

Титульник

T3

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ	4
Введение.....	5
1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству	7
1.1 Основные направления развития.....	7
1.2 Обзор существующих аналогов.....	16
1.3 Постановка задачи.....	20
2 Моделирование предметной области.....	22
2.1 Разработка функциональной модели предметной области	22
2.2 Используемые технологии	24
2.3 Спецификация функциональных требований	30
3 Проектирование программного средства	32
3.1 Проектирование архитектуры программного средства	32
4 Тестирование программного средства.....	41
5 Методика использования разработанного программного средства	45
6 Техничко-экономическое обоснование проекта	46
6.1 Описание проекта.....	46
6.2 Расчёт сметы затрат и цены ПО.....	47
6.3 Расчёт заработной платы исполнителей	49
6.4 Расчет рентабельности программного средства	51
6.5 Оценка экономической эффективности применения программного средства у пользователя	53
6.6 Выводы по технико-экономическому обоснованию	57
Заключение	59
Список использованных источников	60

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

Майнер – это

Майнинг пул – это

Криптовалюта – это

Range

Кросс-браузерность

– DRY;

– SOLID;

– YAGNI;

– KISS.

PPS и PPLNS.

ВВЕДЕНИЕ

В настоящее время слова: «криптовалюта», «блокчейн», «майнинг» у всех на слуху и существуют большой спрос на программные продукты данной программной области. Начали появляться школы, курсы, компании, работающие в этой сфере. Технологии, которые используются в криптовалютах нашли применение уже и в других областях. Отсюда можно сделать вывод, что данная тема является актуальна и спрос не будет падать еще несколько десятков лет.

На информационном рынке существует достаточно большой выбор программных продуктов в этой области. Однако довериться таким приложениям очень сложно и не практично. Ты никогда не можешь быть уверенным в безопасном хранении паролей от твоего электронного кошелька, личных данных, поступлении награды за созданный блок именно в твой кошелек, а также использование мощности твоего персонального компьютеры на твои блага.

Таким образом можно выделить ряд существующих проблем:

- небезопасность хранения данных;
- взимание большой комиссии за пользование программным продуктом;
- платные программные продукты;
- сложная сфера области;
- не кроссплатформенное ПО;
- ПО на иностранном языке;
- невыгодный курс валют;
- честное распределение награды.

При детальном их рассмотрении можно сделать вывод о том, что внедрение IT-решений действительно приведет к их устранению, развитию собственного программного средства.

Тем не менее, несмотря на все преимущества, при внедрении подобного рода решений возникают определённого рода трудности, которые в худшем случае могут привести к более серьёзным проблемам. Согласно данным опросов, могут возникнуть следующие сложности:

- отсутствие времени для разработки;
- финансовые затруднения;
- технические проблемы;
- высокая сложность проекта и большая ответственность;
- сложность математических вычислений;
- сложность выбора технологии.

С учётом всего вышесказанного, нужно проводить тщательный анализ и подготовку перед внедрением подобных технологий и уже разработкой программного средства.

В основу проектирования были положены следующие принципы:

- DRY;
- SOLID;

- YAGNI;
- KISS.

Данные принципы были выбраны в силу их положительного влияния на конечную архитектуру.

Данное программное средство позволяет самим добывать криптовалюту, объединяться с другими участниками для распределения вычислительной нагрузки, что приведет к ускоренному процессу добычи криптовалют. Также возможно подключиться к разным майнинг пулам, удалить соединение, прослушивание сети. Программный продукт обеспечивает распределение прибыли в соответствии с проделанной работой.

Цель работы — формирование архитектуры приложения для добычи криптовалют, организация сети между участниками вместе с последующей его реализацией.

1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

1.1 Основные направления развития

Программные средства для добычи криптовалют являются различными по масштабу и реализации. Однако уже в настоящее время можно предположить какие модули уже включены или будут включены в подобного рода ПС. Ниже рассмотрены основные из них.

1.1.1 Майнинг пул

Майнинг пул (mining pool) – участник одноранговой сети, с помощью которого происходит распределение вычислительной нагрузки между участниками, которые подключены к данному пулу. Майнинг пул необходим для ускоренного процесса добычи криптовалюты. Скорость в данном случае необходима, чтобы раньше других найти верный хеш. В награду за то, что участники, путем сложных вычислений, нашли верный хеш, они получают награду, т.е. криптовалюту.

Однако к майнинг пулу может быть подключено несколько участников, следовательно, награда будет делиться между ними в зависимости от выполненной работы, потраченной мощности ПК. Соответственно если майнер добывает криптовалюту в одиночку, то вся прибыль достается ему.

Чтобы найти нужный хеш, нужно методом перебора искать подходящий хеш. В разных криптовалютах может быть разное возможное количество вариантов хеша, например, в криптовалюте «Bitcoin» это число составляет 2^{32} степени. Это огромное число и вероятность того, что один майнер вычислит это число быстрее всех майнеров и майнинг пулов мала. Как раз для этого и были придуманы майнинг пулы, а именно чтобы ускорить процесс перебора возможных значений, путем распределения вычислительной нагрузки в зависимости от мощности участника, рассмотрим более подробно.

В зависимости от криптовалюты правильный хеш должен начинаться с определенного числа нулей, также с новым блоком транзакций число нулей может меняться. Таким образом для того, чтобы найти хеш, необходимо три значения: хеш предыдущего блока, криптографическая функция от текущего блока транзакций и диапазон чисел. Диапазон чисел выдается майнинг пулом каждому участнику сети. Данное диапазон – это единственная часть криптографической функции, которая у каждого участника различна, что позволяет генерировать каждому участнику разные хеш-значения. Схематично взаимодействие майнинг пула с участниками можно представить на рисунке 1.1

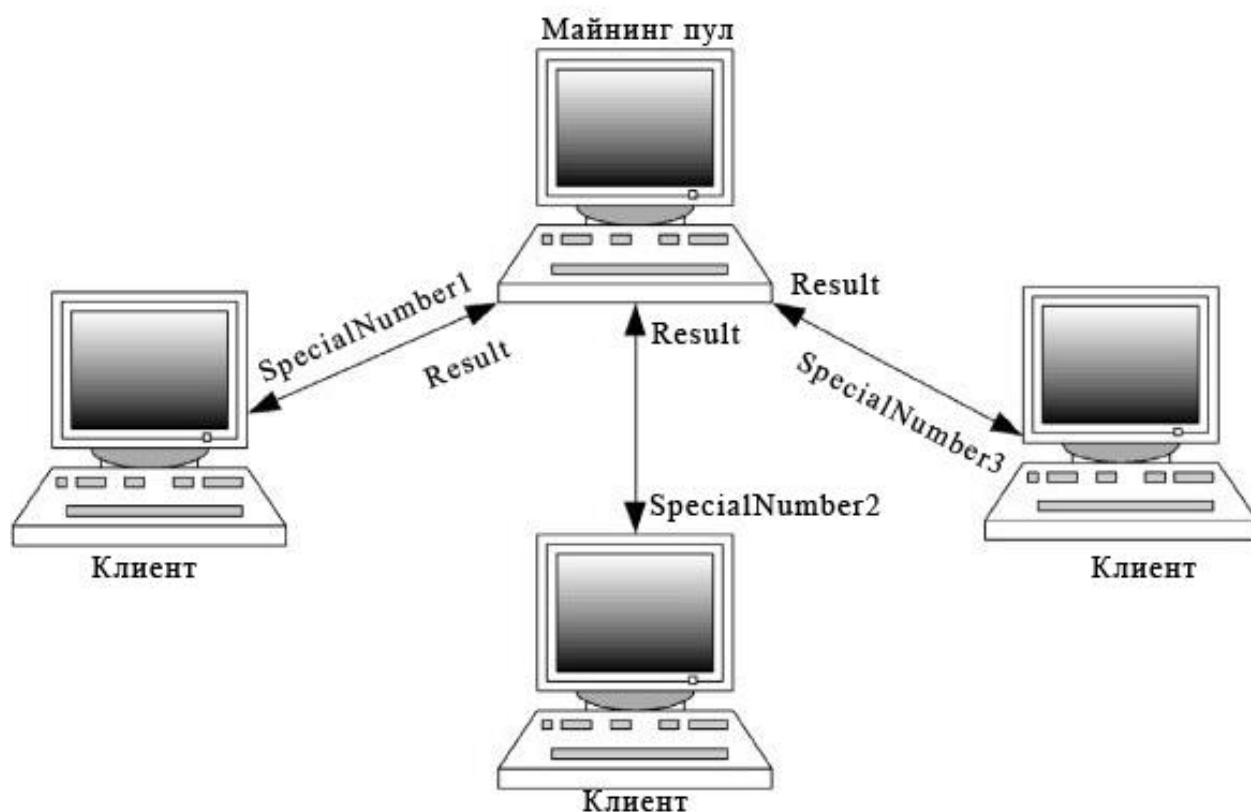


Рисунок 1.1 – Майнинг пул

1.1.2 Процесс добычи криптовалют

Процесс добычи любой криптовалюты называется майнингом. Майнинг нельзя сравнить с обычной печатью денег, потому что в него заложены определенные функции. Сам термин «майнинг» происходит от английского слова Mine (добыча). Если говорить сложными словами, то этот процесс представляет собой деятельность по поддержанию работы сети путем закрытия и создания блоков в «blockchain» с использованием вычислительных мощностей. Участник (майнер) следит за транзакциями и операциями, создают блоки, использует мощности персональных компьютеров для выполнения специальных вычислений по поиску цифровой подписи (хеша), которая закрывает блок. Майнер, который «найдет» верный хеш, оповещает сеть о новом блоке и получает вознаграждение в виде криптовалюты, т.е. в новый блок добавляется транзакция, где майнеру перечисляется прибыль.

Если говорить простыми словами, то майнинг – это процесс добычи криптовалюты путем математических вычислений с использованием мощностей специального оборудования. Майнеры получают вознаграждение, так как их деятельность обеспечивает функционирование и целостность всей системы. В этом и заключается основная задача майнинга.

На начальных этапах добывать криптовалюту мог владелец практически любого компьютера с использованием мощностей процессора. Когда в 2009-м году Сатоши Накамото и компания запускали Bitcoin они изначально

заложили в систему потолок по максимальной эмиссии монет – 21 000 000 BTC. Подобные свойства системы защищают биткоин от инфляции и являются причиной, по которой для добычи новых монет требуются все более мощное оборудование. По разным прогнозам, все монеты BTC будут добыты в середине 21-го века.

Для добычи криптовалюты начали использовать мощные видеокарты. Тогда майнерам удавалось вернуть себе свои вложения за несколько недель. Но минимальные требования для выхода в прибыль продолжали расти. К 2012-ому году добыча криптовалюты даже на самых мощных CPU (процессорах) стала нерентабельной. Наступила эпоха ферм – установок, соединяющих между собой мощные видеокарты, а также асиков (ASIC) – специализированного майнингового оборудования.

Майнинг можно классифицировать в зависимости от формы и используемого оборудования. Основные виды:

1 Майнинг на процессорах компьютеров (CPU) – неэффективный способ добычи криптовалюты. Актуально у кого есть доступ к большому количеству компьютеров и бесплатному электричеству.

2 Майнинг на видеокартах (GPU) актуален для большинства криптовалют, включая эфир, dash и другие. Эффективен при использовании мощных видеокарт.

3 Майнинг на асиках (ASIC) – эффективный способ добычи криптовалют. ASIC – процессоры изготавливают со специальной архитектурой, заточенной под майнинг. Такие устройства имеют высокий уровень окупаемости и их легко обслуживать. Минусы – низкая ликвидность на вторичном рынке и быстрое устаревание асика в связи с растущей сложностью сети.

4 Фермы – установка, объединяющая в себе мощные видеокарты (GPU). Подключается к одному или нескольким компьютерам. Показывает высокую эффективность, при этом оборудование реально продать на вторичном рынке. Увеличение количества майнеров повысило спрос и, как следствие, цены на карты.

5 Браузерный майнинг – процесс добычи криптовалюты через выполнение специального JavaScript-сценария. Эффективность минимальная. Многие сервисы браузерного майнинга созданы в мошеннических целях и внедряют в файловую систему пользователей скрытый майнер без их ведома.

6 Скрытый майнинг – добыча криптовалюты с использованием мощностей чужого оборудования через распространение специальной программы (вируса). Лучшие сборки подобных вирусов практически невозможно удалить с памяти компьютера или обнаружить антивирусным ПО.

7 Майнинг на телефонах и ноутбуках – даже самые мощные модели показывают минимальную эффективность. Зарабатывать на таком способе майнинга не эффективно, также, как и с ноутбуками.

8 Майнинг на сервере – это по сути то же самое, что добыча на CPU, только с высокой производительностью. Может иметь потенциал в будущем при добавлении новых криптовалют.

9 Облачный майнинг – добыча криптовалюты на арендованных серверах в веб-формате. Майнеры платят компаниям деньги за аренду оборудования и в удаленном режиме майнят криптовалюту. Эффективность такого метода зависит от тарифов сервисов по облачному майнингу, текущего курса, а также сложности сети.

1.1.3 Организация сети между участниками

Одно из самых полезных средств для коммуникаций, появившихся в последние несколько лет – организация одноранговых сетей (peer-to-peer networking), часто называемая технологией P2P.

Технология P2P наиболее часто применяется в приложениях для обмена файлами, например, BitTorrent в своих приложениях использует для организации коммуникаций именно технологию одноранговых сетей. Однако она может использоваться и в ряде других приложениях, и она становится все более и более важной в современном мире для реализации коммуникаций между участниками.

В Microsoft тоже не обошли стороной появление технологии P2P. Так появилась платформа Microsoft Windows Peer-to-Peer Networking. В состав этой платформы входят такие важные компоненты, как PNRP (Peer Name Resolution Protocol — протокол преобразования имен членов) и PNM (People Near Me — соседние пользователи). Кроме того, в версию .NET Framework 3.5 было включено новое пространство имен System.Net.PeerToPeer и несколько новых типов и средств, позволяющих создавать приложения P2P с минимальными усилиями.

Рассмотрим два типа архитектур организации сети: технология P2P и "клиент-сервер". Взаимодействие типа «клиент-сервер», представлена на рисунке 1.3, используется в приложениях для коммуникаций по сети (в том числе Интернет). Прекрасным примером могут служить веб-сайты. При просмотре веб-сайта происходит отправка запроса веб-серверу, который затем возвращает требуемую информацию. Если необходимо загрузить какой-то файл, это делается напрямую с веб-сервера. Аналогично, настольные приложения, имеющие возможность подключения к локальной или глобальной сети, обычно устанавливают соединение с каким-то одним сервером, например, сервером баз данных. Однако такой архитектуре присуща проблема с масштабируемостью.

