

Титульник

РЕФЕРАТ

ПРОГРАММНОЕ СРЕДСТВО ДОБЫЧИ КРИПТОВАЛЮТ: ОРАГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОЙ СЕТИ МЕЖДУ УЧАСТНИКАМИ: дипломный проект / Е. В. Борикова – Минск: БГУИР, 2018, - п.з. –118 с., чертежей (плакатов) – 6 л. формата А1.

Объектом изучения и разработки является приложение для добычи криптовалют.

Целью проекта является разработка программного средства для организации вычислительной сети между участниками, распределении прибыли.

При разработке и внедрении программного средства использовался такой стек технологий как .NET Framework и C#.

В первом разделе выдвигаются общие требования к созданию программного средства, устанавливается предметная область разработки, проводится обзор существующих аналогов приложений для добычи криптовалют, алгоритмов распределения прибыли и различные виды архитектуры организации сети, а также определяются перспективы развития первой версии продукта.

Во втором разделе отображается разработка функциональных требований, анализируются используемые при непосредственной реализации программного средства технологии.

Третий раздел посвящен проектированию и реализации программного средства.

Четвертый раздел содержит информацию о тестировании программного средства.

В пятом разделе описана методика использования программного средства.

В шестом разделе приведено технико-экономическое обоснование эффективности разработки и внедрения программного средства.

В разделе заключение содержатся краткие выводы по дипломному проекту.

Дипломный проект является завершенным, поставленная задача решена в полной мере, присутствует возможность дальнейшего развития программного средства и расширение его функциональности.

Дипломный проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности дипломного проекта соответствует норме, установленной кафедрой и составляет 90%.

T3

СОДЕРЖАНИЕ

Введение.....	8
1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству	10
1.1 Основные направления развития.....	10
1.2 Обзор существующих аналогов.....	19
1.3 Постановка задачи.....	24
2 Моделирование предметной области.....	26
2.1 Разработка функциональной модели предметной области	26
2.2 Используемые технологии	28
2.3 Спецификация функциональных требований	36
3 Проектирование и разработка программного средства	37
3.1 Проектирование архитектуры программного средства	37
3.2 Программный модуль «Организация сети»	38
3.3 Программный модуль «Протокол Stratum».....	42
3.4 Программные модули «Майнер» и «Майнер-менеджер».....	44
3.5 Программный модуль «Операции с хеш кодом».....	45
3.6 Программный модуль «Распределение прибыли»	47
3.7 Программный модуль «Распределение вычислительной нагрузки между участниками»	49
3.8 Разработка программного модуля «Майнинг пул»	51
4 Тестирование программного средства.....	53
5 Методика использования разработанного программного средства	59
6 Техничко-экономическое обоснование проекта	64
6.1 Описание проекта.....	64
6.2 Расчёт сметы затрат и цены ПО.....	65
6.3 Расчёт заработной платы исполнителей	67
6.4 Расчет рентабельности программного средства	70
6.5 Оценка экономической эффективности применения программного средства у пользователя	72
6.6 Выводы по технико-экономическому обоснованию	77
Заключение	78
Список использованных источников	80
Приложение А (обязательное) Исходный текст программы.....	82

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

Криптовалюта - разновидность цифровой валюты, которая существует только в Интернете. Создание и контроль за данной валютой базируются на криптографических методах.

Майнинг (от англ. слова mining — добыча полезных ископаемых) — процесс добычи криптовалют, т.е. деятельность по созданию новых структур (обычно речь идёт о новых блоках в блокчейне)

Майнер – физическое устройство, которое занимается майнингом.

Блокчейн (англ. blockchain или block chain) – выстроенная по определённым правилам непрерывная последовательная цепочка блоков, содержащих информацию. Чаще всего копии цепочек блоков хранятся на множестве разных компьютеров независимо друг от друга.

Майнинг ферма – физическое устройство, например, компьютер, с подключенными к нему видеокартами.

Майнинг пул – объединение мощности оборудования сразу многих майнеров для повышения вероятности нахождения нового блока.

Раунд – период времени между двумя блоками, найденными в майнинг пулом. Он может длиться от нескольких секунд до многих часов, в зависимости от удачи пула.

Рабочий (worker) – это имя для вашего майнинг устройства, которое на сайтах используете как логин для авторизации.

Хэш скорость (hash rate) – количество хешей, генерируемых в секунду майнером, или майнинг пулом, или сетью, обычно используется для отображения статистики (графиков).

Хэш код (Share) – это единица вычислительной работы, выполненная майнером. Когда майнер подключается к пулу, он получает вычислительную задачу для решения - он вычисляет значения хэша с определенными свойствами (они должны быть ниже предела, полученного из сложности). Хеши, удовлетворяющие требованиям, отправляются обратно в пул и используются в качестве доказательства работы майнера.

Старая ставка – это количество акций, отправленных после того, как старый блок уже найден, и пул переместился в следующий блок.

Одноранговая, децентрализованная или пиринговая сеть – это компьютерная сеть, основанная на равноправии участников. Часто в такой сети отсутствуют выделенные серверы, а каждый узел является как клиентом, так и выполняет функции сервера

Программный модуль – программа или функционально завершенный фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

Кроссплатформенность – способность программного средства работать более чем на одной аппаратной или программной платформе.

Фреймворк – сторонний программный компонент, задающий правила разработки архитектуры программного средства и напрямую влияющий на поведение приложения.

Веб-сервер – удаленный компьютер с установленным программным средством, реализующим API для обработки входящих HTTP запросов и формирования ответов с целью реализации функционирования клиентских приложений.

API – набор команд, функций и объектов, доступных программисту для реализации программных средств и/или коммуникации этих средств с внешними системами.

SOLID – аббревиатура пяти основных принципов дизайна классов в объектно-ориентированном проектировании: «Single responsibility» (принцип единственной ответственности), «Open-closed» (принцип открытости/закрытости), «Liskov substitution» (принцип подстановки Барбары Лисков), «Interface segregation» (принцип разделения интерфейса) и «Dependency inversion» (принцип инверсии зависимостей).

DRY – Do not repeat yourself (не повторяй себя)

YAGNI – You aren't gonna need it (вам это не понадобится)

KISS – Keep it simple, stupid (будь попроще, глупый)

PPS – Pay Per Share (награда за каждую единицу работы)

PPLNS – Pay Per Last N Share (награда за последние N число единиц работы)

P2P – Peer-to-peer (равный к равному, одноранговая сеть)

CPU – Central processing unit (центральный процессор)

GPU – Graphics processing unit (графический процессор)

ASIC – Application-specific integrated circuit (интегральная схема специального назначения)

ОС – операционная система

ПС – программное средство

ПО – программное обеспечение

БД – база данных

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;
- распределение награды по невыгодному алгоритму.

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 предметной области;
- недостаточное количество литературы на русском языке;
- высокий уровень ответственности;

- 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. Программное средство обеспечивает распределение прибыли в соответствии с проделанной работой.

Цель работы — формирование архитектуры приложения добычи криптовалют, организация сети между участниками, распределение вычислительной нагрузки, распределение прибыли в соответствии с выполненной работой 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 добытчиками криптовалюты, 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. [1]

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. Данный 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[2]

