.Alerts;
using CryptoBot.Commands.GetSettings;
using CryptoBot.Commands.HardwareCost;
using CryptoBot.Commands.PercentChange;
using CryptoBot.Commands.TimeToCompare;
using CryptoBot.Commands.UnconfirmedTransactions;
using CryptoBot.Data;
using CryptoBot.Services;
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Telegram.Bot;

namespace CryptoBot
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services
```

```
                .AddMvcCore()
                .AddCors()
                .AddJsonFormatters();

            var token = Environment.GetEnvironmentVariable("TELEGRAM_BOT_TOKEN");
            var domain = Environment.GetEnvironmentVariable("HEROKU_URL");

            var telegramBotClient = new TelegramBotClient(token);

            var isDevelopment = !string.IsNullOrEmpty(domain);

            var webhookUrl = isDevelopment ? $"{domain}/api/webhook" : string.Empty;
            telegramBotClient.SetWebhookAsync(webhookUrl);

            services.AddSingleton<ITelegramBotClient>(telegramBotClient);
            services.AddSingleton<HttpClientWrapper>();
            services.AddSingleton<Notifyer>();
            services.AddSingleton<DataContext>();
            services.AddSingleton<CurrencyService>();
            services.AddSingleton<ProfitMineService>();

            services.AddSingleton(typeof(IRepository<>), typeof(MongoDbRepository<>));

            // Add scheduled tasks & scheduler
            services.AddSingleton<IScheduledTask, PoloniexTask>();
            services.AddSingleton<IScheduledTask, BittriexTask>();
            services.AddSingleton<IScheduledTask, CoinMarketTask>();
            services.AddSingleton<IHostedService, SchedulerHostedService>();

            // Telegram commands
            services.AddScoped<ITelegramCommand, HelpCommand>();//help
            services.AddScoped<ITelegramCommand, LoginCommand>();//login
            services.AddScoped<ITelegramCommand,
NotifyPoloniexCommand>();//notifypoloniex
            services.AddScoped<ITelegramCommand,
NotifyPoloniexTrueCommand>();//notpolTrue
            services.AddScoped<ITelegramCommand,
NotifyPoloniexFalseCommand>();//notpolFalse
            services.AddScoped<ITelegramCommand, NotifyBittrixCommand>();//notifybittrix
            services.AddScoped<ITelegramCommand, NotifyBittrixTrueCommand>();//notbitTrue
            services.AddScoped<ITelegramCommand,
NotifyBittrixFalseCommand>();//notbitFalse
            services.AddScoped<ITelegramCommand,
PoloniexVolumeChangeCommand>();//poloniexvolumechange
            services.AddScoped<ITelegramCommand,
BittrixVolumeChangeCommand>();//bittrixvolumechange
            services.AddScoped<ITelegramCommand,
PoloniexCostChangeCommand>();//poloniexcostchange
            services.AddScoped<ITelegramCommand,
BittrixCostChangeCommand>();//bittrixcostchange
            services.AddScoped<ITelegramCommand,
TimeToCompareBittrixCommand>();//bittrixtimecompare
            services.AddScoped<ITelegramCommand,
TimeToComparePoloniexCommand>();//poloniextimecompare
            services.AddScoped<ITelegramCommand,
GetTimeToCompareCommand>();//gettimesettings
            services.AddScoped<ITelegramCommand, GetAllPercentCommand>();//getpercent

            services.AddScoped<ITelegramCommand, InformationCommand>();//Information
            services.AddScoped<ITelegramCommand,
NotifyCoinMarketCommand>();//notifycoinmarket
            services.AddScoped<ITelegramCommand,
NotifyCoinMarketFalseCommand>();//notcoinmarketfalse
```

```
            services.AddScoped<ITelegramCommand,
NotifyCoinMarketTrueCommand>();//notcoinmarkettrue
            services.AddScoped<ITelegramCommand, SetAlertCommand>();//alert
            services.AddScoped<ITelegramCommand, GetAlertCommand>();//getalert
            services.AddScoped<ITelegramCommand, DeleteAlertCommand>();//deletealert
            services.AddScoped<ITelegramCommand, MainCommand>();//main
            services.AddScoped<ITelegramCommand, UnconfirmedBtcCommand>();//unconfirmed
transactions BTC
            //services.AddScoped<ITelegramCommand, AsicCommand>();