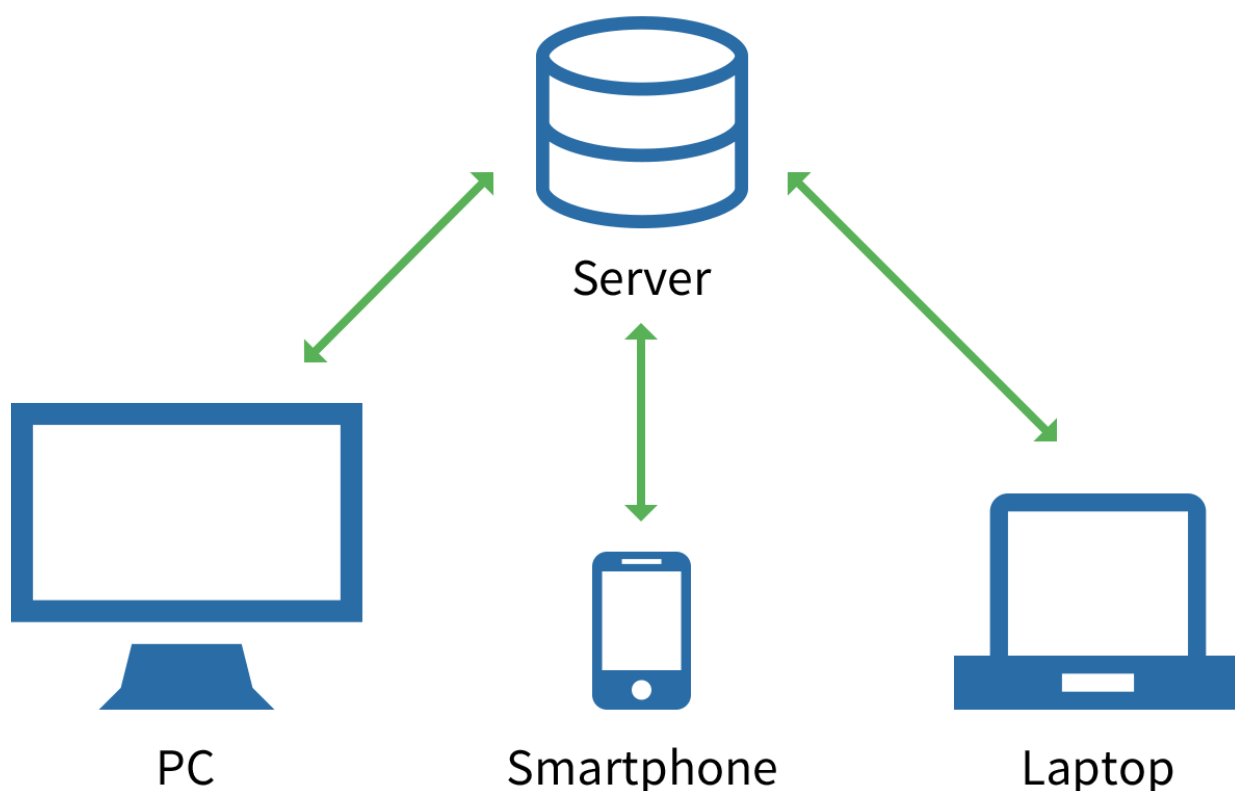


Рисунок 1.3 – Взаимодействие типа «клиент-сервер»

С добавлением каждого клиента нагрузка на сервер, который должен взаимодействовать с каждым клиентом, будет увеличиваться. Если снова взять пример с веб-сайтом, то такое увеличение нагрузки может стать причиной выхода веб-сайта из строя. При слишком большом трафике сервер просто перестанет реагировать на запросы.

Эту проблему можно решить за счет увеличения мощности и ресурсов сервера, а также добавлением еще одного сервера. Первый способ, естественно, ограничивается доступными технологиями и стоимостью более мощного оборудования. Второй способ потенциально более гибкий, но требует добавления дополнительного уровня в инфраструктуру для обеспечения клиентов возможностью либо взаимодействовать с отдельными серверами, либо поддерживать состояние сеанса независимо от сервера, с которым осуществляется взаимодействие.

Одноранговый (peer-to-peer) подход полностью отличается от стандартного подхода, архитектура связи которого продемонстрирована на рисунке 1.4. В случае применения P2P все внимание уделяется поиску способов, которыми клиенты могут взаимодействовать между собой.

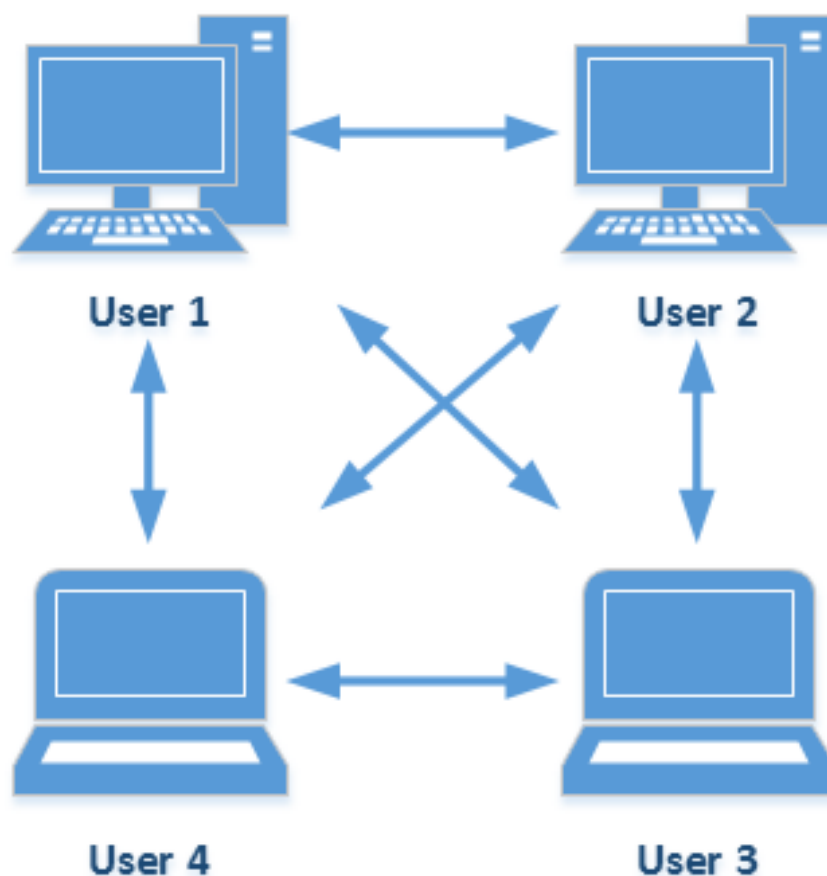


Рисунок 1.4 – Архитектура сети типа «Peer-to-peer»

С применением технологии P2P необязательно для отправки файла передавать его всем клиентам прямо с сервера. Он может быть отправлен только определенному числу клиентов. Несколько остальных клиентов могут далее загрузить его у тех клиентов, у которых он уже есть. После этого еще несколько клиентов могут загрузить его у клиентов, получивших его вторыми, и т.д. По сути, этот процесс может происходить даже быстрее благодаря разбиению файла на куски и распределению этих кусков среди клиентов, одни из которых будут загружать их прямо с сервера, а другие — из других клиентов. Именно так и работают технологии файлообменных систем вроде BitTorrent.

Тем не менее в этой технологии есть некоторые требования. Каждый клиент, участвующий в работе сетевого приложения P2P, для преодоления этих проблем должен быть способен выполнять следующие операции:

- обнаруживать других клиентов;
- подключаться к другим клиентам;
- взаимодействовать с другими клиентами.

В том, что касается способности обнаруживать других клиентов, возможны два очевидных решения: поддержка списка клиентов на сервере, чтобы клиенты могли получать его и связываться с другими клиентами (называемыми peers— равноправными участниками), либо использование инфраструктуры (например, Peer Name Resolution Protocol — протокол преобразования

имен членов), которая позволяет клиентам обнаруживать друг друга напрямую. В большинстве файлообменных систем применяется решение с поддержкой списка на сервере и используются серверы, называемые "трекерами" (trackers).

В роли сервера может также выступать и любой клиент, как показано на рисунке выше, объявляя, что у него имеется доступный файл, и регистрируя его на сервере-трекере. В чистой сети P2P вообще не подразумевается наличие сервера, а лишь равноправные участники.

Проблема подключения к другим клиентам является более тонкой и распространяется на всю структуру используемой приложением P2P сети. При наличии одной группы клиентов, в которой все должны иметь возможность взаимодействовать друг с другом, топология соединений между этими клиентами может приобретать чрезвычайно сложный вид. Зачастую производительность удастся улучшить за счет создания нескольких групп клиентов с возможностью установки подключения между клиентами в каждой из них, но не с клиентами в других группах.

В случае создания этих групп по принципу локальности можно добиться дополнительного повышения производительности, поскольку в таком случае клиенты получают возможность взаимодействовать друг с другом по более коротким (с меньшим числом прыжков) сетевым путям между машинами.

Способность взаимодействовать с другими клиентами, пожалуй, не так важна, поскольку существуют хорошо зарекомендовавшие себя протоколы вроде TCP/IP, которые вполне могут применяться и здесь.

Обеспечение клиентов возможностью обнаруживать, подключаться и взаимодействовать друг с другом играет центральную роль в любой реализации P2P.

В мире «криптовалют» и «майнинга» каждый участник является клиентом и сервером одновременно в сети типа P2P. Под участником можно подразумевать майнинг пул или майнера, который добывает криптовалюту самостоятельно. В пуле архитектура сети типа «клиент-сервер», а сам пул является участником одноранговой сети.

1.1.4 Паттерн проектирования для оповещения сети

Оповещение сети при наступлении какого-либо события реализует широко известный паттерн проектирования «Наблюдатель». Данный паттерн представляет поведенческий шаблон проектирования, который использует отношение "один ко многим". В этом отношении есть один наблюдаемый объект и множество наблюдателей. И при изменении наблюдаемого объекта автоматически происходит оповещение всех наблюдателей.

Данный паттерн еще называют издатель-подписчик, поскольку отношения издателя и подписчиков характеризуют действие данного паттерна: подписчики подписываются email-рассылку определенного сайта. Сайт-издатель с помощью email-рассылки уведомляет всех подписчиков о изменениях. А

подписчики получают изменения и производят определенные действия: могут зайти на сайт, могут проигнорировать уведомления и т.д.

Данный паттерн проектирования стоит применять:

1 Когда система состоит из множества классов, объекты которых должны находиться в согласованных состояниях;

2 Когда общая схема взаимодействия объектов предполагает две стороны: одна рассылает сообщения и является главным, другая получает сообщения и реагирует на них. Отделение логики обеих сторон позволяет их рассматривать независимо и использовать отдельно друга от друга.

3 Когда существует один объект, рассылающий сообщения, и множество подписчиков, которые получают сообщения. При этом точное число подписчиков заранее неизвестно и процессе работы программы может изменяться.

С помощью диаграмм UML данный шаблон можно выразить следующим образом на рисунке 1.2:

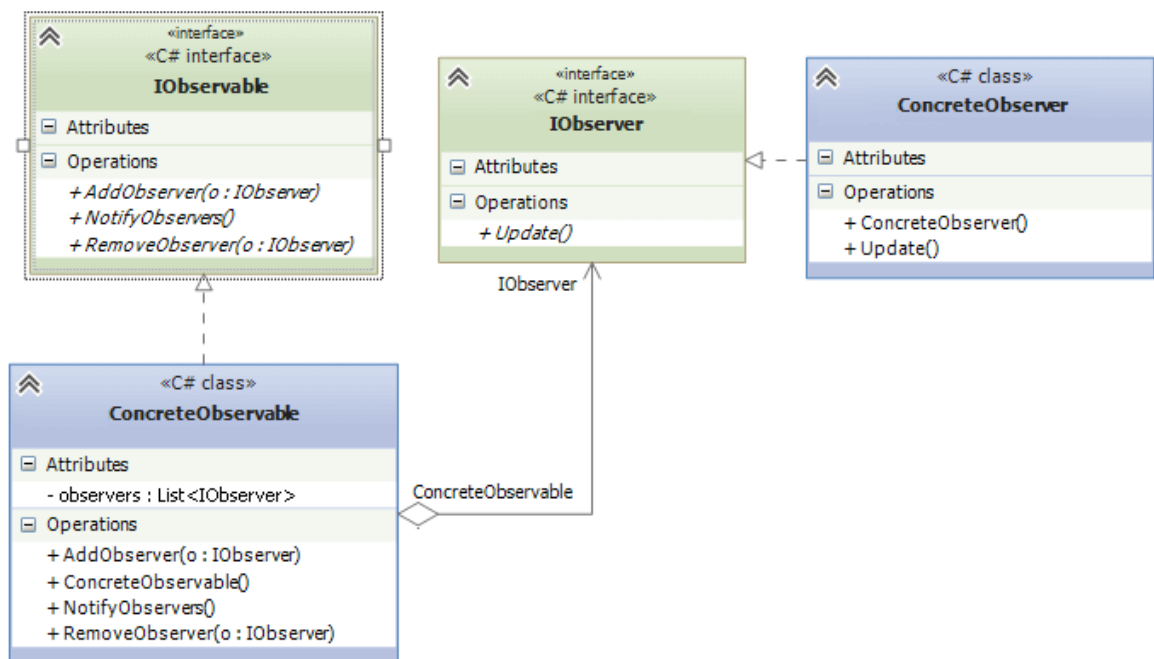


Рисунок 1.2 - UML-диаграмма шаблон проектирования «Наблюдатель»

Участники данного паттерна:

1 **IObservable**: представляет наблюдаемый объект. Определяет три метода: `AddObserver()` (для добавления наблюдателя), `RemoveObserver()` (удаление наблюдателя) и `NotifyObservers()` (уведомление наблюдателей)

2 **ConcreteObservable**: конкретная реализация интерфейса **IObservable**. Определяет коллекцию объектов наблюдателей.

3 IObserver: представляет наблюдателя, который подписывается на все уведомления наблюдаемого объекта. Определяет метод Update(), который вызывается наблюдаемым объектом для уведомления наблюдателя.

4 ConcreteObserver: конкретная реализация интерфейса IObserver.

1.1.5 Распределение прибыли

Майнинг пулы сегодня являются настоящим спасательным кругом для тех, кто хочет добывать криптовалюту, но при этом не имеет возможности модернизировать свое оборудование в соответствии со сложностью сети. На сегодняшний день практически все «добытчики», кроме крупных майнинговых компаний, являются участниками различных пулов. При этом такие сервисы могут включать в себя различные способы начисления наград, самыми популярными из которых являются PROP, PPS и PPLNS. Сейчас проведем анализ данных алгоритмов.

Система выплат PROP является сокращением от слова Proportional, что в переводе на русский означает «пропорциональный». Собственно, в самом названии кроется принцип распределения наград в данной системе. Если кратко, то в пулах, где функционирует данная система вознаграждений, все выплаты распределяются пропорционально вкладу каждого участника.

PPLNS(Pay Per Last N Shares - "Платим за последние N шар") относится к пропорциональным системам начисления наград. Технически данный алгоритм является наиболее сложным, но при этом выгодна как для пулов, так и майнеров, которые «трудятся» в рамках одного пула длительное время. Такая система предполагает, что награда выплачивается не за то количество хеш-кодов, которые были отправлены майнером в период между нахождением блоков, как, например, в системе PROP, а за количество хеш-значений, отправленных за определенный временной участок, которые имеют свое название в данной предметной области - «шифты». Этот участок времени определяется каждым пулом по-своему.

PPS – это система пуловых наград, которая имеет статический характер. То есть, каждый участник пула получает фиксированную плату за каждый хеш-код, принятую пулом. Если немного детализировать сам расчет, то в подсчетах конечной награды учитывается также текущая награда за блок и сложность сети.

Такой подход в начислении наград крайне выгоден пользователям, ведь они получают выплаты за всю проделанную работу, независимо от того, был ли добавлен новый блок в цепь. В свою очередь пулы иногда остаются в проигрыше, выплачивая награды из собственных резервов. Также стоит помнить, что на всех пулах, работающих по системе PPS, имеются достаточно высокие комиссии, которые частично и покрывают траты.

Сказать однозначно, что выгоднее PPS или PPLNS невозможно. Все зависит от мощности вашего оборудования, а также характера майнинга, которого майнер придерживается. Например, для тех пользователей, которые имеют достаточно мощное оборудование и стараются придерживаться всегда

одного и того же пула, более выгодной будет система PPLNS. Проработав в одной системе определенное количество времени, им удастся выйти на чистую прибыль для себя.

1.2 Обзор существующих аналогов

В настоящее время существует значительное количество программных средств для добычи криптовалюты различного уровня реализации. Далее рассмотрены некоторые из тех, которые заслуживают особого внимания.