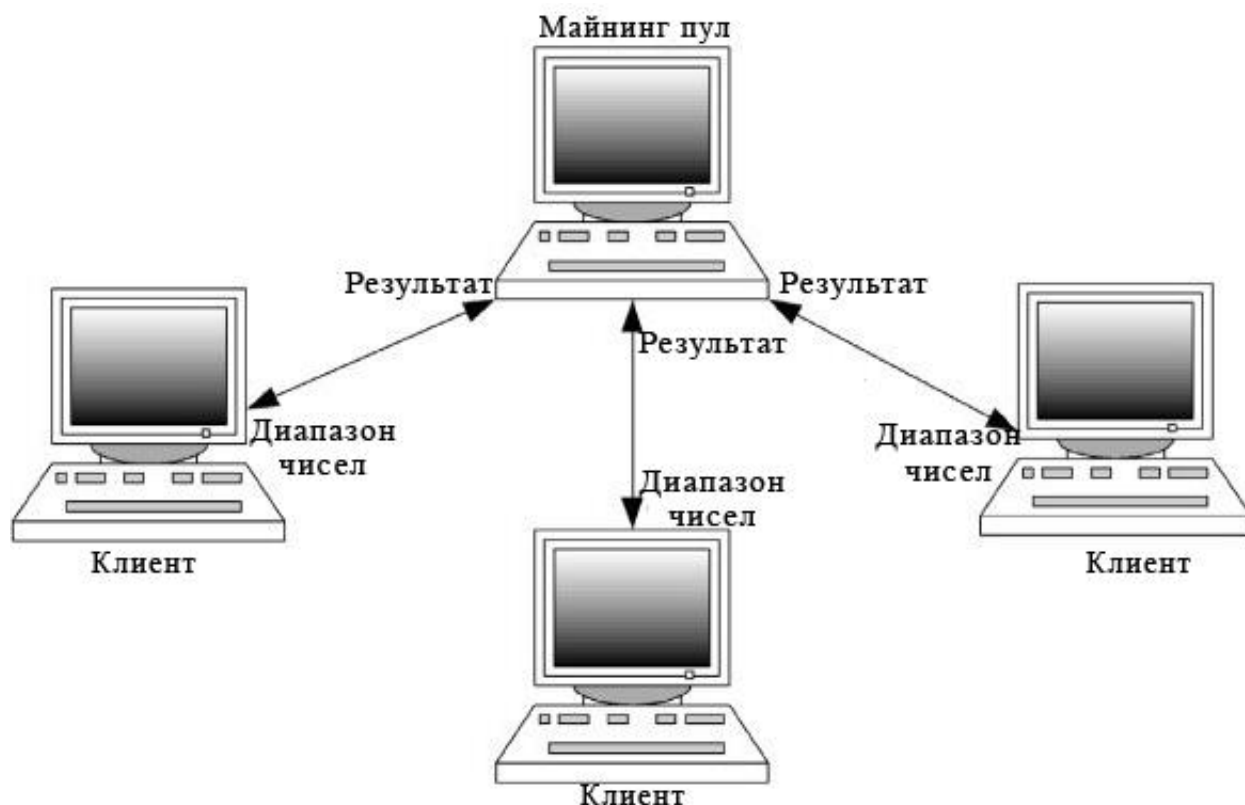


Рисунок 1.1 – Взаимодействие майнинг пула с участниками

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. Добытчик криптовалюты (майнер) следит за транзакциями и операциями, создает блоки, использует мощность персонального компьютера для выполнения специальных вычислений по поиску цифровой подписи (хеша), которая закроет 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[3].

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 [4]:

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 Майнинг ферма – 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[5].

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.

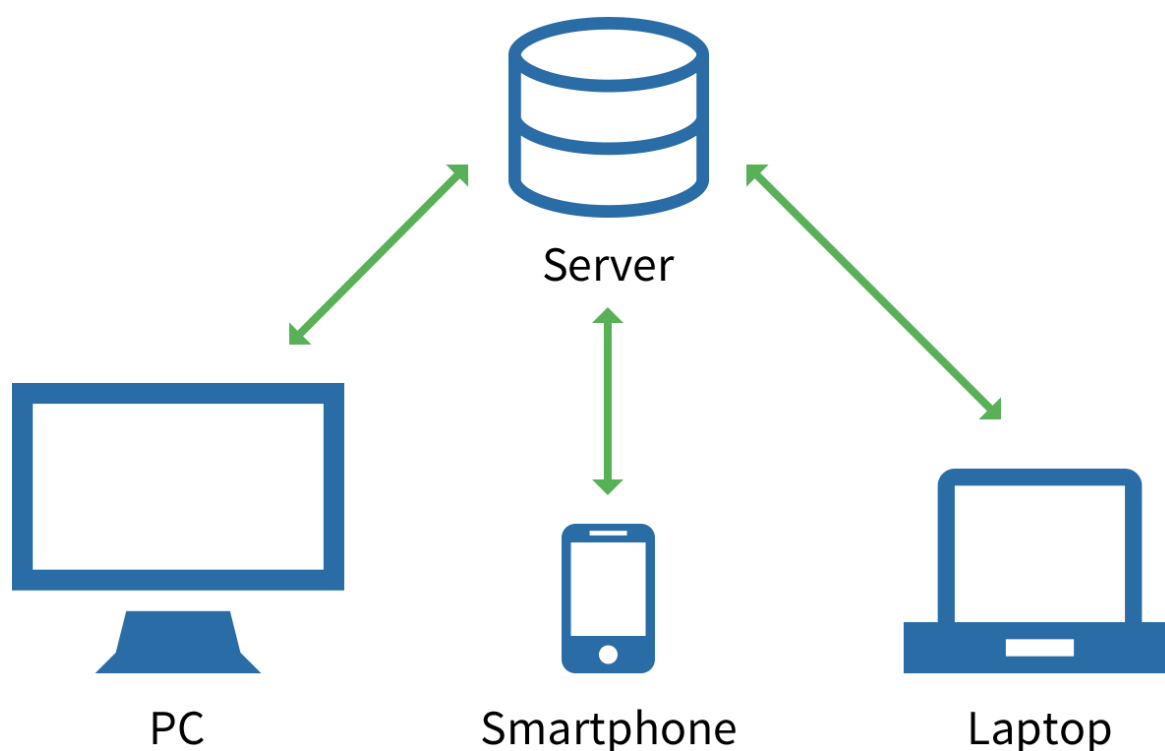


Рисунок 1.3 – Взаимодействие типа «клиент-сервер»

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.

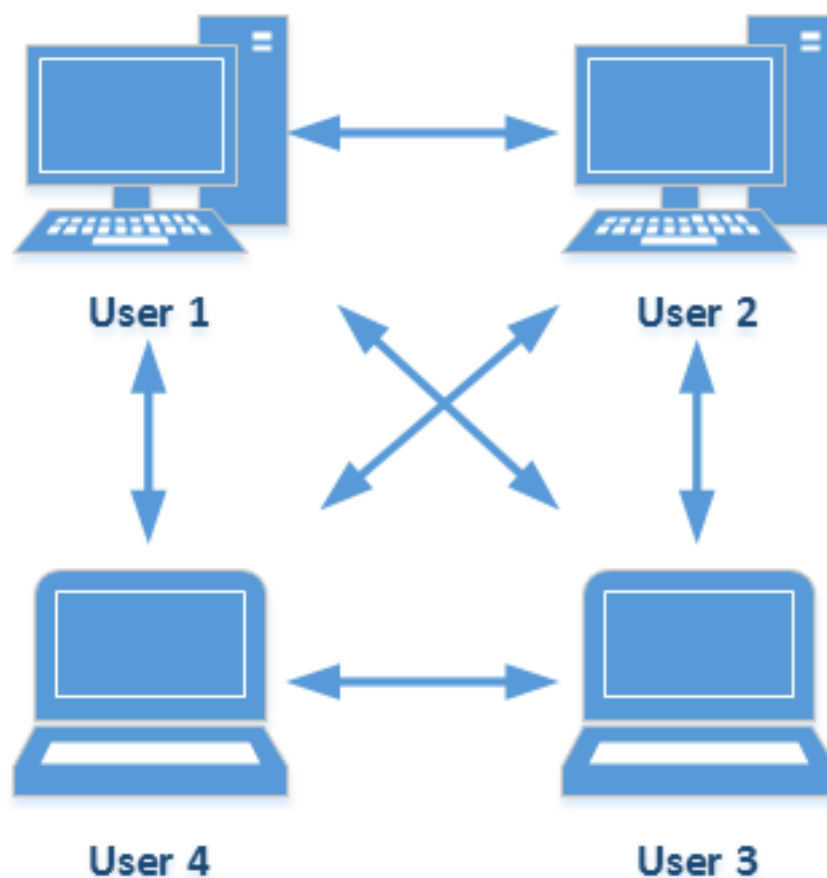


Рисунок 1.4 – Архитектура сети типа «Peer-to-peer»

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[6].