            //services.AddScoped<ITelegramCommand, GetAsicCommand>();

            services.AddScoped<ITelegramCommand, ProfitCommand>();//profit
            services.AddScoped<ITelegramCommand, S9Command>();//s9
            services.AddScoped<ITelegramCommand, D3Command>();//d3
            services.AddScoped<ITelegramCommand, L3PlusCommand>();//l3
            services.AddScoped<ITelegramCommand, B8Command>();//b8
            services.AddScoped<ITelegramCommand, Nvidia1070>();//n1070
            services.AddScoped<ITelegramCommand, Nvidia1080Ti>();//n1080ti
            services.AddScoped<ITelegramCommand, HardwareSettingsCommand>();//hw


            services.AddScoped<TelegramCommandService>();
        }

        public void Configure(IApplicationBuilder app)
        {
              app
                          .UseDefaultFiles()
                          .UseStaticFiles()
                          .UseMvcWithDefaultRoute()
                          .UseDeveloperExceptionPage()
                          .UseDatabaseErrorPage();
        }
    }}
```

ПРИЛОЖЕНИЕ 2

JsonBTC.cs

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EMA.Json
{
    public class BTC
    {
        [JsonProperty("USD")]
        public double USD { get; set; }

        public DateTime dateTime { get; set; }
    }

    public class JsonBTC
    {

        [JsonProperty("BTC")]
        public BTC BTC { get; set; }
    }
}
```

ConvertUnixTime.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EMA
{
    static class ConvertUnixTime
    {
        public static DateTime ConvertFromUnixTimestamp(double timestamp)
        {
            DateTime origin = new DateTime(1970, 1, 1, 0, 0, 0, 0);
            return origin.AddSeconds(timestamp);
        }

        public static double ConvertToUnixTimestamp(DateTime date)
        {
            DateTime origin = new DateTime(1970, 1, 1, 0, 0, 0, 0);
            TimeSpan diff = date - origin;
            return Math.Floor(diff.TotalSeconds);
        }
    }
}
```

EMA.cs

```
using EMA.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EMA
{
    static class EMA
    {
        public static double Alpha { get; private set; }
        public static double SMA { get; private set; }

        public static List<EMAParameters> GetValue(List<JsonBTC> list, int interval)
        {
            Alpha = 2.0/ (interval + 1);
            SMA = list.Take(interval).Sum(z => z.BTC.USD)/interval;
            List<EMAParameters> list1 = new List<EMAParameters>();
            list1.Add(new EMAParameters
            {
                value = SMA,
                dateTime = list[interval - 1].BTC.dateTime
            });
            foreach (var item in list.Skip(interval+1))
            {
                int i = 0;
                list1.Add(new EMAParameters
                {
                    value = Alpha * item.BTC.USD + (1 - Alpha) * list1[i].value,
                    dateTime = item.BTC.dateTime
                });
                i++;
            }
            return list1;
        }
    }
}
```

EMAParameters.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EMA
{
    class EMAParameters
    {
        public double value { get; set; }
        public DateTime dateTime { get; set; }
    }
}
```

MiningData.cs

```
using EMA.Json;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace EMA
{
    class MiningData
    {
        List<JsonBTC> list;

        DateTime start;
        double stepSec;
        string writePath = "data.txt";

        public MiningData()
        {
            list = new List<JsonBTC>();
            start = new DateTime(2017,01,01);
            stepSec = 60*60*24;
        }

        public void StartMining()
        {
            for (int i = 365; i > 0; i--)
            {
                var time = ConvertUnixTime.ConvertToUnixTimestamp(start);
                var response = new WebClient().DownloadString("https" + $"://min-
api.cryptocompare.com/data/pricehistorical?fsym=BTC&tsyms=BTC,USD,EUR&ts={time}");
                list.Add(JsonConvert.DeserializeObject<JsonBTC>(response));
                list.LastOrDefault().BTC.dateTime = start;
                start = start.AddSeconds(stepSec);
                Thread.Sleep(400);
            }
            using (var sw = new StreamWriter(writePath))
            {
                sw.Write(JsonConvert.SerializeObject(list));
            }
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace EMA
{
    static class Program
    {
```

```
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Form1.cs

```
using EMA.Json;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace EMA
{
    public partial class Form1 : Form
    {
        List<JsonBTC> list;

        public Form1()
        {
            InitializeComponent();
            list = new List<JsonBTC>();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            using (var sr = new StreamReader("data.txt"))
            {
                list = JsonConvert.DeserializeObject<List<JsonBTC>>(sr.ReadToEnd());
            }
            chart1.Series["EMA1"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline;
            chart1.Series["EMA2"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline;
            chart1.Series["BTC"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline;

            var ema1p = 30;
            var ema2p = 20;

            var ema1 = EMA.GetValue(list, ema1p);
            var ema2 = EMA.GetValue(list, ema2p);

            int i = 0;
```

```
            foreach (var item in ema1)
            {
                if(i > ema1p)
                    chart1.Series["EMA1"].Points.AddXY(item.dateTime.ToShortDateString(),
item.value);
                else
                    chart1.Series["EMA1"].Points.AddXY(item.dateTime.ToShortDateString(),
0);

                i++;

            }
            i = 0;
            foreach (var item in ema2)
            {
                if (i > ema2p)
                    chart1.Series["EMA2"].Points.AddXY(item.dateTime.ToShortDateString(),
item.value);
                else
                    chart1.Series["EMA2"].Points.AddXY(item.dateTime.ToShortDateString(),
0);
                i++;
            }
        }
    }
}
```