1.2.1 CG MINER

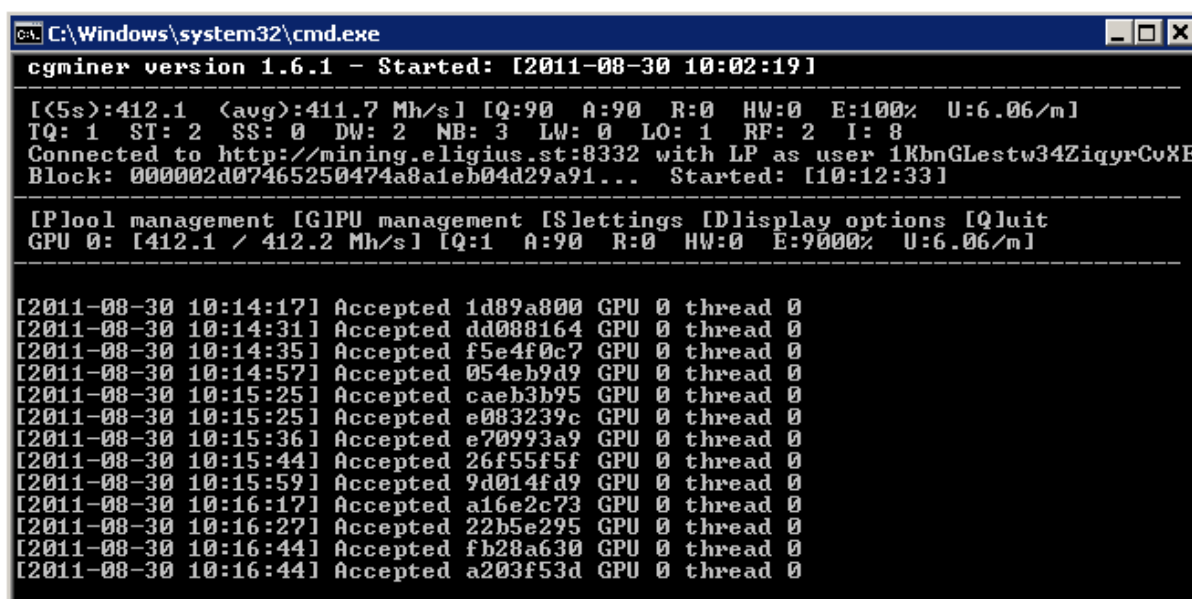
Консольный клиент для майнинга CG MINER (рис 1.5). Главное ее преимущество – высокая стабильность и эффективная работа в фоновом режиме. Иными словами, вам не нужно постоянно следить за работой программы – запустили вычисления, которые не требуют от вас дополнительного внимания. Процессы вычисления выполняются в фоновом режиме.

Достоинства:

- поддерживает многие видеокарты;
- существует большое количество настроек;
- кроссплатформенное ПС;
- работа в фоновом режиме.

Недостатки:

- только для опытного пользователя;
- отсутствие удобного интерфейса;
- невозможно отображать графические элементы, например, графики;
- добывает только одну криптовалюту.



```
C:\Windows\system32\cmd.exe
cgminer version 1.6.1 - Started: [2011-08-30 10:02:19]

[<5s>:412.1 <avg>:411.7 Mh/s] [Q:90 A:90 R:0 HW:0 E:100% U:6.06/m]
TQ: 1 ST: 2 SS: 0 DW: 2 NB: 3 LW: 0 LO: 1 RF: 2 I: 8
Connected to http://mining.eligius.st:8332 with LP as user 1KbnGLestw34ZiqyrCvXE
Block: 000002d07465250474a8a1eb04d29a91... Started: [10:12:33]

[P]ool management [G]PU management [S]ettings [D]isplay options [Q]uit
GPU 0: [412.1 / 412.2 Mh/s] [Q:1 A:90 R:0 HW:0 E:9000% U:6.06/m]

[2011-08-30 10:14:17] Accepted 1d89a800 GPU 0 thread 0
[2011-08-30 10:14:31] Accepted dd088164 GPU 0 thread 0
[2011-08-30 10:14:35] Accepted f5e4f0c7 GPU 0 thread 0
[2011-08-30 10:14:57] Accepted 054eb9d9 GPU 0 thread 0
[2011-08-30 10:15:25] Accepted caeb3b95 GPU 0 thread 0
[2011-08-30 10:15:25] Accepted e083239c GPU 0 thread 0
[2011-08-30 10:15:36] Accepted e70993a9 GPU 0 thread 0
[2011-08-30 10:15:44] Accepted 26f55f5f GPU 0 thread 0
[2011-08-30 10:15:59] Accepted 9d014fd9 GPU 0 thread 0
[2011-08-30 10:16:17] Accepted a16e2c73 GPU 0 thread 0
[2011-08-30 10:16:27] Accepted 22b5e295 GPU 0 thread 0
[2011-08-30 10:16:44] Accepted fb28a630 GPU 0 thread 0
[2011-08-30 10:16:44] Accepted a203f53d GPU 0 thread 0
```

Рис. 1.5 – Внешний вид CGMINER

1.2.2 GUI Miner

Простая, но функциональная программное средство для CPU-майнинга. По сути, это практически точная копия CG Miner, но «завернутая» в графическую оболочку и, что очень удобно, переведенная на русский язык. Работать в ней гораздо удобнее, но опытные майнеры предпочитают привычную и более надежную CG Miner. Внешний вид GUI Miner представлен на рисунке 1.6.

Достоинства:

- графический интерфейс;
- ПС на русском языке;
- большое количество настроек.

Недостатки:

- оболочка для CG MINER, т.е. зависит от другого ПС;
- инструмент для GPU/CPU-майнинга;
- только для Windows.

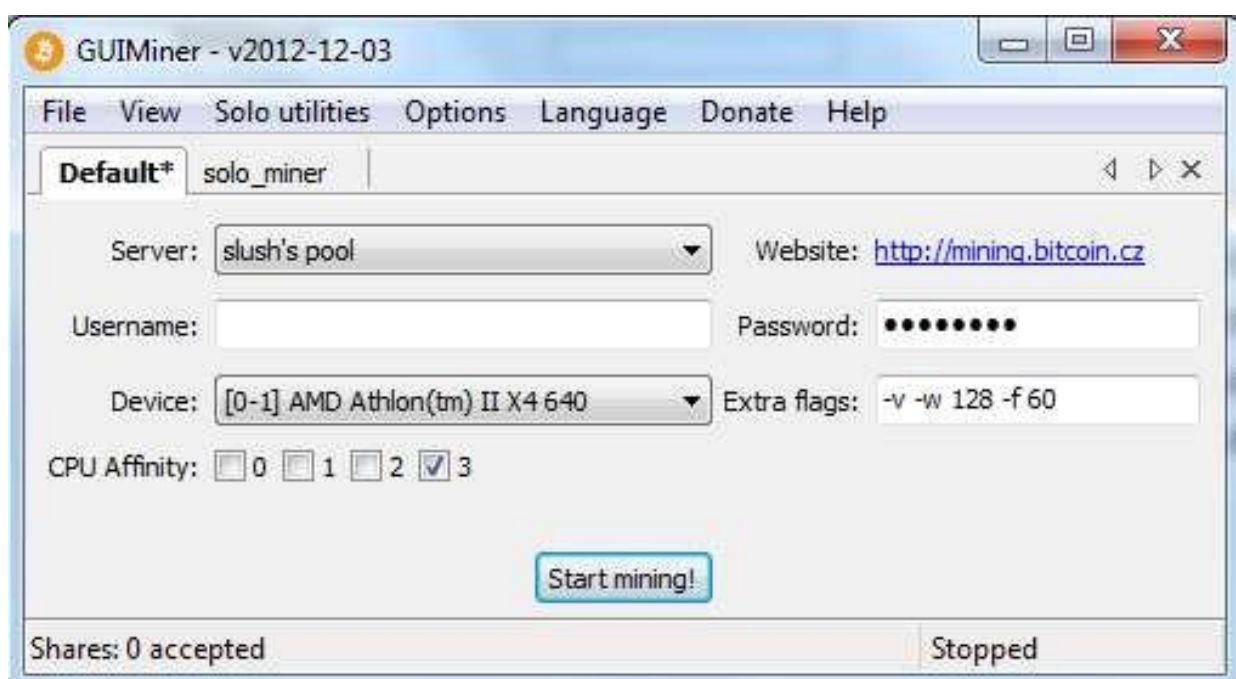


Рис. 1.6 – Внешний вид GUI Miner

1.2.3 Miner Gate

Универсальная и очень простая в использовании программа для майнинга 14 криптовалют. Отличается удобной графической панелью и встроенным конвертором виртуальных валют. А еще смарт-режимом, в котором система сама выбирает, какую криптовалюту выгоднее добывать именно сейчас. Свой выбор программа делает, исходя из используемых мощностей и текущего курса криптовалют. Внешний вид Miner Gate представлен на рисунке 1.7.

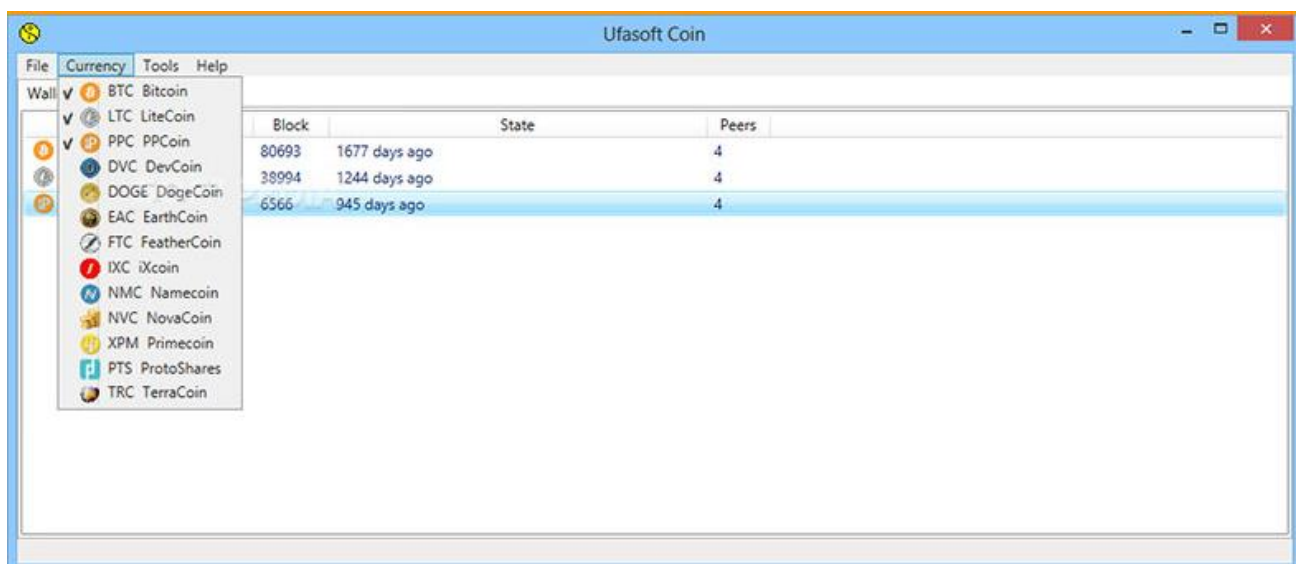


Рис. 1.7 - Внешний вид Ufasoft Miner

Достоинства:

- добывать можно около 14 видов криптовалют;
- графический интерфейс;
- кроссплатформенное ПО;
- умный-режим.

Недостатки:

- ПО является платным;
- высокая цена на ПО;
- высокий курс конвертации валют;
- ПС доступно только на английском языке.

1.2.4 Ufasoft Miner

Очень простая программа для CPU-майнинга. В системе существуют настройки: регулировка температуры процессора и небольшие системные требования. С последним пунктом связан и существенный недостаток программы – она не подойдет для мощного оборудования, а значит, результаты майнинга будут весьма скромными. Тем не менее, это идеальный вариант для новичков, которые хотят разобраться в процессе добычи криптовалют. Внешний вид Ufasoft Miner представлен на рисунке 1.8.

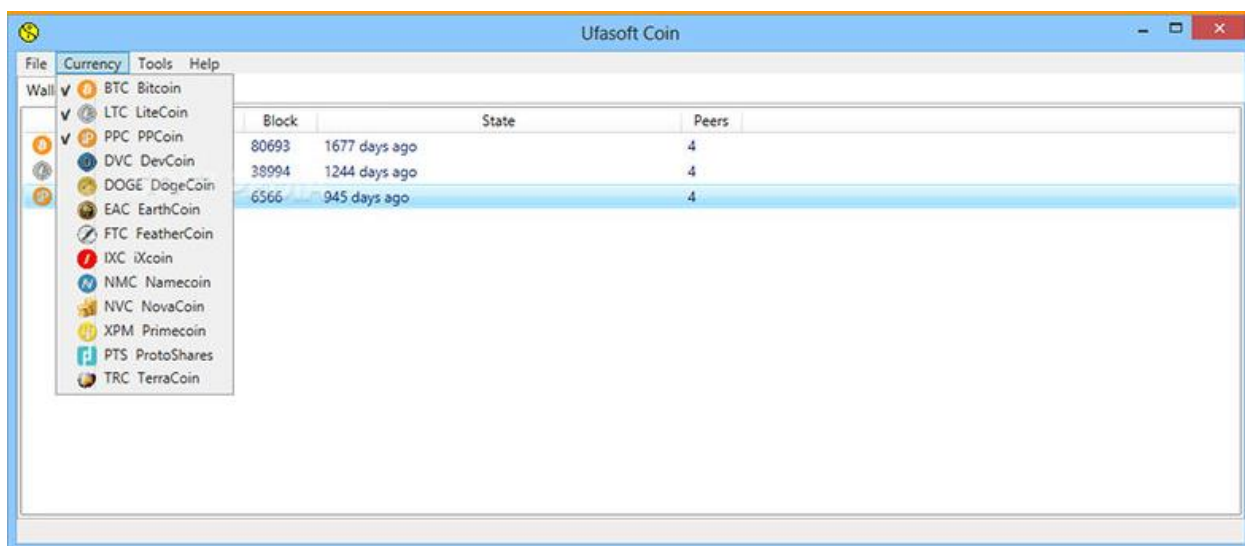


Рис. 1.8 - Внешний вид Ufasoft Miner

Достоинства:

- добывать можно около 4 видов криптовалют;
- удобный графический интерфейс;
- дополнительные настройки.

Недостатки:

- ПО не подходит для мощного ПО;
- Результаты добычи криптовалют маленькие;
- только для Windows;
- отсутствие документации.

1.2.5 Nice Hash Miner

Nice Hash Miner – универсальная программа, которая позволяет майнить монеты как через процессор, так и через видеокарту. Основное преимущество – автоматический подбор оптимального алгоритма для добычи монет на имеющемся оборудовании. Программа все добытые монеты сразу переводит в биткоины. Последнее, к слову, нравится далеко не всем, ведь автоматическая конвертация не дает возможности копить другие криптовалюты и зарабатывать на изменениях их курса. Внешний вид Nice Hash Miner представлен на рисунке 1.9.