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 проектирование 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 1.5:

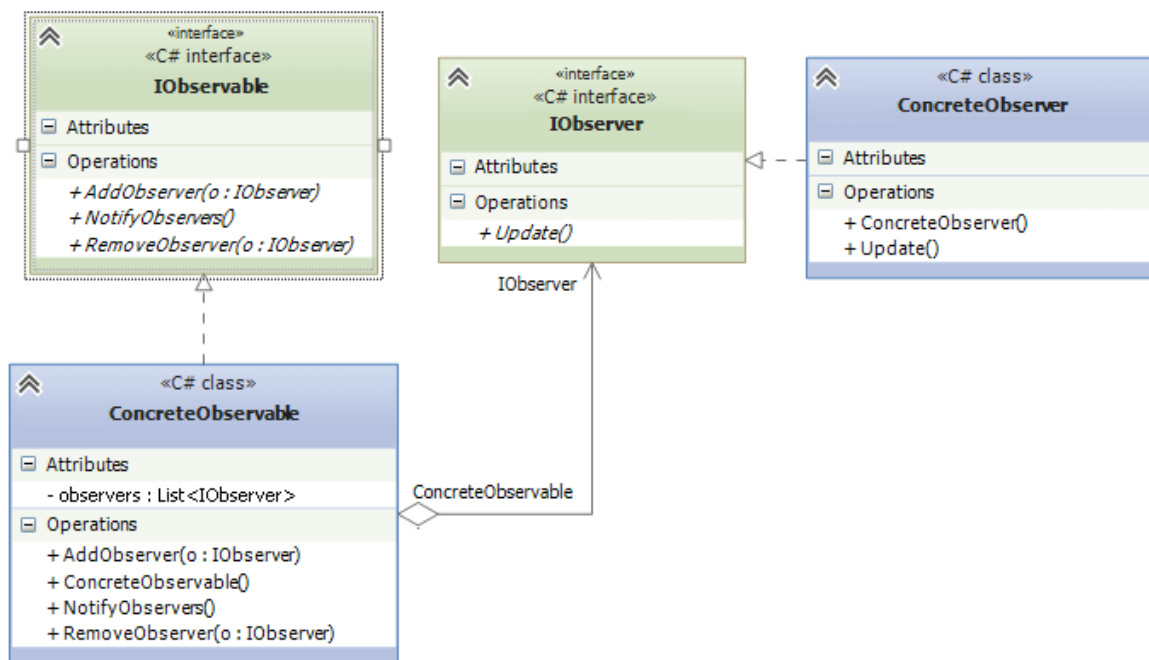


Рисунок 1.5 – UML-диаграмма шаблон проектирования «Наблюдатель»

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 [7].

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, были 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