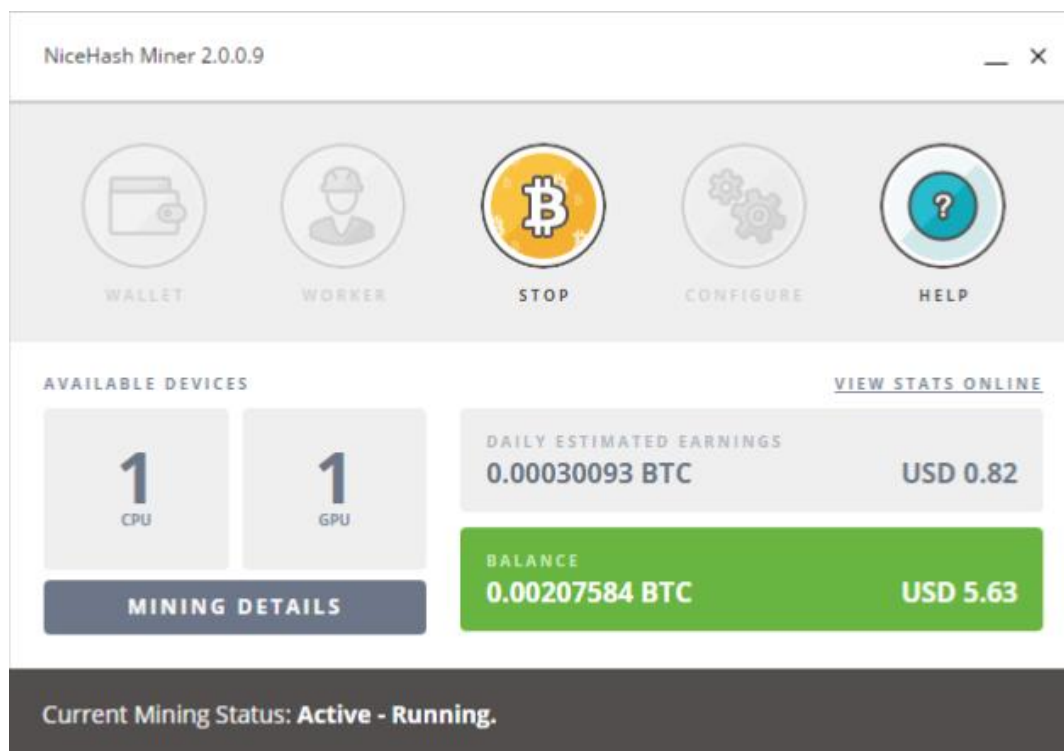


Рисунок 1.9 - Внешний вид Nice Hash Miner

Достоинства:

- добывать можно большое количество криптовалют;
- майнинг через видеокарту или процессор;
- ПО подбирает оптимальный алгоритм для добычи криптовалют.

Недостатки:

- перевод добытые монеты сразу в биткоины;
- ПС только для Linux.

1.3 Постановка задачи

В результате анализа вышеприведённых аналогов можно выделить ряд недостатков, характерных для большинства из них:

- загруженный интерфейс или отсутствие его;
- отсутствие возможности настройки интерфейса;
- отсутствия необходимой документации, как пользоваться программным средством;
- конвертация валют в другие криптовалюты;
- большая комиссия;
- невыгодный вывод денежных средств;
- большое количество ложных программных средств;

Данные позиции играют важную роль в ПС данной предметной области.

Но одной из главной проблемы является то, что у значительного числа аналогов лишь несколько модулей исполнены на достаточном уровне, в то время как остальные находятся в зачаточном состоянии либо отсутствуют вообще.

Поэтому целью проекта является разработка программного средства, которое позволит добывать криптовалюту, оптимально распределять прибыль между участниками сети. Установка взаимодействия между участниками на высоком уровне.

Проектируемое средство должно отвечать следующим функциональным требованиям:

- обладать пользовательским интерфейсом в соответствии с требованиями;
- иметь встроенную документацию;
- поддержка подключения нескольких майнинг пулов;
- поддержка подключения нескольких криптовалют;
- распределения прибыли;
- распределение вычислительной нагрузки;
- организация сети между участниками;
- кросс-браузерность.

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

2.1 Разработка функциональной модели предметной области

Для моделирования предметной области в качестве среды была выбрана Enterprise Architect. Enterprise Architect обеспечивает всестороннюю поддержку всех элементов, связей и диаграмм, соответствующих UML.

На рисунке 2.1 приведена диаграмма вариантов использования для создаваемого программного средства. Согласно данной диаграмме, в разрабатываемом программном средстве предусмотрено 4 вида актеров: администратор, майнер, майнинг пул и гость.

Гость не является пользователем в полном смысле этого слова. В основном ему доступны лишь страницы авторизации и регистрации. Но также гость как потенциальный полноценный пользователь системы может просматривать документацию, тем самым получая информацию о программном средстве, составляя мнение о нём и формируя оценку. На основании этой оценки он впоследствии принимает решение о необходимости данного ПС.

Майнер является пользователем системы, который имеет свой электронных кошелёк. Для майнера открыт доступ ко всем основным функциям, а именно: добыча криптовалюты, подключиться к майнинг пулу, отключиться от майнинг пула, выход из личного кабинета, авторизоваться, отправить запрос на сервер, принимать входящие сообщения, получить награду.

Майнинг пул является сервером для майнера, он также, как и майнер имеет свой электронный счёт. Этот менеджер распределяет вычислительную нагрузку между участниками, распределяет прибыль, устанавливает алгоритм вычислений, ведет статистику выполненной работы для дальнейшего распределения прибыли.

Майнер менеджер служит для связи между майнинг пулом и майнером. С помощью его мы можем к майнинг пулу присоединить список майнеров, удалять и добавлять в него участников.

Менеджер вычислительных операций служит для работы с хеш кодами, а именно: вычислить хеш код, определить данный хеш код является валидным или нет, отправить сгенерированный блок в сеть, достать аккаунт майнинг пула. Также менеджер вычислительных операций обрабатывает события:

- полученный хеш код верный;
- полученный хеш код неверный;
- оповестить всех майнеров, что новый блок найден.

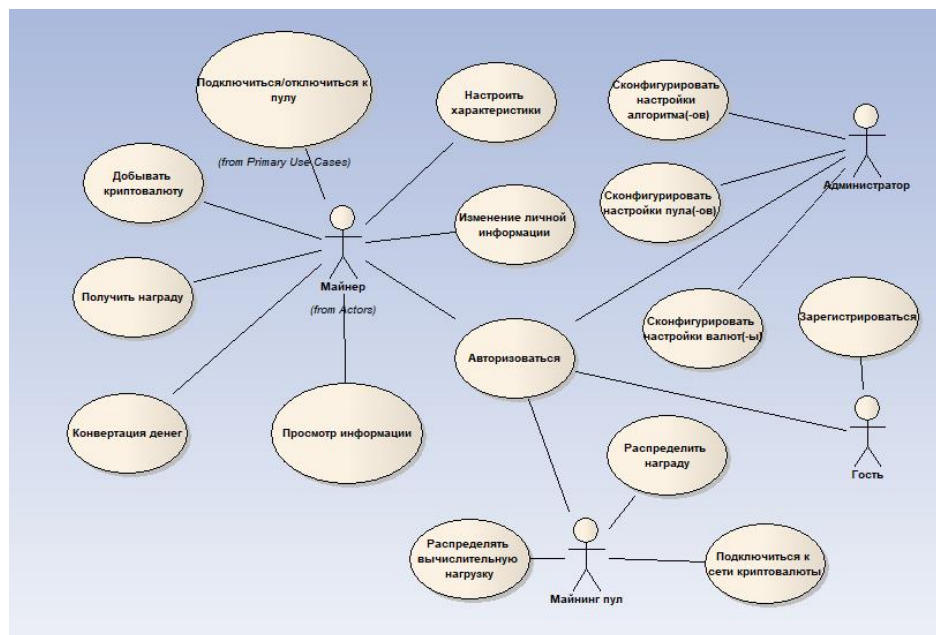


Рисунок 2.1 – Диаграмма вариантов использования

2.2 Используемые технологии

Выбор технологий является важным предварительным этапом разработки сложных информационных систем. Платформа и язык программирования, на котором будет реализована система, заслуживает большого внимания, так как исследования показали, что выбор языка программирования влияет на производительность труда программистов и качество создаваемого ими кода. Ниже перечислены некоторые факторы, повлиявшие на выбор технологий:

- разрабатываемое ПО должно запускаться как веб-сайт;
- среди различных платформ разработки имеющийся программист лучше всего знаком с разработкой на платформе ASP .NET;
- дальнейшей поддержкой проекта, возможно, будут заниматься разработчики, не принимавшие участие в выпуске первой версии;
- имеющийся разработчик имеет опыт работы с объекто-ориентированными и с функциональными языками программирования.

Основываясь на опыте работы имеющихся программистов разрабатывать ПО целесообразно на платформе ASP .NET. Приняв во внимание необходимость обеспечения доступности дальнейшей поддержки ПО, возможно, другой командой программистов, целесообразно не использовать малоизвестные и сложные языки программирования. С учетом этого фактора выбор языков программирования сужается до четырех официально поддерживаемых Microsoft и имеющих изначальную поддержку в Visual Studio 2017: Visual C++/CLI, C#, Visual Basic .NET и JavaScript. Необходимость использования низкоуровневых возможностей Visual C++/CLI в разрабатываемом ПО отсутствует, следовательно, данный язык можно исключить из списка кандидатов. Visual Basic .NET уступает по удобству использования двум другим кандидатам из нашего списка. Оставшиеся два языка программирования C# и JavaScript являются первостепенным, элегантными, современными языками программирования для платформы ASP.NET. Таким образом, с учетом вышеперечисленных факторов, целесообразно остановить выбор на следующих технологиях:

- платформа разработки ASP .NET;
- языки программирования C# и Nancy framework (для клиентской части).

Для реализации поставленной задачи нет необходимости в использовании каких-либо прикладных библиотек для создания веб-приложения, достаточно использовать стандартные библиотеки указанных выше языков. Поддержка платформой .NET различных языков программирования позволяет использовать язык, который наиболее просто и «красиво» позволяет решить возникающую задачу. Разрабатываемое программное обеспечение в некоторой степени использует данное преимущество платформы.

Nancy предназначена для обработки запросов DELETE, GET, HEAD, OPTIONS, POST, PUT и PATCH и предоставляет простой, элегантный, доменный язык (DSL) для возврата ответа всего несколькими нажатиями клавиш,

что дает вам больше времени для фокусировки на важные задачи внутри приложения, язык C# — для реализации логики приложения, функций и методов, иерархия дерева классов, применения паттернов проектирования. В разрабатываемом программном продукте Nancy Framework используется для предоставления внешнего интерфейса в соответствии с требованиями заказчика, C# — для проектирования и реализации вычислительной логики. Далее проводится характеристика используемых технологий и языков программирования более подробно.

2.2.1 Программная платформа ASP.Net

ASP.NET предлагает три структуры для создания веб-приложений: Web Forms, ASP.NET MVC и ASP.NET Web Pages. Все три структуры являются стабильными и существуют относительно долго с помощью которых можно создавать отличные веб-приложения. Независимо от того, какая структура используется в программном средстве, все преимущества и возможности ASP.NET будут доступны.

Каждая структура нацелена на различную архитектуру и стиль разработки. Выбор которой, зависит от комбинации программных знаний, навыков и опыта разработки, типа приложения, которое разрабатывается, и подхода к разработке, с которым будет удобно.

Все три структуры ASP.NET основаны на .NET Framework и имеют общие функциональные возможности .NET и ASP.NET. Например, все три структуры предлагают модель безопасности входа, также имеют одинаковые возможности для управления запросами, обработки сеансов и все другие возможности, которые являются частью основных функций ASP.NET.

Кроме того, выбор одной структуры не исключает возможности использовать в этом приложении другую структуру. Поскольку разные модули могут сосуществовать в одном и том же веб-приложении, нередко можно увидеть отдельные компоненты приложений, написанные с использованием разных фреймворков. Например, клиентские части приложения могут быть разработаны в MVC для оптимизации разметки, тогда как доступ к данным и административные части разрабатываются в Web Forms, чтобы использовать преимущества управления данными и простого доступа к данным.

С помощью ASP.NET Web Forms вы можете создавать динамические веб-сайты с помощью знакомой модели, управляемой событиями. Конструктивная поверхность и сотни элементов управления и компонентов позволяют быстро создавать сложные, мощные сайты с интерфейсом UI с доступом к данным.

ASP.NET MVC дает вам мощный способ создания динамических веб-сайтов, который обеспечивает четкое разделение серверной и клиентской части, дает вам полный контроль над разметкой для приятного и гибкого развития. ASP.NET MVC включает в себя множество функций, которые позволяют

быстро разрабатывать сложных приложений, которые используют новейшие веб-стандарты.

Основная идея паттерна MVC(Model-View-Controller) в том, что и контроллер, и представление зависят от модели, но модель никак не зависит от этих двух компонент. Взаимодействие между компонентами представлено на рисунке 2.2.

Model-View-Controller

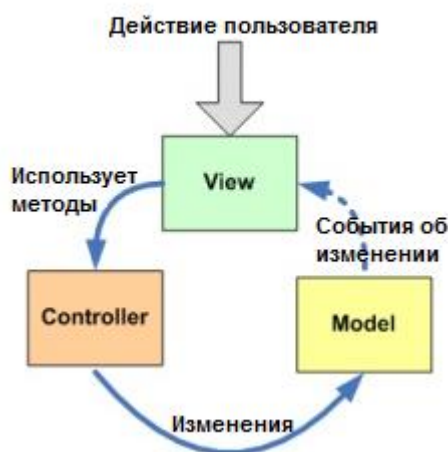


Рисунок 2.2 – Взаимодействие между компонентами

Контроллер перехватывает событие извне и в соответствии с заложенной в него логикой, реагирует на это событие изменяя Модель, посредством вызова соответствующего метода. После изменения Модель использует событие о том что она изменилась, и все подписанные на это события Представления, получив его, обращаются к Модели за обновленными данными, после чего их и отображают.

ASP.NET Web Pages и синтаксис Razor обеспечивают быстрый, доступный и легкий способ объединить код сервера с HTML для создания динамического веб-контента. Соединитесь с базами данных, добавьте видео, ссылку на сайты социальных сетей и включите еще множество функций, которые помогут вам создавать красивые сайты, соответствующие новейшим веб-стандартам.

Также нельзя не упомянуть о ASP.NET Web API - это структура, которая упрощает создание HTTP-сервисов, которые охватывают широкий круг клиентов, включая браузеры и мобильные устройства. С помощью Web API фреймворка можно очень быстро создавать веб-приложения, которые будут возвращать данные через HTTP-вызовы. Данная структура применяется, когда необходимо быстро реализовать приложение, и основные силы вложить в серверную часть.

С помощью данного анализа структур я сделала выбор для своего приложения: ASP.NET Web API. Для этой структуры существует библиотека Nancy, которая упрощает работу.

2.2.2 Язык программирования С# и платформа .Net Framework

Язык программирования С# и среду программирования .NET Framework можно назвать одними из самых значительных и современных технологий для разработчиков. .NET является такой средой, которая была создана для того, чтобы в ней можно было разрабатывать практически любое приложение для запуска в Windows, а С# является языком программирования, который был специально создан для использования в .NET Framework. Хочется отметить, что платформа и язык не стоят на месте и уже в современных версиях .Net Core приложения являются кроссплатформенными. Например, с применением С# и .NET Framework можно создавать динамические веб-страницы, приложения Windows Presentation Foundation, веб-службы XML, компоненты для распределенных приложений, компоненты для доступа к базам данных, классические настольные приложения Windows. А с появлением .Net Core стал применяться такой способ программирования, как микросервисы.

Слово "NET" в названии не означает, что данная среда предназначена только для создания веб-приложений, это лишь показатель того, что, по мнению Microsoft, распределенные приложения, в которых обработка распределяется между клиентом и сервером, являются шагом вперед. Однако важно понимать, что С# представляет собой язык, предназначенный не только для написания приложений, способных работать в Интернете и в сети. Он предоставляет средства для кодирования практически любого типа программного обеспечения или компонентов для платформы Windows. Язык С# и среда .NET привели к революционным изменениям в способе написания разработчиками программ и сделали программирование приложений для Windows гораздо более простым, чем когда-либо.

Язык программирования С# — это современный язык программирования, который характеризуется двумя следующими преимуществами:

- С# спроектирован и разработан специально для применения с Microsoft .NET Framework (развитой платформой разработки, развертывания и выполнения распределенных приложений).

- С# — язык, основанный на современной объектно-ориентированной методологии проектирования, при разработке которого специалисты из Microsoft опирались на опыт создания подобных языков, построенных в соответствии с предложенными около 20 лет назад объектно-ориентированными принципами.

Нужно подчеркнуть то важное обстоятельство, что С# — это полноценный язык программирования. Хотя он и предназначен для генерации кода, выполняемого в среде .NET, сам по себе он не является частью .NET. Существует

ряд средств, которые поддерживаются .NET, но не поддерживаются C#, и, возможно, вас удивит, что есть также средства, поддерживаемые C# и не поддерживаемые .NET (например, некоторые случаи перегрузки операций). Однако поскольку язык C# предназначен для применения на платформе .NET, важно иметь представление о .NET Framework, чтобы эффективно разрабатывать приложения на C#.

C# спроектирован и разработан специально для применения с .NET Framework.

Назначение .NET Framework — служить средой для поддержки разработки и выполнения сильно распределенных компонентных приложений. Она обеспечивает совместное использование разных языков программирования, а также безопасность, переносимость программ и общую модель программирования для платформы Windows.

Базовые функциональные возможности платформы .NET включают в себя:

1 Возможность обеспечения взаимодействия с существующим программным кодом. Эта возможность, несомненно, является очень хорошей вещью, поскольку позволяет комбинировать существующие двоичные единицы СОМ (т.е. обеспечивать их взаимодействие) с более новыми двоичными единицами .NET и наоборот. С выходом версии .NET 4.0 эта возможность стала выглядеть даже еще проще, благодаря добавлению ключевого слова *dynamic*.

2 Поддержка для многочисленных языков программирования. Приложения .NET можно создавать с помощью любого множества языков программирования (C#, Visual Basic, F#, S# и т.д.). При этом в .NET код, написанный на любом языке компилируется в код на промежуточном языке (Intermediate Language - IL).

3 Полная интеграция языков. В .NET поддерживается межъязыковое наследование, межъязыковая обработка исключений и межъязыковая отладка кода. При этом .NET использует общий исполняющий механизм, основным аспектом которого является хорошо определенный набор типов, который способен понимать каждый, поддерживающий .NET язык. Так же в .NET был полностью переделан способ разделения кода между приложениями за счет введения понятия сборки (*assembly*) вместо традиционных библиотек DLL. Сборки обладают формальными средствами для управления версиями и допускают одновременное существование рядом нескольких различных версий сборок.

4 Усовершенствованная поддержка для создания динамических веб-страниц. Хотя в классической технологии ASP предлагалась довольно высокая степень гибкости, ее все равно не хватало из-за необходимости использования интерпретируемых сценарных языков, а отсутствие объектно-ориентированного дизайна часто приводило к получению довольно запутанного кода ASP. В .NET предлагается интегрированная поддержка для создания веб-страниц с

помощью ASP.NET. В случае применения ASP.NET код создаваемых страниц поддается компиляции и может быть написан на любом поддерживающем .NET языке высокого уровня, например, C# или Visual Basic 2010. В новой версии .NET эта поддержка улучшилась еще больше, сделав возможным применение новейших технологий вроде Ajax и jQuery.

5 Эффективный доступ к данным. Набор компонентов .NET, известный под общим названием ADO.NET, позволяет получать эффективный доступ к реляционным базам данных и многим другим источникам данных. Также предлагаются компоненты, позволяющие получать доступ к файловой системе и каталогам. В частности, в .NET встроена поддержка XML, позволяющая манипулировать данными, импортируемыми и экспортируемыми на платформы, отличные от Windows.

6 Установка с нулевым воздействием. Сборки бывают двух типов: разделяемые и приватные. Разделяемые сборки представляют собой обычные библиотеки, доступные всему программному обеспечению, а приватные сборки предназначены для использования только с определенными программами. Приватные сборки являются полностью самодостаточными, поэтому процесс их установки выглядит просто. Никакие записи в системный реестр не добавляются; все нужные файлы просто размещаются в соответствующей папке файловой системы.

7 В мире программирования наблюдается значительный рост применения динамических языков, таких как JavaScript, Python и Ruby. По этой причине в C# была добавлена возможность динамической типизации (dynamic typing). Знать статическим образом, какими объектами могут получаться в конце, не всегда возможно. Теперь вместо использования ключевого слова `object` и назначения этого типа всем сущностям можно предоставить возможность решать этот вопрос среде DLR (Dynamic Language Runtime — исполняющая среда динамического языка) непосредственно во время выполнения.

Динамические возможности C# обеспечивают лучшее взаимодействие. Появляется возможность взаимодействовать с различными динамическими языками и работать с DOM гораздо более простым образом. Кроме того, облегчается работа с API-интерфейсами COM для Microsoft Office.

2.2.3 Библиотека Nancy

Nancy - это легкая библиотека с низкой сложностью для создания HTTP-сервисов на .NET. Целью этой библиотеки является максимально возможное предотвращение и обеспечение пути для всех взаимодействий, функций. Главное преимущество данной библиотеки — это скорость и простота разработки.

Nancy предназначена для обработки запросов DELETE, GET, HEAD, OPTIONS, POST, PUT и PATCH и предоставляет простой, элегантный, доменный язык (DSL) для возврата ответа всего несколькими нажатиями клавиш, что дает вам больше времени для фокусировки на более важные задачи.

Одной из основных концепций в Nancy является хост. Хост выступает в качестве адаптера для среды размещения и Nancy, что позволяет библиотеке работать с существующими технологиями, такими как ASP.NET, WCF и OWIN, или интегрироваться в любое заданное приложение.

В официальной документации Nancy представлено несколько идей на которых базируется данная библиотека:

1 Простая в применении. Даже если добавить новую зависимость или еще один модуль к существующему модулю, по умолчанию Nancy найдёт и добавим его для вас - настройка не требуется;

2 Легко настраиваемая. Разработчики Nancy предусмотрели, чтобы не было должно быть никаких барьеров, которые мешают настройке. В настройках можно конфигурировать выбор контейнера зависимостей, способ выбора маршрута и многое другое.

3 Легко применимая. Количество «кода Nancy», которое нужно в клиентском модуле, минимальное. В клиентских модулях практически не присутствует код Nancy, чтобы не загружать модуль сторонним кодом, что является плюсом. С помощью Nancy API-интерфейсы должны помогают добраться в нужный модуль к нужному методы.

Основной аспект данной библиотеки занимают маршруты, поэтому следует уделить особое внимание выбору и правильному составлению маршрута. Путь определяется в конструкторе модуля. Чтобы определить маршрут в Nancy, нужно указать метод + шаблон + действие + (необязательное условие).

Метод – это HTTP-запрос, который используется для доступа к ресурсу. Nancy поддерживает следующие методы DELETE, GET, HEAD, OPTIONS, POST, PUT и PATCH. По умолчанию запросы HEAD автоматически обрабатываются для всех маршрутов, объявленных для запросов GET.

Маршруту также нужен шаблон, который объявляет URL-адрес приложения, на который отвечает маршрут. Синтаксис шаблона настраивается, но реализация по умолчанию, поставляемая с Nancy, поддерживает различные комбинации, которые описаны в официальной документации.

Следующий параметр в маршруте – действие, что означает поведение, которое вызывается, когда запрос сопоставляется с маршрутом.

Последняя часть является необязательным условием, которое может использоваться, чтобы убедиться, что маршрут согласован только при выполнении определенных условий. Это может быть, например, проверка, чтобы гарантировать, что маршрут вызывается только в том случае, если он использовался мобильным пользовательским агентом. Условие маршрута определяется с помощью лямбда-выражения.

2.3 Спецификация функциональных требований

Была составлена функциональная спецификация требований к программному средству добычи криптовалют.

Основными функциями ПС являются:

- подключение майнера к майнинг пулу;
- отключение майнера от майнинг пула;
- авторизация;
- подтверждение работы;
- запуск сервера;
- выключение сервера;
- добавление криптовалюты;
- добавление майнинг пулов;
- просмотр документации;
- сбор статистики;
- распределение прибыли;
- вычисление хеш кодов;
- распределение вычислительной нагрузки между участниками;
- просмотр информации о майнинг пуле.

Нефункциональные требования для разрабатываемого программного средства:

- работа в современных браузерах (Chrome 60+, Firefox 47+, Opera 50+);
- поддержка локализации;
- реализовать интерфейс в соответствии с требованиями.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Проектирование архитектуры программного средства

Архитектура программного средства состоит из трёх основных компонентов, которые продемонстрированы на рисунке 3.1:

- серверная часть приложения;
- клиентская часть приложения;
- сервер базы данных.

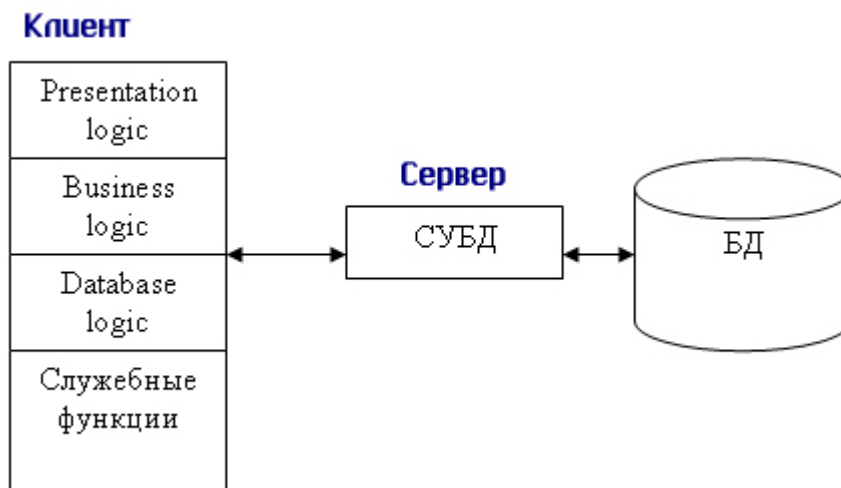


Рисунок 3.1 – Архитектура программного средства

Такая архитектура часто встречается в различных приложениях. Плюсы такого подхода в разработке ПС в том, что каждый компонент имеет свои обязанности и может разрабатываться отдельно от других компонентов.

Клиентская часть системы обменивается данными с серверной, отправляя запросы пользователя в виде HTTP запросов и получая ответы в виде структурированной структуры данных.

Серверная часть включает в себя реализацию двух видов организации сети между участниками: «Клиент-Сервер» и «Сервер-Сервер». Первый тип архитектуры предназначен для обеспечения взаимодействий между майнинг пулом и клиентами, которые к нему подключились, т.е. другими словами майнерами. Сеть, где каждый участник имеет соединение с другим участником «Сервер-Сервер», называется «одноранговая сеть». Следовательно, майнинг пул является участником сети, в которой все имеют равные права. Серверная часть обеспечивает взаимодействие всех участников сети и распределение вычислительной нагрузки.

Для иллюстрации концепции всей организации взаимодействия сети была выбрана диаграмма развёртывания, представленная на рисунке 3.1.

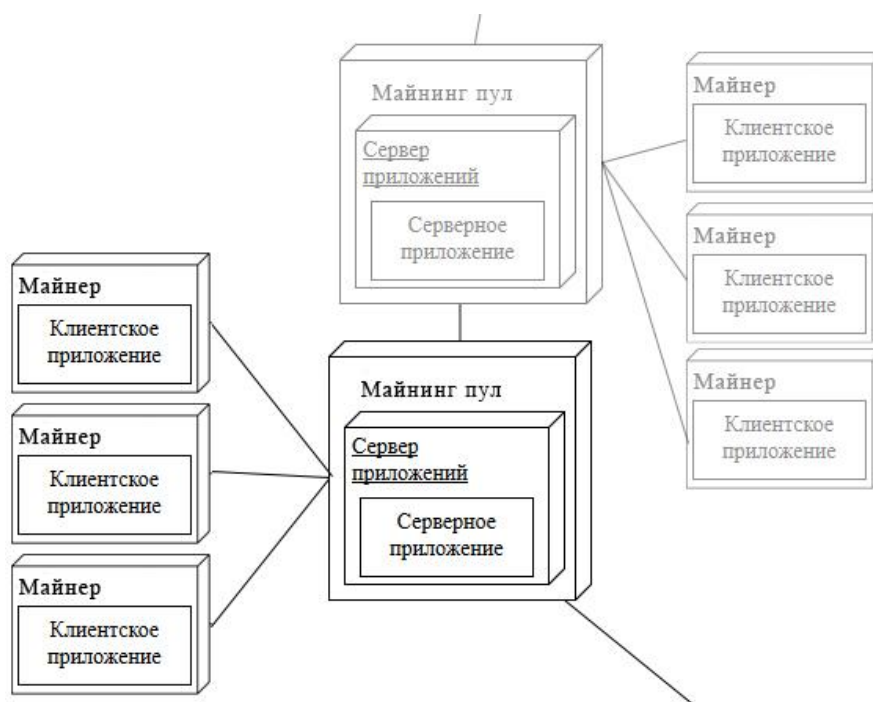


Рисунок 3.1 – Диаграмма развёртывания

3.1.1 Организация сети между участниками. Программный модуль SocketServer.cs

Для обеспечения обмена данными между участниками сети существуют два протокола TCP (Transmission Control Protocol) и UDP (User Datagram Protocol). В данном ПС будем использовать протокол TCP, так как он обеспечивает доставку пакета, отправляет подтверждения о получении, а если произошёл сбой или данные искажены, то запрашивает отправку пакета еще раз, также гарантировано, что пакеты придут в том же порядке, в котором и были отправлены.

Для обмена данными между участниками должно быть создано соединение. Таким образом, в основе межсетевых взаимодействий по протоколам TCP и UDP лежат сокеты. В .NET сокеты представлены классом System.NET.Sockets.Socket, который предоставляет интерфейс для приема и отправки сообщений по сети.

Рассмотрим основные свойства данного класса, которые необходимы для разработки ПС:

- AddressFamily - возвращает все адреса, используемые сокетом;
- Available - возвращает объем данных, которые доступны для чтения;
- Connected - возвращает true, если сокет подключен к удаленному хосту;
- LocalEndPoint - возвращает локальную точку, по которой запущен сокет и по которой он принимает данные;
- ProtocolType - возвращает одно из значений перечисления ProtocolType, представляющее используемый сокетом протокол;

- RemoteEndPoint - возвращает адрес удаленного хоста, к которому подключен сокет;

- SocketType - возвращает тип сокета.

При работе с сокетами я опиралась на методы класса Socket:

- Accept() - создает новый объект Socket для обработки входящего подключения;

- Bind() - связывает объект Socket с локальной конечной точкой;

- Close() - закрывает сокет;

- Connect() - устанавливает соединение с удаленным хостом;

- Listen() - начинает прослушивание входящих запросов;

- Poll() - определяет состояние сокета;

- Receive() - получает данные;

- Send() - отправляет данные;

- Shutdown() - блокирует на сокете прием и отправку данных.

При применении TCP протокола, который требует установление соединения, сервер должен вызвать метод Bind для установки точки для прослушивания входящих подключений и затем запустить прослушивание подключений с помощью метода Listen. Далее с помощью метода Accept можно получить входящие запросы на подключение в виде объекта Socket, который используется для взаимодействия с удаленным узлом. У полученного объекта Socket вызываются методы Send и Receive соответственно для отправки и получения данных. Если необходимо подключиться к серверу, то вызывается метод Connect. Для обмена данными с сервером также применяются методы Send или Receive.

Что же касается самого майнинг пула и как он становится участником одноранговой сети. Майнинг пул должен быть подключен к сети той криптовалюты, которую он добывает, чтобы он мог прослушивать/сообщать сообщения о создании нового блока. Подключение майнинг пула к сети криптовалюты задается в данном программном средстве через конфигурационные файлы. Это сделано с целью того, что разные криптовалюты имеют разные параметры и настройки для подключения.