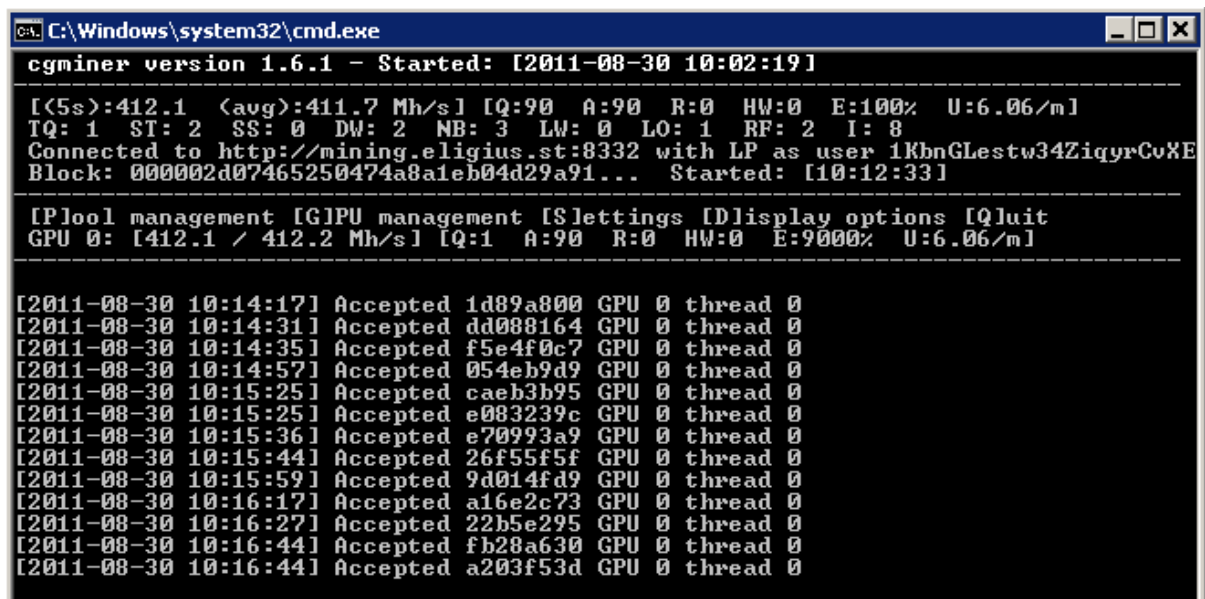
Консольный клиент для майнинга CG MINER (рисунок 1.6). Главное ее преимущество – высокая стабильность и эффективная работа в фоновом режиме. Иными словами, вам не нужно постоянно следить за работой программы – запустили вычисления, которые не требуют от вас дополнительного внимания. Процессы вычисления 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
```

Рисунок 1.6 – 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 1.7.

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.

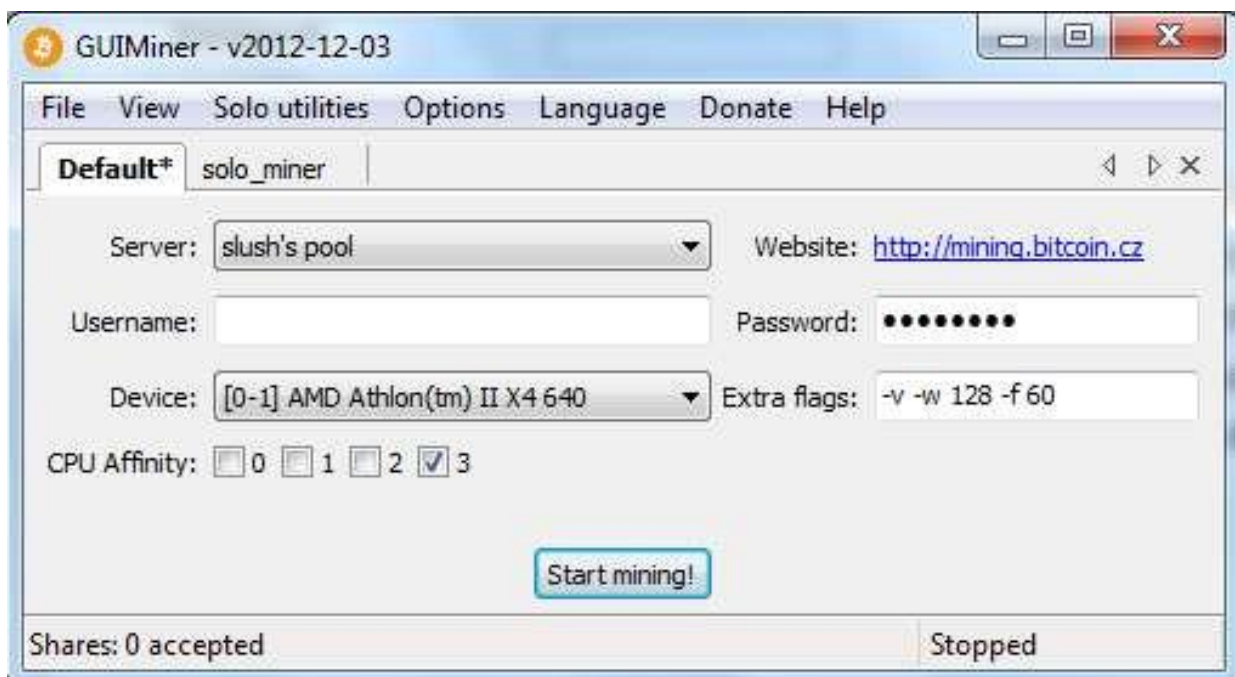


Рисунок 1.7 – Внешний вид GUI Miner

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 1.8.

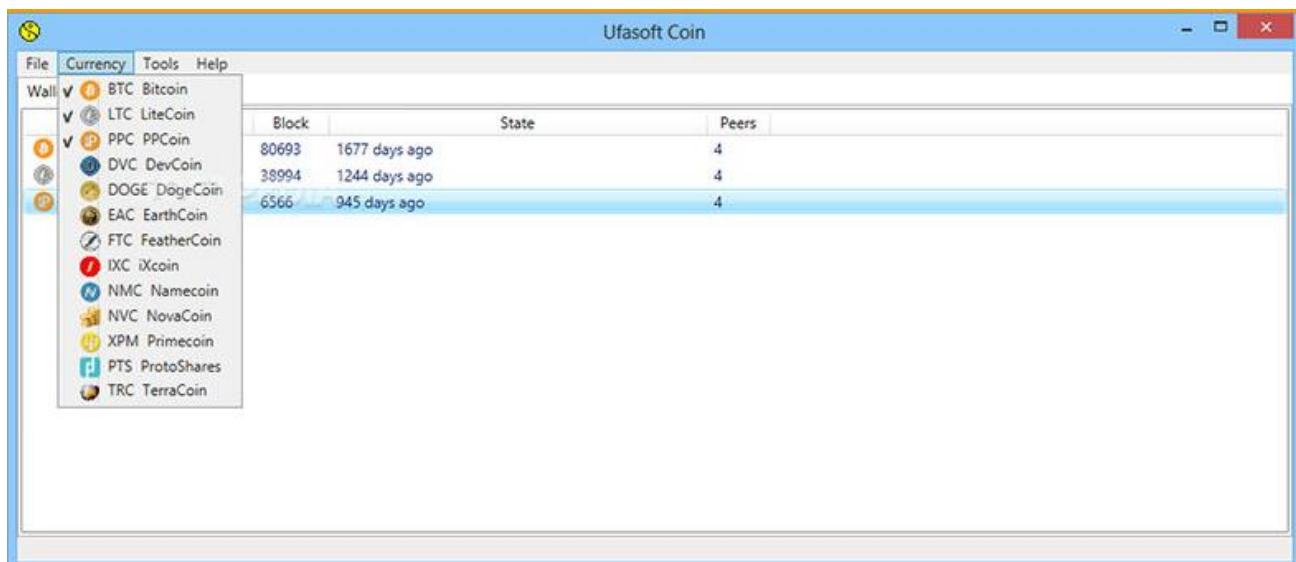


Рисунок 1.8 - Внешний вид Ufasoft Miner

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 1.9.

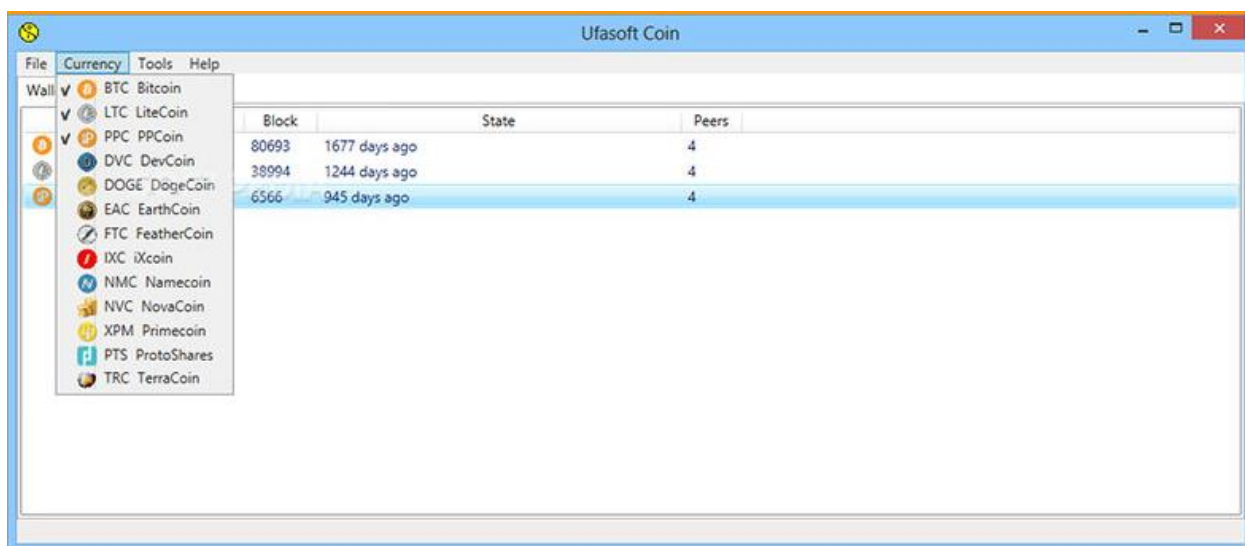


Рисунок 1.9 - Внешний вид Ufasoft Miner

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 1.10.

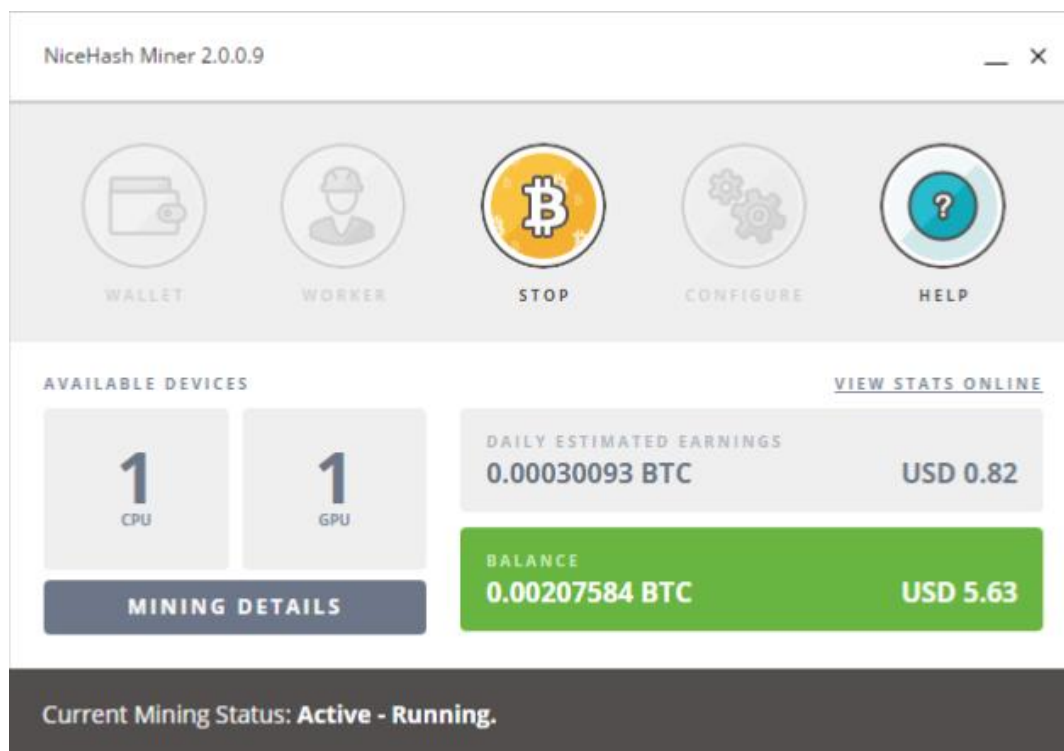


Рисунок 1.10 - Внешний вид Nice Hash Miner

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 прецедентов 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.

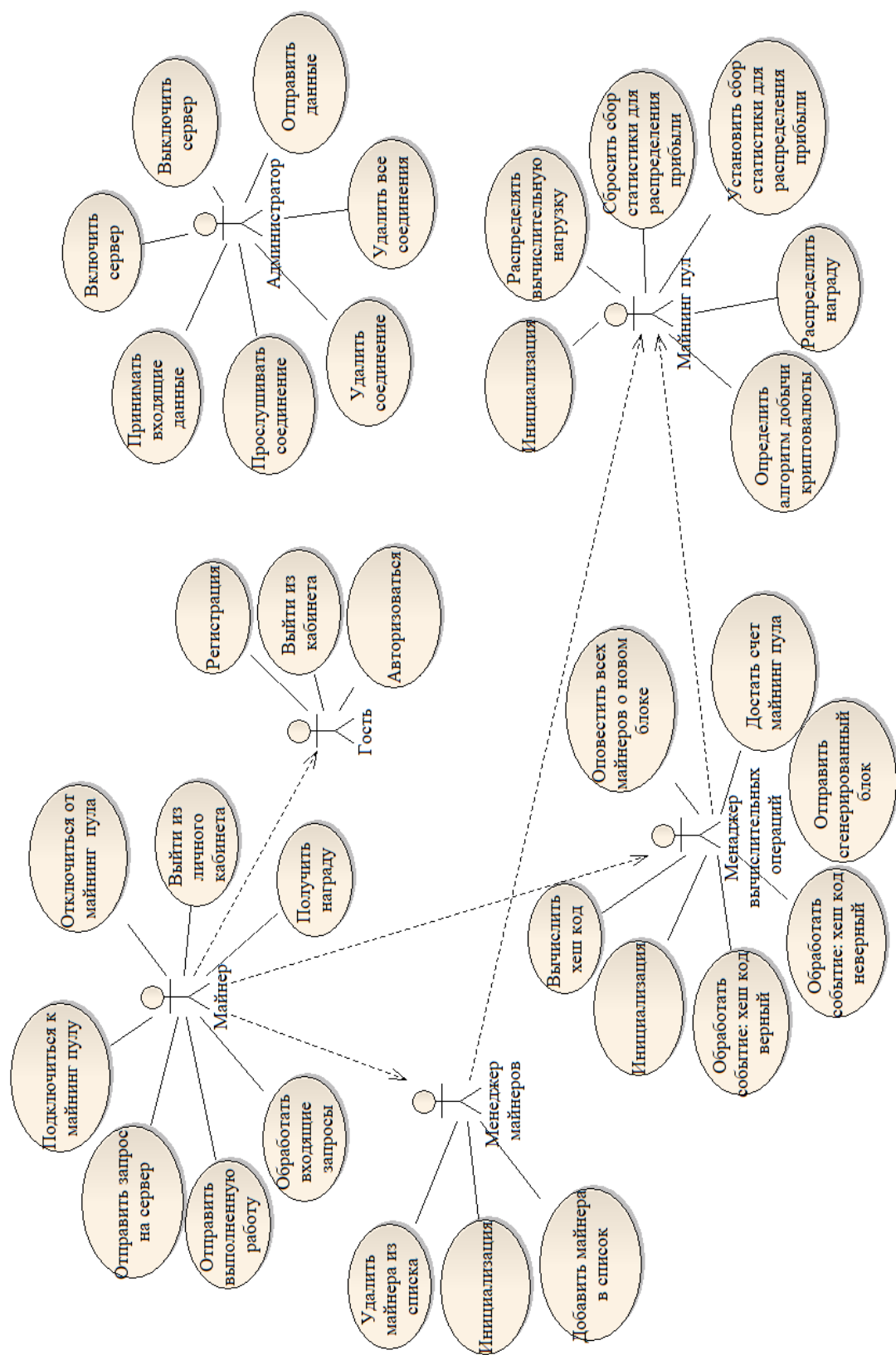


Рисунок 2.1 – Диаграмма прецедентов

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 первостепенными, 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. Под-

держка 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[8].

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 разрабо-

таны 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[9].

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

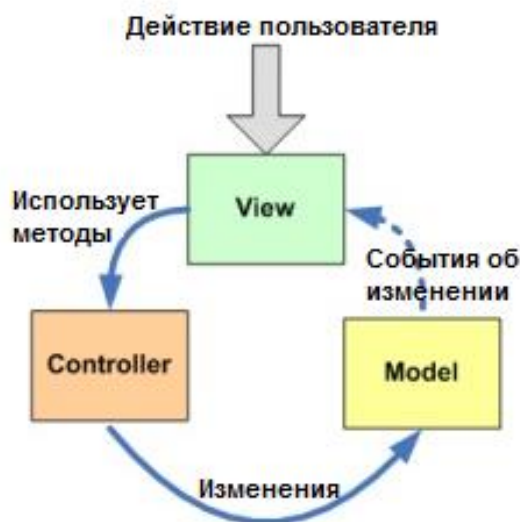


Рисунок 2.2 – Взаимодействие между компонентами

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 C# и платформа .Net Framework

Язык программирования C# 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 C# 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 C# 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 мне-