В классе ServerSocket.cs предоставлены методы для установки/разрыва соединения, отправки и получения сообщений, выполнена обработка исключений, выключить сервер для сети типа «Клиент-Сервер». Все взаимодействие между «Майнинг пулом» и «Майнером» осуществляется с помощью протокола Stratum, реализация которого, как раз основана на работе с сокетами.

3.1.2 Протокол Stratum. Программные модули StratumServer.cs и StratumMiner.cs

Протокол Stratum представляет собой линейный протокол, использующий простой TCP-сокет, с полезной нагрузкой, закодированной как сообщения JSON. Клиент просто открывает TCP-сокет и пишет запросы на сервер в

виде сообщений JSON, завершенных символом новой строки. Каждая полученная клиентом строка снова является действительным фрагментом JSON, содержащим ответ.

Плюсы данного протокола: очень легко реализовать и отлаживать, потому что участники сети посылают сообщения в формате удобочитаемом человеком. Протокол отличается от многих других решений легко расширяемым, не нарушая обратной совместимости. Также JSON широко поддерживается на всех платформах, и у нынешних майнеров уже есть библиотеки JSON.

Отличительной особенностью от других протоколов в том, что тут нету связанных с HTTP накладных расходов. Но самым большим улучшением в сетях, основанных на HTTP, является тот факт, что сервер может управлять нагрузкой сам по себе, он может отправлять широковещательные сообщения майнерам в любое время без каких-либо долговременных обходных решений и проблем балансировки нагрузки.

Программный модуль `StratumServer.cs` вызывается через `WebServer` метод `Start()`. Алгоритм работы данного сервера представлен на рисунке 3.2. Также в данном модуле реализованы такие события как:

- `OnClientConnection` – событие о подключении нового клиента;
- `OnClientDisconnect` - событие о отключении клиента;
- `OnDataReceived` - событие о получении данных.

А также свойство `IsBanned`, с помощью которого мы можем узнать заблокирован ли наш клиент (майнер) и конструктор, который проинициализирует поля данного класса.

Программный модуль `StratumMiner.cs` выполняет функционал для майнера (клиента), такие как:

- `StratumMiner` - создать нового клиента;
- `OnClientDisconnect` - событие о отключении клиента;
- `OnDataReceived` - событие о получении данных.

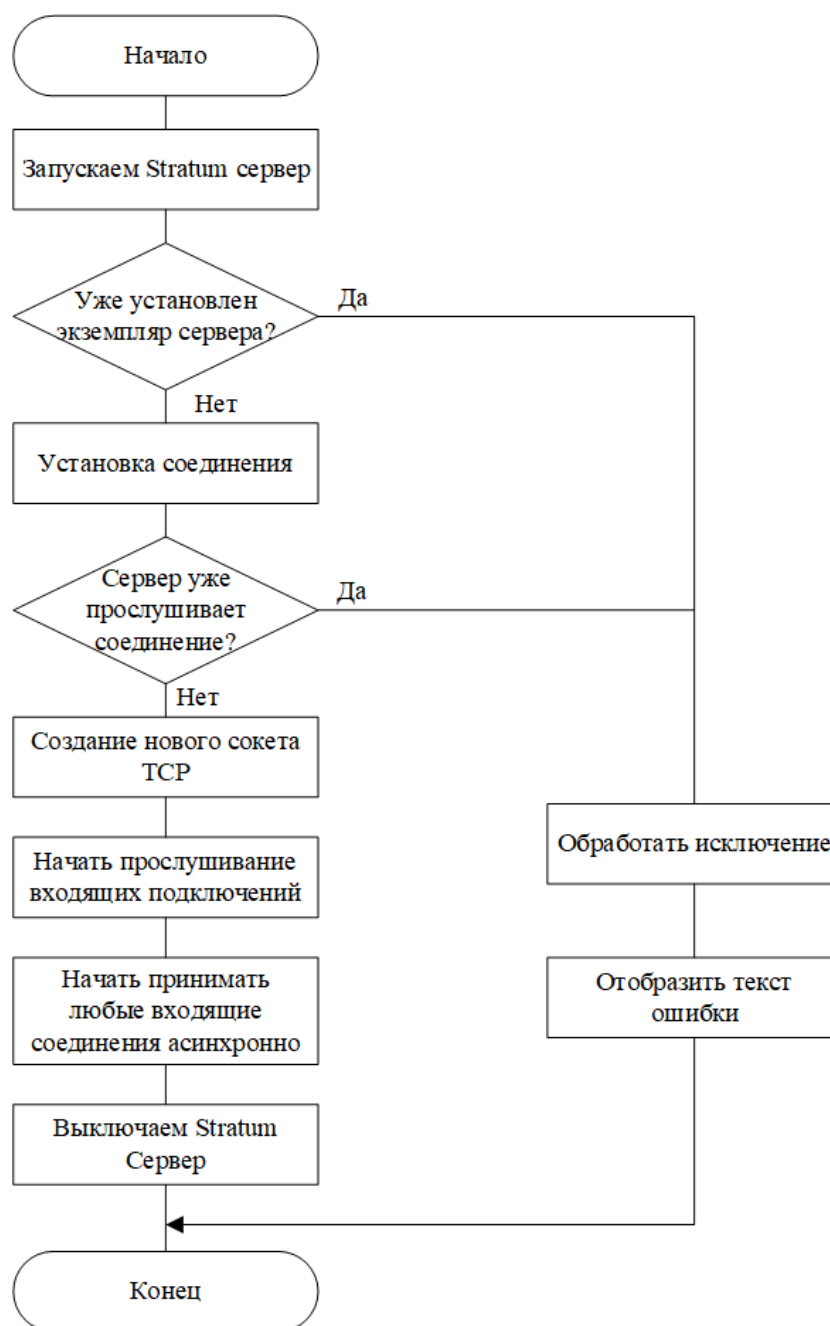


Рисунок 3.2 – Алгоритм работы модуля Stratum Server

А также свойство `IsBanned`, с помощью которого мы можем узнать заблокирован ли наш клиент (майнер), конструктор, который проинициализирует поля данного класса.

3.1.3 Разработка программного модуля `MinerManager.cs`

Данный модуль предоставляет функциональность для работы с майнером. С помощью данного класса можно создать экземпляр класса, т.е. сущность «Майнер», атрибуты которой представлена в таблице 3.1.

Таблица 3.1 – Атрибуты сущности «IMiner»

Атрибут	Описание
---------	----------

Id	Идентификатор майнера
Account	Сущность Account, чтобы связать пользователя и майнера
Pool	Сущность Pool, чтобы присвоить майнера определенному майнинг пулу
Authenticated	Флаг, который в состоянии true, если клиент прошел авторизацию
ValidShareCount	Переменная, которая показывает количество сгенерированных валидных хешей данным майнером
InvalidShareCount	Переменная, которая показывает количество сгенерированных неверных хешей данным майнером
Software	Сущность Software, которая содержит информацию о программном обеспечении майнера
SoftwareVersion	Версия программного обеспечения
Authenticate	Метод авторизации майнера

Модуль MinerManager.cs содержит список всех подключенных к одному пулу майнеров List<IMiner>. Также реализует следующие функции:

- GetMiner() – Достать майнера из списка;
- Create() – Создать майнера и добавить майнера в список List<IMiner>;
- Remove() – Удалить майнера и убрать его из списка List<IMiner>;
- Authenticate() – Выполнить аутентификацию майнера.

Методы Create и Remove вызываются по событиям OnClientConnection и OnClientВшыconnection соответственно рассмотренного выше модуля StartumServer.cs.

3.1.4 Разработка программного модуля ShareManager.cs

Сущность «Share» подразумевает под собой хеш код, который может быть, как валидным, так и нет, но только за валидный хеш клиенты могут получить прибыль. Стоит отметить, что если майнер добывает криптовалюту самостоятельно, то ему не надо иметь дело с «Share», так как вся прибыль достается ему. Следовательно, можно сделать вывод, что «Share» нам необходимы, чтобы мы могли узнать сколько каждый майнер сделал вычислительных операций, т.е. количество выполненных работы, чтобы узнать его вклад в создание нового блока. Таким образом можно поделить награду в соответствии с работой майнера.

Программный модуль ShareManager.cs обеспечивает нас функциональностью, такой как:

- ProcessShare служит для получения Share (алгоритм работы данной функции представлен на рисунке 3.3);
- HandleValidShare обрабатывает событие о том, что полученная Share является валидной;

- HandleInvalidShare обрабатывает событие о том, что полученная Share является невалидной;
- OnBlockFound обрабатывает событие, которое оповещает слушателей о том, что блок найден;
- SubmitBlock функция вызывается в событии HandleValidShare отправляет сгенерированный блок.

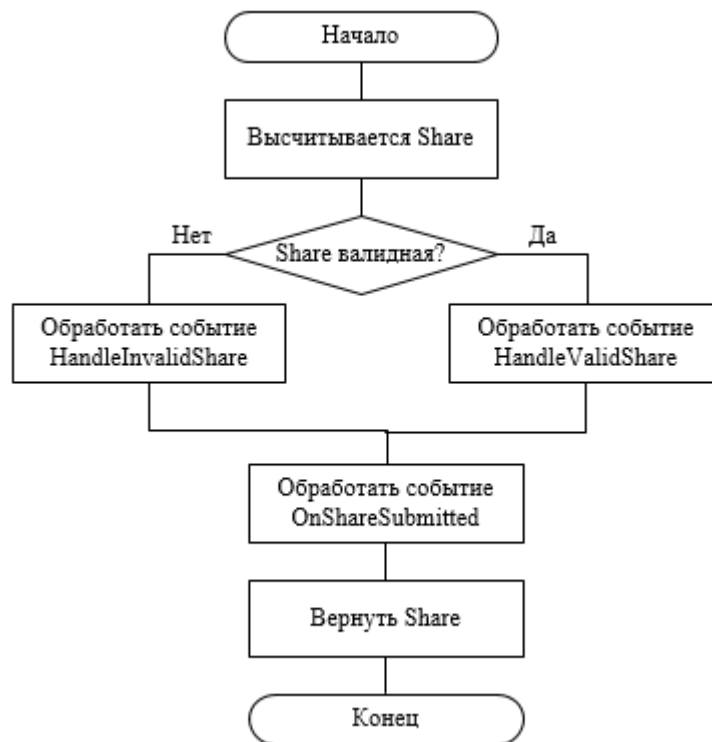


Рисунок 3.3 – Алгоритм процесса получения Share

3.1.5 Алгоритм распределение прибыли и вычислительной нагрузки.

За каждый добытый блок следует награда. В модуле GenerateTransaction.cs выполняется распределение прибыли между участниками: майнером и майнинг пулом. Алгоритм распределения прибыли представлен на рисунке 3.4.

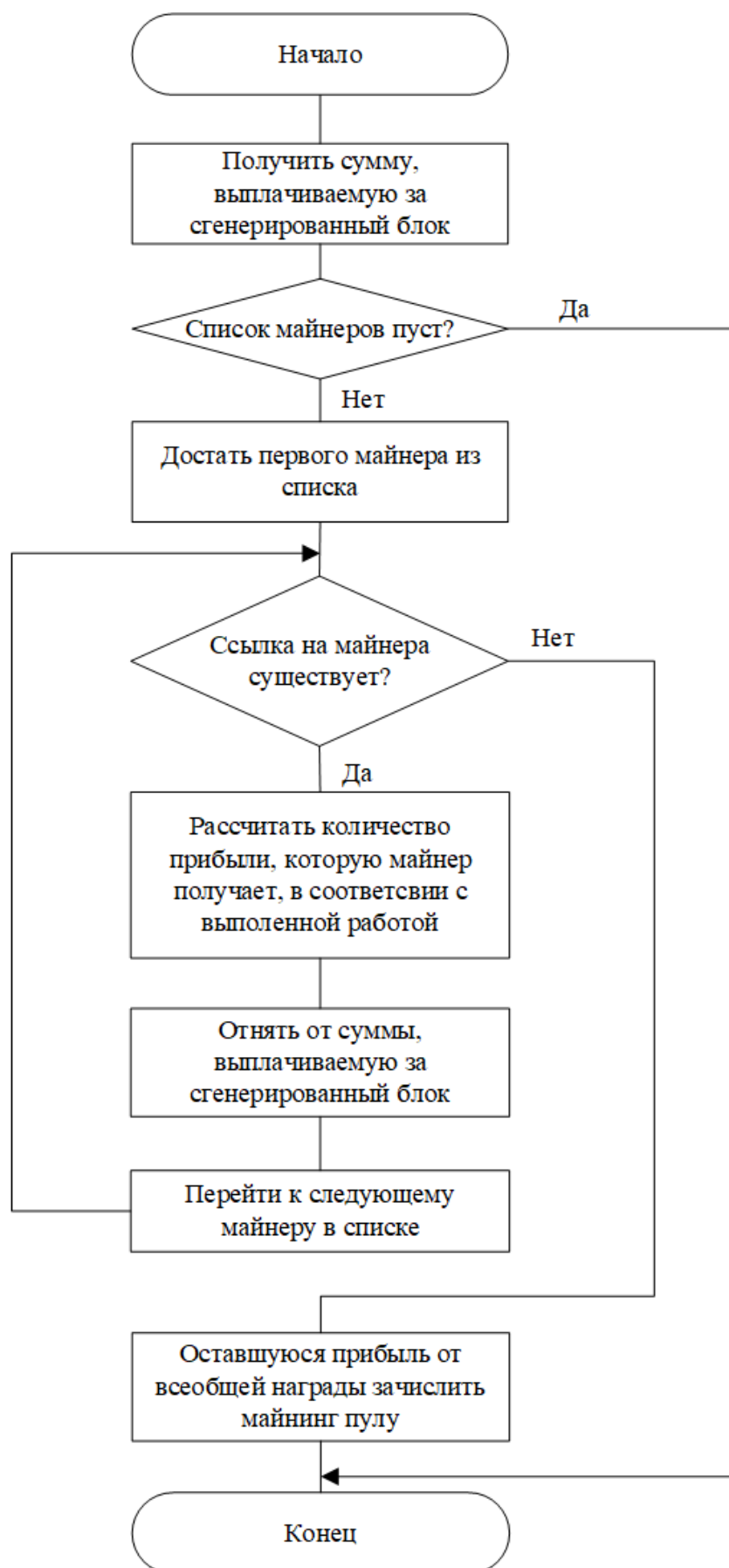


Рисунок 3.4 – Алгоритм распределения прибыли

Чтобы майнинг пулу распределить нагрузку между участниками, используется класс Range.cs. Данный класс содержит 3 поля: start, stop, step.

Сама сущность Range подразумевает диапазон чисел, которые используют майнеры для нахождения Share. Range будет использоваться в классе, который будет высчитывать хеш код и так как, чтобы найти валидный код необходимо методом перебора перебирать возможные значения share. Соответственно, у каждого майнера будет свой диапазон чисел для генерирования разных хешей. Майнер будет начинать с первого числа в диапазоне через заданный шаг до последнего числа.

3.1.6 Разработка программного модуля PoolConfig.cs

Одной из ключевых сущностей программного средства является майнинг пул. Структура пула представлена в таблице 3.2.

Таблица 3.2 – Атрибуты сущности «PoolConfig»

Атрибут	Описание
Id	Идентификатор майнинг пула
Enabled	Флаг, который если установлен в true, то пул доступен
Coin	Сущность монета
Wallet	Сущность, которая описывает электронный кошелек
Rewards	Сущность награда
Payments	Сущность платежи
MinerManager	Сущность MinerManager для нахождения всех майнеров, которые присоединились к майнинг пулу
Job	Сущность работа
Banning	Сущность, которая обеспечивает блокирование

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Современные программные средства достаточно сложные, выполняют много функций и содержат много зависимостей. Существует интеграционное тестирование, которое проверяет, что несколько компонентов системы работают вместе правильно. Тестирование — это контроль качества программного продукта. Это процесс, который длится на протяжении всего жизненного цикла ПО

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Юнит-тестирование – тестирование, которое помогает обнаружить ошибки быстрым и качественным способом.