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

С# 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 объекты 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[10].

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[11].

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. Пример использования библиотеки Nancy представлен в приложении А.

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 И РАЗРАБОТКА 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.

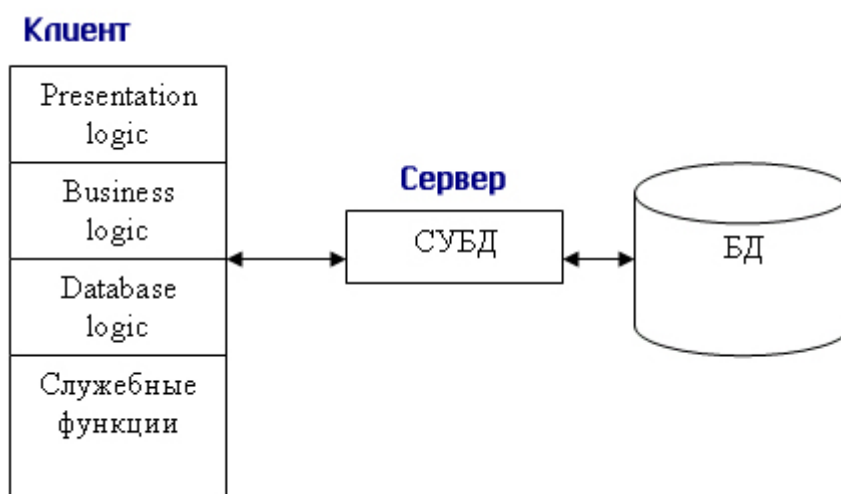


Рисунок 3.1 – Архитектура программного средства

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 3.2.

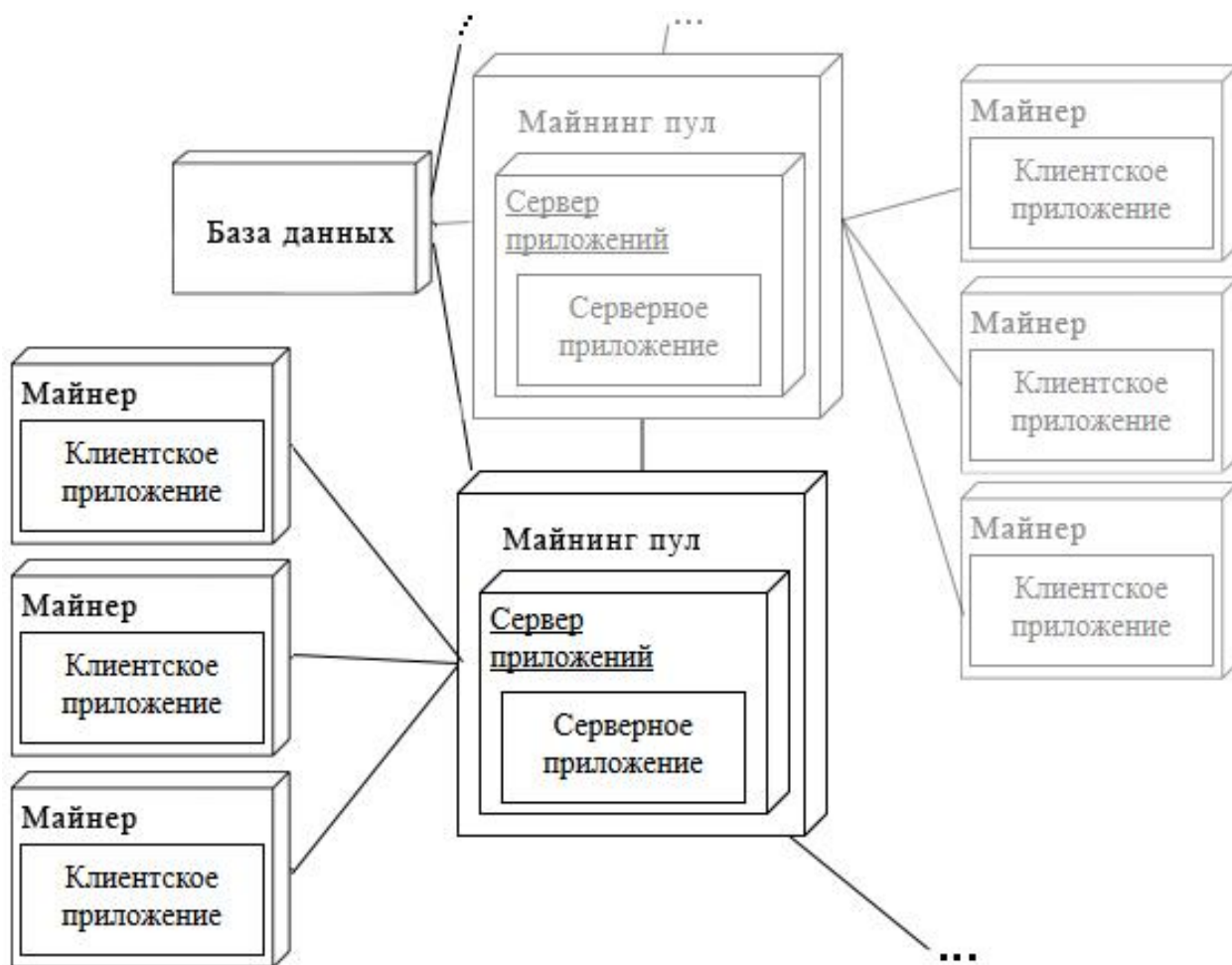


Рисунок 3.2 – Диаграмма развёртывания

3.2 Программный модуль «Организация сети»

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[12].

Для обмена данными между участниками должно быть создано соединение. Таким образом, в основе межсетевых взаимодействий по протоколам

TCP и UDP лежат сокет. В .NET сокет представлен 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` [13].

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` представлен метод `RemoveConnection`, который разрывает установленное соединение, схема алгоритма метода представлена на рисунке 3.3. Также в данном модуле, помимо обработки исключительных ситуаций и метод, предоставлены методы:

- `Listen` – начать прослушивание соединения;
- `Send` - отправки сообщения;
- `Shutdown` - выключить сервер;
- `DisconnectAll` - разорвать все соединения;
- `Start` - включить сервер;
- `AcceptCallback` - принять обратные вызовы от получателя.
- `Dispose` - метод освобождения занимаемых ресурсов.

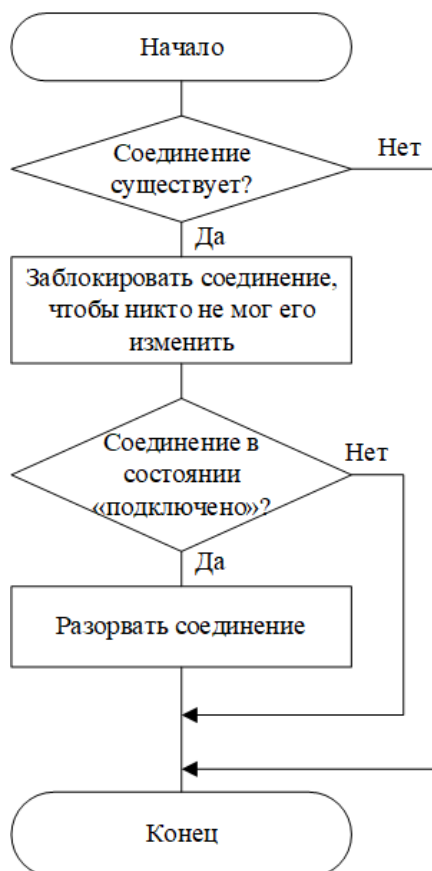


Рисунок 3.3 – Схема алгоритма метода разорвать соединение

Все взаимодействие между «Майнинг пулом» и «Майнером» осуществляется с помощью протокола Stratum, реализация которого, как раз основана на работе с сокетами. В приложении А приведен пример реализации программного модуля организации сети.

Сервер, работающий по протоколу TCP, с помощью сокетов. Общая схема работы серверного сокета TCP представлена на рисунке 3.3 [14].



Рисунок 3.3 - Общая схема работы серверного сокета TCP

Общая схема работы клиента на TCP сокетах будет на рисунке 3.4.



Рисунок 3.4 - Общая схема работы клиента на TCP сокетах

3.3 Программный модуль «Протокол Stratum»

Протокол Stratum представляет собой протокол, который использует простой TCP-сокеты, с полезной нагрузкой, закодированной через JSON-сообщения. Клиент просто открывает TCP-сокеты и пишет запросы на сервер в виде JSON-сообщений, завершенных символом новой строки. Каждая полученная клиентом строка снова является действительным фрагментом JSON, содержащим ответ [15].

Плюсы данного протокола: очень легко реализовать и отлаживать, потому что участники сети посылают сообщения в формате легко читаемом человеком. Протокол отличается от многих других решений. Он является 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 заблокировано ли TCP соединение и конструктор, который проинициализирует поля данного класса.

Программный модуль `StratumMiner.cs`, который выполняет функционал для взаимодействия майнера и сервера:

- `StratumMiner` – создать инстанса для майнеров;
- `Authenticate` – аутентификация майнера;
- `Subscribe` – подключить майнера к майнинг пулу;
- `Parse` – принять и прочитать входящие данные;
- `ProcessRequest` – сгенерировать запрос серверу;
- `SendRequest` – отправить запрос серверу;
- `OnClientDisconnect` – событие о отключении клиента;

– OnDataReceived - событие о получении данных.

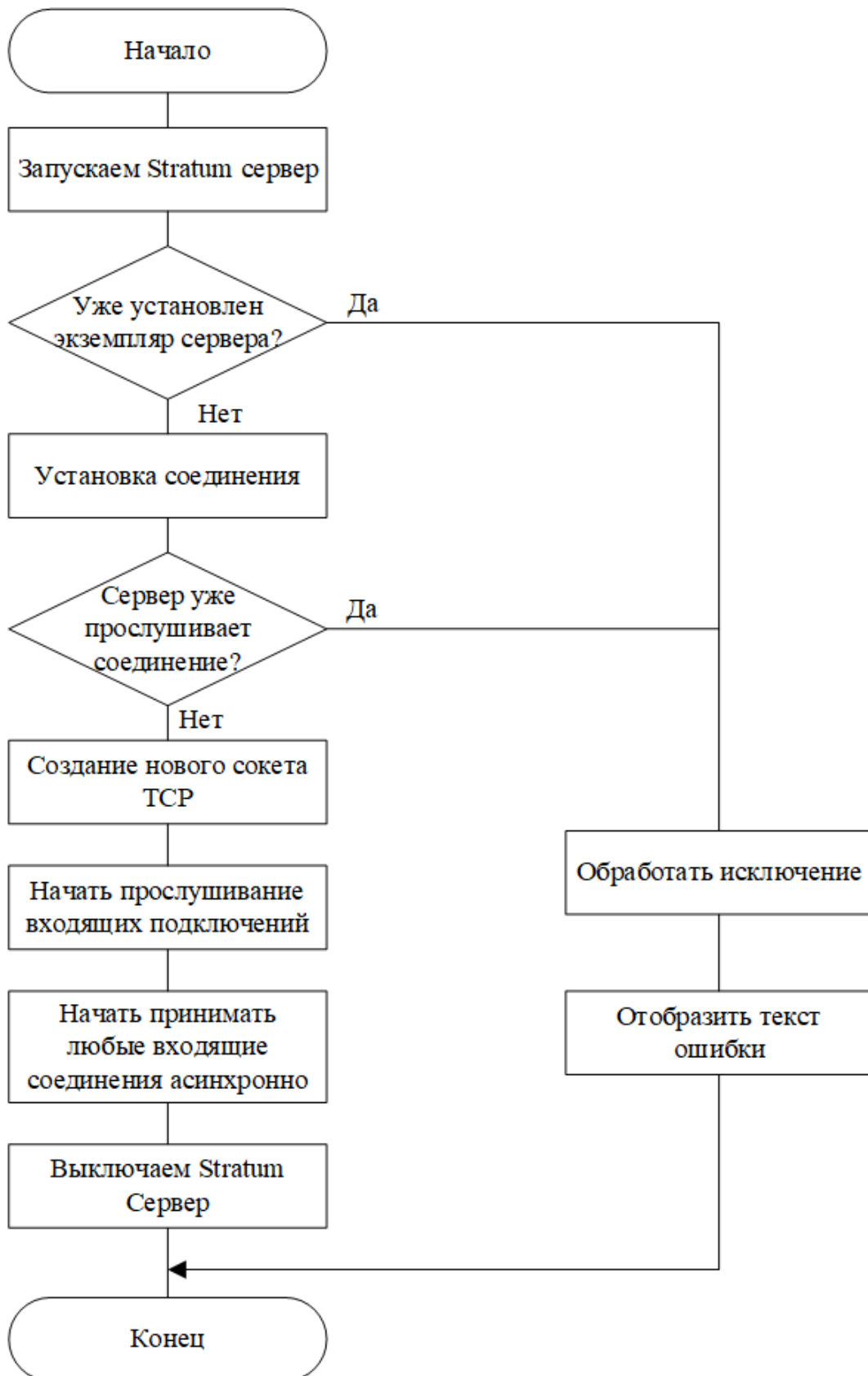


Рисунок 3.2 – Алгоритм работы модуля Stratum Server

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

3.4 Программные модули «Майнер» и «Майнер-менеджер»

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