Тестирование приложения проводилось на основании функциональных требований, представленных в разделе 2.3 дипломного проекта. При этом проверялось соответствие ожидаемых и полученных данных при выполнении запросов к базе данных.

Некоторые из тестов представлены в таблице 4.1.

Таблица 4.1 – Тестирование программного средства

Тестируемый зарос	Ожидаемый результат	Полученный результат
Создать майнинг пул со всеми валидными параметрами (PoolTests.cs)	Создание майнинг пула	Создание майнинг пула
Создать майнинг пул заданными параметрами (PoolTests.cs)	Обработка ошибки	Обработка ошибки
Запрос майнера на подключения к майнинг пулу (StratumServiceTests.cs)	Майнер подключился	Майнинг подключился
Запрос майнера на подключения к майнинг	Майнер подключился	Майнер подключился

пулу с необязательным параметром «signature» (StratumServiceTests.cs)		
Генерирование Share, которая будет являться кандидатом blockchain цепи. (ShareTests.cs)	В объекте Share установлен флаг IsBlockCandidate в true	В объекте Share установлен флаг IsBlockCandidate в true
Генерирование Share, которая будет являться кандидатом blockchain цепи. (ShareTests.cs)	В объекте Share установлен флаг IsBlockCandidate в true	В объекте Share установлен флаг IsBlockCandidate в false
Распределение прибыли между майнерами (OutputsTests.cs)	Прибыль распределена корректна	Прибыль распределена корректна
Распределение прибыли майнинг пулу (OutputsTests.cs)	Прибыль распределена корректна	Прибыль распределена корректна
Запуск сервера с не прописанными параметрами в конфигурационном файле (Pool.cs)	Обработка исключения. Вывод сообщения об ошибке: "Can't start pool as configuration is not valid."	Обработка исключения. Вывод сообщения об ошибке: "Can't start pool as configuration is not valid."
Запуск сервера с несуществующим или неверным hashAlgorithm	Обработка исключения. Вывод текста об ошибке сообщения: "Unknown hash algorithm, pool initialization failed"	Обработка исключения. Вывод текста об ошибке сообщения: "Unknown hash algorithm, pool initialization failed"
Подключение майнинг пула, при условии, что StratumServer включен и доступен (Pool.cs)	Майнинг пул подключен к сети	Майнинг пул подключен к сети
Подключение майнинг пула, при условии, что	Майнинг пул не подключен к сети. Вывод	Майнинг пул не подключен к сети. Вывод

StratumServer не включен или недоступен (Pool.cs)	сообщения: "No connected servers to network!"	сообщения: "No connected servers to network!"
Запрос на аутентификацию незарегистрированного майнера (MinerManager.cs)	Обработка исключения. Вывод сообщения: "Miner authentication failed"	Обработка исключения. Вывод сообщения: "Miner authentication failed"
Майнер отправляет запрос удалить соединение с сервером (SocketServer.cs RemoveConnection)	Соединение разорвано	Соединение разорвано
Майнер отправляет запрос удалить соединение с сервером, но соединение установлено не было до этого (SocketServer.cs RemoveConnection)	Обработка ошибки и вывод сообщения: «Exception connection does not exist.»	Обработка ошибки и вывод сообщения: «Exception connection does not exist.»
Создание нового TCP сокета (SocketServer.cs)	Сокет создан	Сокет создан
Создание нового TCP сокета с недоступным IP-адресом (SocketServer.cs)	Обработка исключения. Вывод сообщения: "Can not bind on IP-address, server shutting down.."	Обработка исключения. Вывод сообщения: "Can not bind on IP-address, server shutting down.."
Запуск сервера (StratumServer.cs)	Сервер начал прослушивание входящих сообщений. Вывод сообщения о успешном запуске: "Stratum server listening on the port"	Сервер начал прослушивание входящих сообщений. Вывод сообщения о успешном запуске: "Stratum server listening on the port"
Авторизация существующего майнера на основании его имени и пароля	Авторизация прошла успешно	Авторизация прошла успешно

Авторизация несуществующего майнера на основании его имени и пароля (StratumMiner.cs)	Обработка ошибки. Вывод текста сообщения: "Unauthorized worker: {username}"	Обработка ошибки. Вывод текста сообщения: "Unauthorized worker: {username}"

5 МЕТОДИКА ИСПОЛЬЗОВАНИЯ РАЗРАБОТАННОГО ПРОГРАММНОГО СРЕДСТВА

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОЕКТА

6.1 Описание проекта

Программное средство позволяет добывать криптовалюту, распределять вычислительную нагрузку между участниками и получать вознаграждение в виде криптовалюты, также участник может получать некоторую информацию о пуле, количестве участников.

Основной целью данного программного средства является добыча криптовалюты путем распределения вычислительной нагрузки между участниками, которые подключились к серверу. Как только один из участников находит нужный ключ, то блок транзакций присоединяется к цепочке, а прибыль делится между всеми участниками в соответствии с потраченной мощностью. У данного программного средства шансы найти нужный ключ выше, чем у участника, который решил добывать криптовалюту в одиночку.

Участники, использующие программное средство могут, в свою очередь, получать релевантную информацию о добытой криптовалюте, каким алгоритмом выполнялись вычисления, сколько участников принимали участие, сумма вознаграждения за участие.

Основные функции, выполняемые системами для распределения вычислительной нагрузки между участниками:

- подключиться клиенту к майнинг пулу;
- отключиться клиенту от майнинг пула;
- распределить награду;
- распределить вычислительную нагрузку;
- выбор криптовалюты;
- редактирование личной информации;
- конвертация денег в другие валюты;
- управление мощностью компьютера;
- просмотр документации;
- просмотр информации о майнинг пуле.

Основная задача данного раздела дипломного проекта – подтвердить актуальность и экономическую целесообразность разработки данного программного средства программного обеспечения (ПО) и его использования потенциальными пользователями. Раздел включает расчет следующих показателей:

1 Расчёт сметы затрат и цены ПО. Упрощённый расчет затрат на разработку ПО делается в разрезе следующих статей: затраты на основную заработную плату разработчиков, затраты на дополнительную заработную плату разработчиков, отчисления на социальные нужды, прочие затраты.

2 Расчёт экономического эффекта от применения программного средства у пользователя (заказчика). Использование ПО напрямую влияет на экономические показатели деятельности пользователя (заказчика). Данный эффект поддается стоимостной оценке и рассчитывается при экономическом

обосновании.

6.2 Расчёт сметы затрат и цены ПО

Разрабатываемый программный продукт относится к первой категории сложности, поскольку режим работы ПО проходит в реальном времени.

Программный продукт является ПО общего назначения и относится к категории новизны В ($K_n = 0,7$).

Отправной точкой для расчёта плановой сметы затрат на разработку ПО, требуется определить общий объем программного продукта (V_o). В качестве единицы измерения примем количество строк исходного кода (Lines of Code, LOC). Прогнозируемый общий объём ПО определяется по каталогу функций. Каталог функций данного программного продукта представлен в таблице 6.1.

Таблица 6.1 – Перечень и объём функций программного средства

№ функции	Наименование (содержание)	Объём функции (LOC)
101	Организация ввода информации	100
205	Обслуживание базы данных в пакетном режиме	1030
207	Манипулирование данными	8400
405	Система настройки ПО	250
507	Обеспечение интерфейса между компонентами	730
703	Расчет показателей	410
704	Процессор отчетов	1070
707	Графический вывод результатов	300

Общий объем программного продукта определяется исходя из количества и объёма функций, реализованных в программе формула:

$$V_o = \sum_{i=0}^n V_i, \quad (6.1)$$

где V_i – объём отдельной функции ПС;
 n – общее число функций.

На основе данных, приведённых в таблице 6.1, общий объём ПО составил **12290** строк кода. Нормативная трудоёмкость составит 312 человеко-дней.

Далее определяется нормативная трудоёмкость, которая высчитывается по формуле:

$$T_o = T_n \cdot K_c \cdot K_T \cdot K_n, \quad (6.2)$$

где K_c – коэффициент, учитывающий сложность ПС;

K_T – поправочный коэффициент, учитывающий степень использования при разработки стандартных модулей;

K_H – коэффициент, учитывающий степень новизны ПС.

$$K_c = 1 + \sum_{i=1}^n K_i, \quad (6.3)$$

где K_i – коэффициент, соответствующий степени повышения сложности ПО за счёт конкретной характеристики;

n – количество учитываемых характеристик.

Для проектируемого программного средства коэффициент сложности составляет $K_c = 1 + 0,08 + 0,07 + 0,12 = 1,27$.

Указанный коэффициент для разрабатываемого приложения $K_T = 0,8$, так как разрабатываемое ПО использует стандартные компоненты. Также для разрабатываемого ПО $K_H = 0,7$, так как разрабатываемое ПО не является новым (существуют аналоги). Таким образом, можно рассчитать общую трудоёмкость разработки по формуле:

$$T_o = 312 \cdot 1,27 \cdot 0,7 \cdot 0,7 \approx 194 \text{ чел./дн.} \quad (6.4)$$

На основе общей трудоёмкости и требуемых сроков реализации проекта вычисляется плановое количество исполнителей. Численность исполнителей проекта рассчитывается по формуле:

$$Ч_p = \frac{T_o}{T_p \cdot \Phi_{эф}}, \quad (6.5)$$

где T_o – общая трудоёмкость разработки проекта, чел./дн.;

$\Phi_{эф}$ – эффективный фонд времени работы одного работника в течение года, дн.;

T_p – срок разработки проекта, лет.

Эффективный фонд времени работы одного разработчика вычисляется по формуле:

$$\Phi_{эф} = D_{г} - D_{п} - D_{в} - D_{о}, \quad (6.6)$$

где $D_{г}$ – количество дней в году, дн.;

$D_{п}$ – количество праздничных дней в году, не совпадающих с выходными днями, дн.;

$D_{в}$ – выходных дней в году, дн.;

$D_{о}$ – количество дней отпуска, дн.

Согласно данным, приведённым в производственном календаре для пятидневной рабочей недели году для Беларуси, фонд рабочего времени составит 231 день. Расчет $\Phi_{эф}$ приведен в следующей формуле:

$$\Phi_{эф}=366-9-102-24=231 \text{ дн.} \quad (6.7)$$

Учитывая срок разработки проекта $T_p = 4 \text{ мес.} = 0,33 \text{ года}$, общую трудоёмкость и фонд эффективного времени одного работника, вычисленные ранее, можно рассчитать численность исполнителей проекта:

$$\chi_p = \frac{194}{0,33 \cdot 231} \approx 2 \text{ человека.} \quad (6.8)$$

6.3 Расчёт заработной платы исполнителей

Основной статьей расходов на создание ПО является заработная плата разработчиков (исполнителей) проекта. Основная заработная плата исполнителей рассчитывается по формуле:

$$Z_o = \sum_{i=1}^n Z_{ci} \cdot \Phi_{pi} \cdot K, \quad (6.9)$$

где Z_{ci} – среднедневная заработная плата i -го исполнителя, руб;
 n – количество исполнителей, занятых в разработке ПС;
 Φ_{pi} – плановый фонд рабочего времени i -го исполнителя, дн.;
 K – коэффициент премирования.
 Для расчета Z_{ci} применяется формула:

$$Z_{ci} = \frac{O_i}{D_c}, \quad (6.10)$$

где O_i – должностной оклад конкретного специалиста;
 D_c – среднемесячное количество рабочих дней, дн.
 В свою очередь месячный тарифный оклад одного работника вычисляется по формуле:

$$O_i = T_{m1} \cdot T_{ki}, \quad (6.11)$$

где T_{m1} – месячная тарифная ставка первого разряда, руб.;
 T_{ki} – тарифный коэффициент определенного работника.

С 1 марта 2018 года для бюджетных организаций утверждена тарифная ставка 34 руб. С учетом тарифного коэффициента для работников по квалификации инженер-программист (2,84), месячный оклад одного работника составляет:

$$O_i = 34 * 2,84 = 96,56 \text{ руб.} \quad (6.12)$$

Подставив полученный месячный оклад, получим среднедневную заработную плату по формуле:

$$З_{ci} = \frac{96,56}{21} = 4,59 \text{ руб.} \quad (6.13)$$

Эффективный фонд рабочего времени составляет 4 месяца или 84 рабочих дней. Коэффициент премирования — 1,5. Из всех данных, вычисленных выше, получим основную заработную плату исполнителей по формуле:

$$З_o = 4,59 * 3 * 84 * 1,5 = 1735,02 \text{ руб.} \quad (6.14)$$

Дополнительная заработная плата работников (20%) рассчитывается по формуле:

$$З_{di} = \frac{1735,02 * 20}{100} = 347 \text{ руб.} \quad (6.15)$$

Отчисления в фонд социальной защиты (35%) и по обязательному страхованию (0,6%) рассчитывается в следующих формулах:

$$З_{czi} = \frac{(1735,02 + 347) * 35}{100} = 728,71 \text{ руб.} \quad (6.16)$$

$$З_{oci} = \frac{(1735,02 + 347) * 0,6}{100} = 12,49 \text{ руб.} \quad (6.17)$$

Размер затрат на расходные материалы (3%) составит:

$$M_i = \frac{1735,02 * 3}{100} = 52,01 \text{ руб.} \quad (6.18)$$

Расходы по статье «Машинное время» определяются:

$$P_{mi} = 1,2 * \frac{12290 * 0,12}{100} = 17,70 \text{ руб.} \quad (6.19)$$

Расходы по статье «Научные командировки» (30%) составят:

$$P_{\text{нкi}} = \frac{1735,02 \cdot 30}{100} = 520,51 \text{ руб.} \quad (6.20)$$

Расходы по статье «Прочие затраты» (20%) включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы и определяются:

$$P_{\text{зи}} = \frac{1735,02 \cdot 20}{100} = 347 \text{ руб.} \quad (6.21)$$

Затраты по статье «Накладные расходы» связаны с необходимостью содержания аппарата управления, вспомогательных хозяйств, а также остальные общехозяйственные нужды. Затраты по статье «Накладные расходы» (100%) составят:

$$P_{\text{ни}} = \frac{1735,02 \cdot 100}{100} = 1735,02 \text{ руб.} \quad (6.22)$$

Общая сумма расходов по всем статьям сметы ($C_{\text{пi}}$) на ПС рассчитывается по формуле:

$$C_{\text{пi}} = Z_{\text{oi}} + Z_{\text{дi}} + Z_{\text{сzi}} + P_{\text{ci}} + P_{\text{ми}} + M_i + P_{\text{нкi}} + P_{\text{зи}} + P_{\text{ни}}, \quad (6.23)$$

где P_{ci} – расходы на специальное оборудование.

В связи с тем, что специальное оборудование и специальные программы не были приобретены, то данное значение данных расходов равно нулю. Тогда общая сумма расходов по всем статьям сметы на разработку ПС составит:

$$C_{\text{пi}} = 1735,02 + 347 + 728,71 + 12,49 + 52,01 + 17,70 + 520,51 + 347 + 1735,02 = 5495,46 \text{ руб.} \quad (6.24)$$

6.4 Расчет рентабельности программного средства

Рентабельность программного средства определяется исходя из результатов анализа рыночных условий, переговоров с заказчиком, согласования требований, возможная корректировка требований и согласования с ним отпускной цены.