Пример реализации представлен в приложении А. Методы `Create` и `Remove` вызываются по событиям `OnClientConnection` и `OnClientDisconnection`

соответственно рассмотренного выше модуля StartumServer.cs. Схема алгоритма метода Create представлена на рисунке 3.3



Рисунок 3.3 – Схема алгоритма функции создать майнера

3.5 Программный модуль «Операции с хеш кодом»

Сущность «Share» подразумевает под собой хеш код, который может быть, как валидным, так и нет, но только за валидный хеш код майнеры могут получить прибыль. Сущность «ShareManager» подразумевает вод собой объект, который владеет этими вычислительными операциями для расчета хеш

кода. Стоит отметить, что если майнер добывает криптовалюту самостоятельно, то ему не надо иметь дело с распределением прибыли, 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 работ, чтобы узнать его вклад в создание нового блока. Таким образом зная количество выполненной работы, можно поделить награду в соответствии с работой майнера[16].

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

- ProcessShare служит для получения Share, схема алгоритма работы данной функции представлен на рисунке 3.4;
- 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.

Сущность «Share», которая представляет под собой хеш код имеет следующие атрибуты представленные в таблице 3.2.

Таблица 3.2 – Атрибуты сущности «Share»

Атрибут	Описание
Id	Идентификатор хеш кода
IsValid	Флаг, который определяет валидный ли хеш код
IsBlockCandidate	Флаг, который определяет содержит ли хеш блок
Block	Ссылка на блок
GenerationTransaction	Ссылка на генерируемые транзакции, в случае если хеш верный
IsBlockAccepted	Флаг, который определяет блок принят
Miner	Ссылка на майнерв
Error	Ссылка на ошибку, в ходе вычисления хеша
Job	Ссылка на работу, которая высчитывает хеш

Пример реализации данного программного модуля представлен в приложении А.

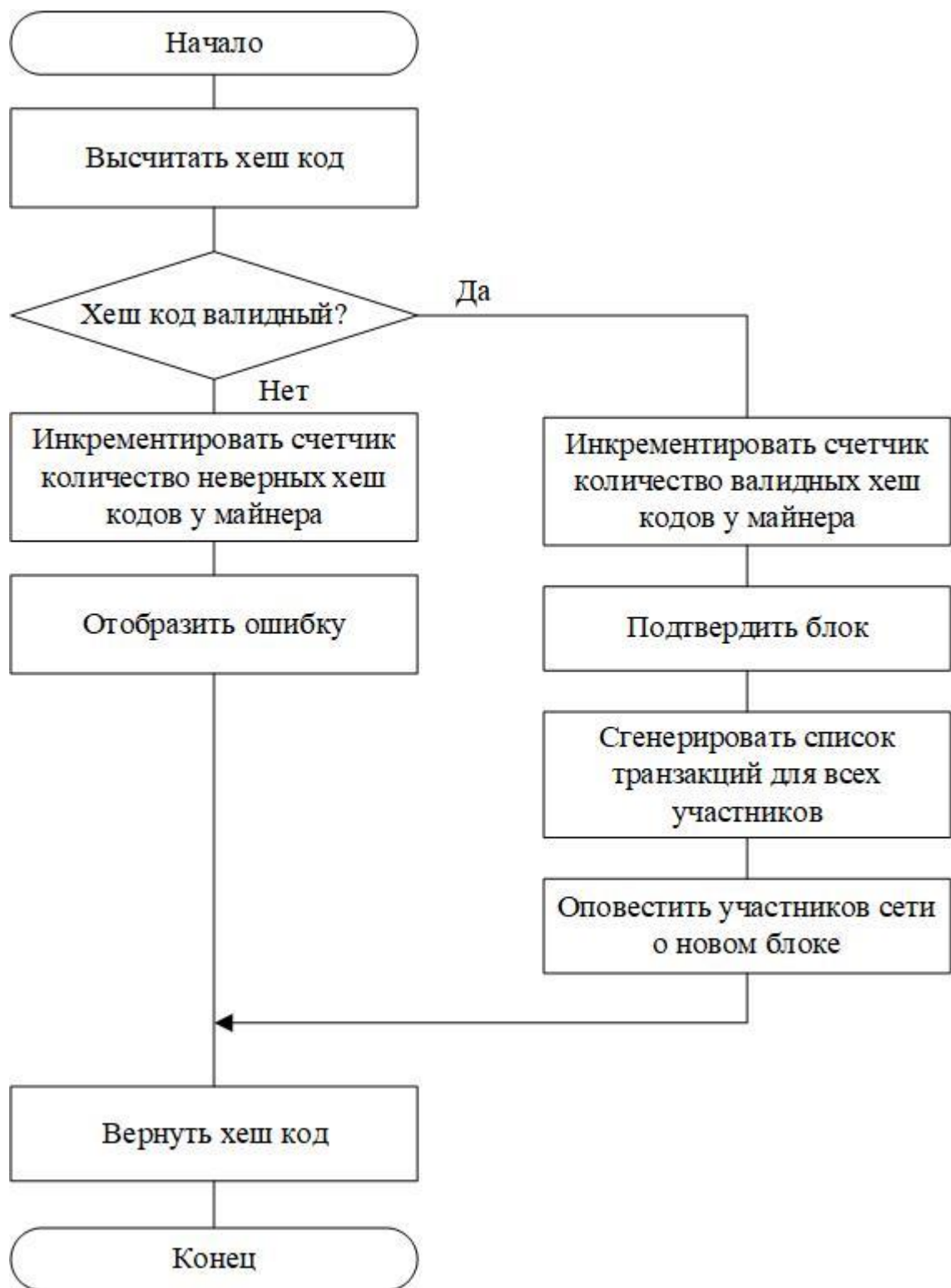


Рисунок 3.4 – Алгоритм процесса получения Share

3.6 Программный модуль «Распределение прибыли»