Таблица 6.2 – Исходные данные для расчета рентабельности программного средства

Прогнозируемый уровень рентабельности, %	Y_{pp}	40
Норматив НДС, %	$H_{дс}$	20
Норматив расходов на сопровождение программного средства, %	H_c	20
Норматив расходов на освоение программного средства, %	H_o	10

Прибыль от программного средства рассчитывается по формуле:

$$\Pi_{oi} = \frac{C_{pi} \cdot Y_{ppi}}{100}, \quad (6.25)$$

где Y_{ppi} – уровень рентабельности программного средства;

C_{pi} – себестоимость программного средства.

Применив формулу (6.25) к расчету прибыли от программного средства получим:

$$\Pi_{oi} = \frac{5495,46 \cdot 40}{100} = 2198,18 \text{ руб.} \quad (6.26)$$

Для расчета цены на программное средство без налога вычисляется сначала прогнозируемая цена без налога. Для этого применяется формула:

$$\Pi_{pi} = C_{pi} + \Pi_{oi} \quad (6.27)$$

Таким образом, прогнозируемая цена без налога равна:

$$\Pi_{pi} = 5495,46 + 2198,18 = 7693,64 \text{ руб.} \quad (6.28)$$

Для расчета налога на добавленную стоимость применяется формула:

$$HDC_i = \frac{\Pi_{pi} \cdot HDC}{100}, \quad (6.29)$$

где HDC – норматив налога на добавленную стоимость.

Учитывая, что норматив налога на добавленную стоимость равен 20%, то сам налог равен 1538,73 рублей по формуле:

$$HDC_i = \frac{7693,64 \cdot 20}{100} = 1538,73 \text{ руб.} \quad (6.30)$$

Прогнозируемая отпускная цена на программное средство вычисляется по формуле:

$$\Pi_{oi} = \Pi_{pi} + \text{НДС}_i \quad (6.31)$$

Таким образом, прогнозируемая отпускная цена:

$$\Pi_{oi} = 7693,64 + 1538,73 = 9232,37 \text{ руб.} \quad (6.32)$$

Кроме того, организация-разработчик осуществляет затраты на освоение и сопровождение ПС, которые определяются по формулам:

$$P_{oi} = \frac{C_{pi} \cdot H_o}{100}, \quad (6.33)$$

$$P_{ci} = \frac{C_{pi} \cdot H_c}{100}. \quad (6.34)$$

где H_o – норматив расходов на освоение.

H_c – норматив расходов на сопровождение.

С учетом того, что норматив расходов на освоение равен 10%, затраты на освоение (6.35) и сопровождение (6.36) будут равны соответственно:

$$P_{oi} = \frac{5495,46 \cdot 10}{100} = 549.55 \text{ руб.}, \quad (6.35)$$

$$P_{ci} = \frac{5495,46 \cdot 20}{100} = 1099.09 \text{ руб.} \quad (6.36)$$

6.5 Оценка экономической эффективности применения программного средства у пользователя

Общие капитальные вложения заказчика, связанные с приобретением, внедрением и использованием программного средства рассчитываются по формуле:

$$K_o = K_{пр} + K_{ос} + K_c + K_{тс} + K_{об}, \quad (6.37)$$

где $K_{пр}$ – затраты пользователя на приобретение программного средства по отпускной цене у разработчика с учетом стоимости услуг по эксплуатации;

$K_{ос}$ – затраты пользователя на освоение программного средства;

K_c – затраты пользователя на оплату услуг по сопровождению программного средства;

$K_{тс}$ – затраты на доукомплектование ВТ техническими средствами в связи с внедрением нового программного средства;

$K_{об}$ – затраты на пополнение оборотных средств в связи с использованием нового ПО. В данном случае, значение $K_{об}$ составит 5% от $K_{пр}$.

Таблица 6.3 – Исходные данные для расчета экономии ресурсов в связи с применением нового программного средства

Наименование показателей	Усл. обозн.	Ед. изм.	Значение показателя		Наименование источника информации
			в базовом варианте	в новом варианте	
1. Капитальные вложения, включая стоимость услуг по сопровождению и адаптации ПС	$K_{пр}, (K_c)$	руб.	-	9232,37	Договор заказчика с разработчиком
2. Затраты пользователя на освоение ПО	$K_{ос}$	руб.	-	1099,09	Смета затрат на внедрение
3. Затраты на доукомплектование ВТ техническими средствами в связи с внедрением нового ПС	$K_{тс}$	руб.	-	0	Смета затрат на внедрение
4. Затраты на пополнение оборотных фондов, связанных с эксплуатацией нового ПС	$K_{об}$	руб.	-	420	Смета затрат на внедрение
5. Время простоя сервиса, обусловленное ПО, в день	$П_1, П_2$	мин	50	10	Расчётные данные пользователя и паспорт
6. Стоимость одного часа простоя	$C_{п}$	руб.	30,1	30,1	Расчётные данные пользователя и паспорт
7. Среднемесячная ЗП одного программиста	$З_{см}$	руб.	1730	1730	Расчётные данные пользователя

8. Коэффициент начислений на зарплату	$K_{\text{нз}}$		1,5	1,5	Принято для расчета
9. Среднемесячное количество рабочих дней	D_p	день	21.5		Принято для расчета
10. Количество типовых задач, решаемых за год	Z_{T1}, Z_{T2}	задача	4000	4200	План пользователя
11. Объем работ, выполняемый при решении одной задачи	A_1, A_2	задача	4000	4200	План пользователя
12. Средняя трудоемкость работ на задачу	T_{c1} T_{c2}	человеко-часов	0,7	0,2	Рассчитывается по данным пользователя
13. Количество часов работы в день	$T_{\text{ч}}$	ч.	8		Принято для расчета
15. Ставка налога на прибыль	$H_{\text{п}}$	%		1 8	Утверждено законодательством

Экономия затрат на заработную плату при использовании нового программного средства (C_3) в расчете на объем выполненных работ рассчитывается по формуле:

$$C_3 = C_{3\text{е}} \cdot A_2, \quad (6.39)$$

где $C_{3\text{е}}$ – экономия затрат на заработную плату при решении задач с использованием нового ПС в расчете на 1 задачу;

A_2 – объем выполненных работ с использованием нового ПС.

Экономия затрат на заработную плату ($C_{3\text{е}}$) рассчитывается по формуле:

$$C_{3\text{е}} = \frac{Z_{\text{см}} \cdot (T_{c1} - T_{c2})}{D_p \cdot T_{\text{ч}}}, \quad (6.40)$$

где $Z_{\text{см}}$ – среднемесячная заработная плата одного программиста;

T_{c1}, T_{c2} – снижение трудоемкости;

$T_{\text{ч}}$ – количество часов работы в день;

D_p – среднемесячное количество рабочих дней.

Таким образом, экономия затрат на заработную плату составит 5,16 рубля по формуле:

$$C_{ze} = \frac{1730 \cdot (0,7 - 0,2)}{21,5 \cdot 8} = 5,03 \text{ руб.} \quad (6.41)$$

Тогда экономия затрат на заработную плату при использовании нового ПС (C_3) будет равна по формуле 21687,75 рубля:

$$C_3 = 5,03 \cdot 4200 = 21126 \text{ руб.} \quad (6.42)$$

Экономия с учётом начисления на зарплату (C_H) будет равна:

$$C_H = 21126 \cdot 1,5 = 31689 \text{ руб.} \quad (6.43)$$

Экономия за счёт сокращения простоев сервиса (C_c) будет равна:

$$C_c = \frac{(П1 - П2) \cdot Др \cdot Сп}{60},$$

$$C_c = \frac{40 \cdot 21,5 \cdot 30,1}{60} = 431,43 \text{ руб.} \quad (6.44)$$

Общая годовая экономия текущих затрат, связанных с использованием нового ПС (C_o) вычисляется по формуле:

$$C_o = C_H + C_c, \quad (6.45)$$

И составит 32120,43 руб. по формуле:

$$C_o = 31689 + 431,43 = 32120,43 \text{ руб.} \quad (6.46)$$

Чистая прибыль от экономии текущих затрат высчитывается по формуле:

$$\Delta\Pi_{\text{ч}} = C_o - \frac{C_o \cdot H_{\text{п}}}{100}, \quad (6.47)$$

где $H_{\text{п}}$ – ставка налога на прибыль.

Размер чистой прибыли от экономии текущих затрат, с учетом ставки налога на прибыль, равен:

$$\Delta\Pi_{\text{ч}} = 32120,43 - \frac{32120,43 \cdot 18}{100} = 26338,75 \text{ руб.} \quad (6.48)$$

В процессе использования нового программного средства чистая прибыль в конечном итоге возмещает капитальные затраты. Однако, полученные

при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2018 год) путем умножения результатов и затрат за каждый год на коэффициент приведения. В данном примере используются коэффициенты: 2018 г. – 1, 2019-й – 0,869. Все рассчитанные данные экономического эффекта сводятся в таблицу 6.4.

Таблица 6.4 – Расчет экономического эффекта от использования нового программного средства

Показатели	Ед. изм.	2018	2019
Результат:			
1. Прирост прибыли за счет экономии затрат ($\Pi_{\text{ч}}$)	руб.	26338,75	26338,75
2. Прирост прибыли за счет экономии затрат ($\Pi_{\text{ч}}$) с учетом фактора времени	руб.	26338,75	22904,18
Затраты:			
3. Приобретение, адаптация и освоение ПС ($K_{\text{пр}}$)	руб.	9232,37	—
4. Освоение ПС ($K_{\text{ос}}$)	руб.	1099,09	—
5. С учетом фактора времени	руб.	10331,46	—
Экономический эффект			
6. Превышение результата над затратами	руб.	-10331,46	22904,18
7. С нарастающим итогом	руб.	-10331,46	12572,72
8. Коэффициент приведения	ед.	1	0,8696

Из указанной таблицы видно, что все затраты заказчика окупятся на первый год эксплуатации программного средства.

6.6 Выводы по технико-экономическому обоснованию

В результате технико-экономического обоснования применения программного продукта были получены следующие значения показателей их эффективности:

- экономический эффект за год работы программного средства составит 13173,52 руб;
- затраты на разработку и внедрение программного средства окупятся на первый год его использования.

Чистая прибыль от реализации ПС ($\Delta\P_{\text{ч}} = 26338,75$ руб.) остаётся организации-разработчику и представляет собой экономический эффект от создания нового программного средства. Положительный экономический эффект главным образом достигается за счёт уменьшения трудоёмкости работ пользователей в расчёте на одну задачу.

Продукт является экономически выгодным, так как он окупается за год эксплуатации, что означает экономическую целесообразность данной разработки.

ЗАКЛЮЧЕНИЕ

В ходе преддипломной практики была проделана работа, в ходе которой я изучила такие понятия как: «криптовалюта», «блокчейн», «майнинг», «майнер», «транзакции», «майнинг пул», были рассмотрены алгоритмы шифрования информации: синхронное, асинхронное, комбинированное. Также изучен цикл добычи криптовалют, а также некоторые принципы проектирования.

Я провела анализ аналогов на IT-рынке, выявила слабые и сильные стороны каждого, чтобы предотвратить на этапе проектирования ошибки проектирования и выявлены некоторые требования:

- подключение к майнинг пулу;
- выбор криптовалюты;
- редактирование личной информации;
- конвертация денег в другие валюты;
- управление мощностью компьютера;
- просмотр документации;
- просмотр статистики;
- просмотр информации о майнинг пуле.

Был проведен анализ технологий для разработки программного средства, в ходе которого был сделан выбор:

- платформа ASP.Net;
- язык программирования для серверной части C#;
- язык программирования для клиентской части JavaScript.

За период преддипломной практики были разработаны модули программного средства для добычи криптовалют, алгоритмы для нахождения эффективного хеша, организация сети между участников для распределения вычислительной нагрузки, что приведет к ускоренному процессу добычи криптовалют.

В дальнейшем будет проводиться оптимизация алгоритмов вычислений, добавление нового функционала, рефакторинг кода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. <https://prostocoin.com/blog/what-is-mining>
2. <https://learn.javascript.ru/intro>
<https://metanit.com/sharp/mvc5/1.2.php>
<http://mtblog.mtbank.by/chto-takoe-kriptovalyuta-i-kak-ee-zarabotat-razvernutyj-putevoditel-v-voprosah-i-otvetah/> - криптовалюта 1 раздел
<https://golos.io/ru--golos/@aleco/prosto-i-dostupno-o-blockchain-chto-eto-i-kak-rabotaet> - блокчейн 1 раздел
<https://habrahabr.ru/post/98323/> - шифрование 1 раздел
<https://probtc.info/materialy/30642/> - распределение прибыли 1 раздел
3. Radeck K. C# and Java: Comparing Programming Languages [Электронный ресурс] // MSDN: [сайт]. URL: <http://msdn.microsoft.com/en-us/library/ms836794.aspx>
4. Kurniawan B. Comparing C# and Java [Электронный ресурс] // O'Reilly Media: [сайт]. URL: <http://www.windowsdevcenter.com/lpt/a/889>
5. Chandra S.S., Chandra K. A comparison of Java and C# // Journal of Computing Sciences in Colleges, No. 20, 2005. pp. 238-254.
6. Rowe G.W. From Java to C#. Addison Wesley, 2004. 204-206 pp.
7. Johnson M. C#: A language alternative or just J--?, Part 2 [Электронный ресурс] // JavaWorld: [сайт]. [2000]. URL: http://www.javaworld.com/cgi-bin/mailto/x_java.cgi?pagetosend=/export/home/httpd/javaworld/javaworld/jw-12-2000/jw-1221-csharp2.html&pagename=/javaworld/jw-12-2000/jw-1221-csharp2.html&pageurl=http://www.javaworld.com/javaworld/jw-12-2000/jw-1221-csharp2.
8. Krikorian R. Contrasting C# and Java Syntax [Электронный ресурс] // O'Reilly Media: [сайт]. URL: <http://www.windowsdevcenter.com/lpt/a/929>
9. sealed [Электронный ресурс] // Справочник по C#: [сайт]. URL: <http://msdn.microsoft.com/ru-ru/library/88c54tsw.aspx>
10. Generics in C#, Java, and C++ [Электронный ресурс] URL: <http://www.artima.com/intv/genericsP.html>
11. Balagurusamy E. Programming in C#: A Primer. Tata McGraw-Hill, 2008. 8 pp.
12. Johnson M. C#: A language alternative or just J--?, Part 1 [Электронный ресурс] // JavaWorld: [сайт]. [2000]. URL: http://www.javaworld.com/cgi-bin/mailto/x_java.cgi?pagetosend=/export/home/httpd/javaworld/jw-11-2000/jw-1122-csharp1.html&pagename=/jw-11-2000/jw-1122-csharp1.html&pageurl=http://www.javaworld.com/jw-11-2000/jw-1122-csharp1.html&site=jw_core

13. Anders Hejlsberg B.V.B.E. The Trouble with Checked Exceptions [Электронный ресурс] [2003]. URL: <http://www.artima.com/intv/handcuffs.html>
14. cs2j [Электронный ресурс] // github.com: [сайт]. URL: <https://github.com/twiglet/cs2j>
15. CS2J: The User Guide [Электронный ресурс] URL: <http://www.cs2j.com/documentation/>
16. var (справочник по C#) [Электронный ресурс] // MSDN: [сайт]. URL: <https://msdn.microsoft.com/ru-ru/library/bb383973.aspx>
17. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. - Введ. 01.01.1992, М: Изд-во стандартов, 1991.
18. Глухова, Л.А. Основы алгоритмизации и программирования: лаб. практикум для студ. спец. 1-40 01 01 "Программное обеспечение информационных технологий" дневной формы обуч. В 4 ч / Л. А. Глухова, Е. П. Фадеева, Е. Е. Фадеева. - Минск: БГУИР, 2004.
19. Доманов, А.Т. Стандарт предприятия. Дипломные проекты (работы). Общие требования / А. Т. Доманов, Н. И. Сорока - Минск: БГУИР, 2010