За каждый добытый блок следует награда. В модуле GenerateTransaction.cs выполняется распределение прибыли между участниками:

- майнеры;
- майнинг пул.

Алгоритм распределения прибыли представлен на рисунке 3.5.

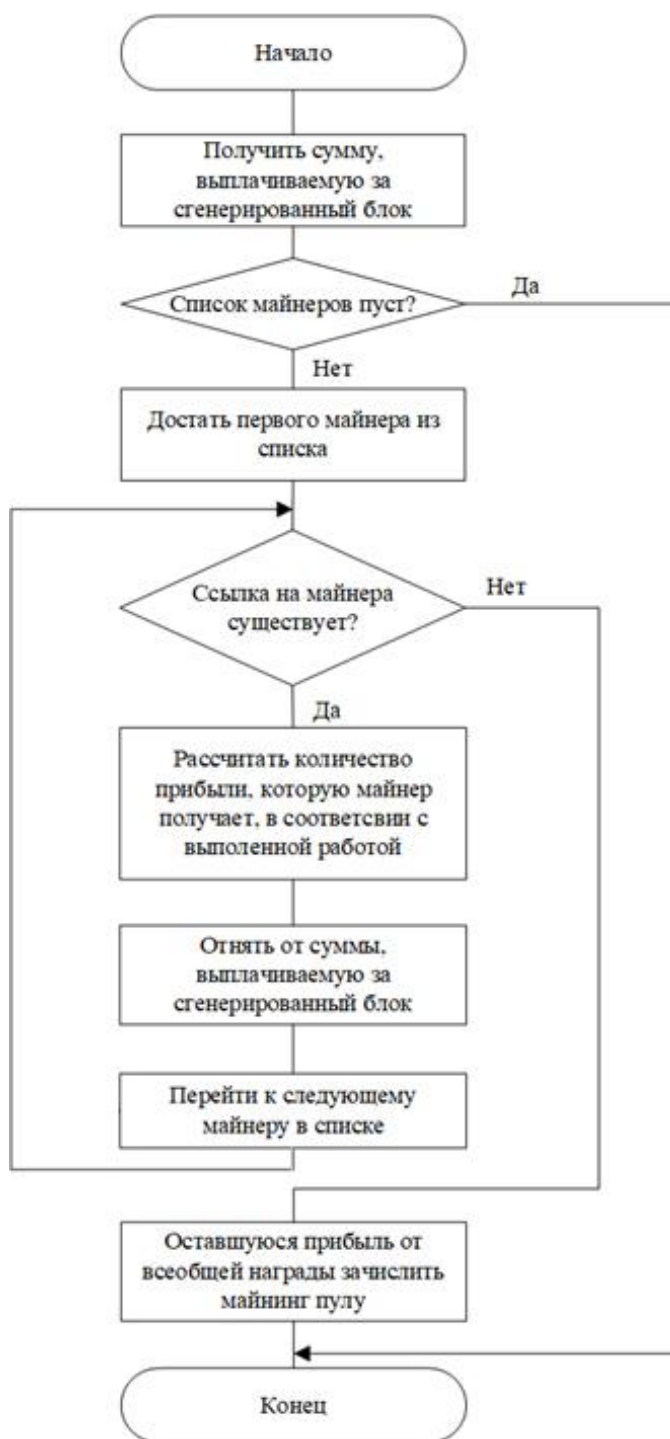


Рисунок 3.5 – Алгоритм распределения прибыли

Прибыль каждому майнеру рассчитывается в соответствии с его выполненной работой. За каждый верный хеш код награда распределяется между всеми участниками. Награда у каждой криптовалюты разная, чтобы узнать ту

самую награду надо послать запрос к серверу криптовалюты. Пример реализации данного программного модуля представлен в приложении А. Схема алгоритма вычисления прибыли представлен на рисунке 3.6 [17].

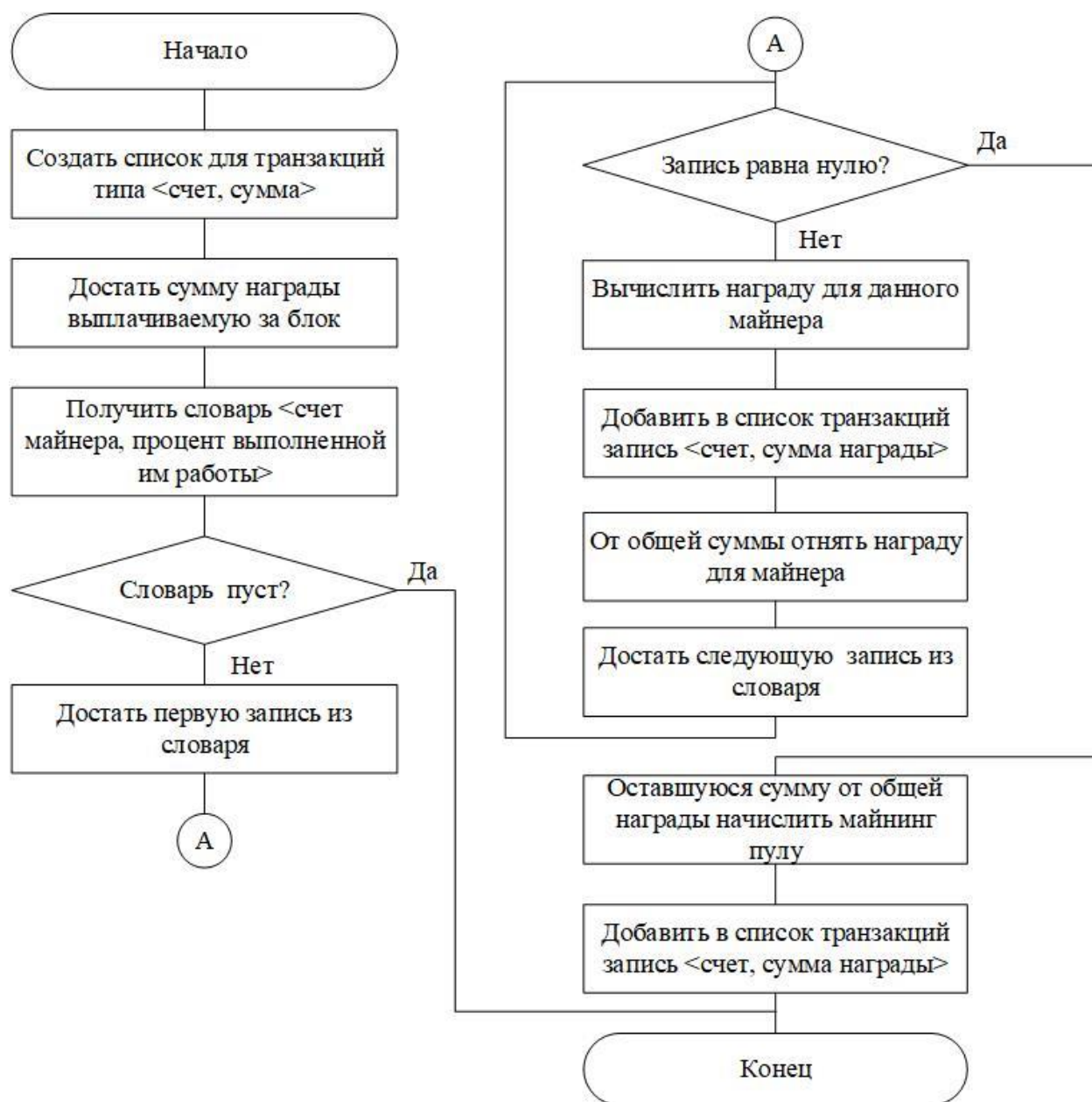


Рисунок 3.6 – Схема алгоритма вычисления прибыли

3.7 Программный модуль «Распределение вычислительной нагрузки между участниками»

Сама сущность Range подразумевает диапазон чисел, которые используют майнеры для нахождения хеш кода. 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 в диапазоне. Пример реализации данного программного модуля представлен в приложении А[18].

Чтобы майнинг пулу распределить нагрузку между участниками, используется класс Range.cs. Данный класс содержит:

- start - свойство, которое указывает начальное значение диапазона;
- stop - свойство, которое указывает конечное значение диапазона;
- step - свойство, которое определяет шаг в диапазоне;
- GetEnumerator - метод, который реализует проход по диапазону, схема алгоритма представлена на рисунке 3.7.

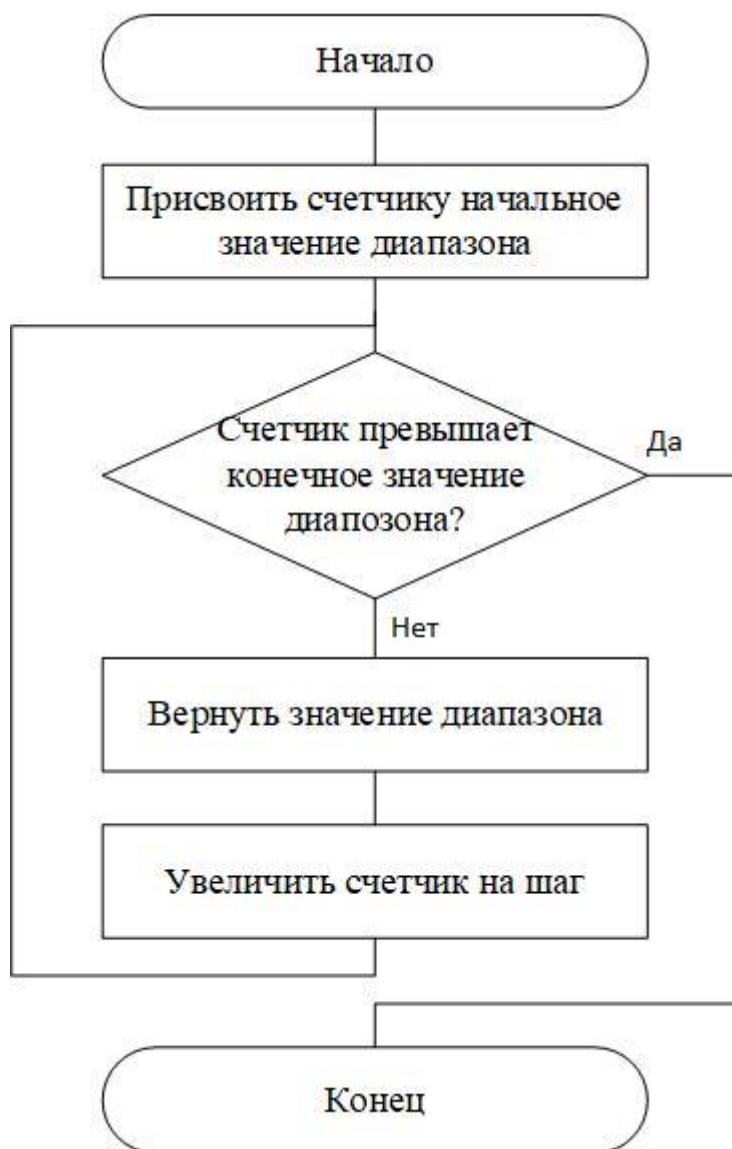


Рисунок 3.7 – Схема алгоритма прохода по диапазону

3.8 Разработка программного модуля «Майнинг пул»

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

Таблица 3.2 – Атрибуты сущности майнинг пула

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

Класс Pool.cs реализует следующие функции:

- InitHashAlgorithm – инициализировать алгоритм(-ы) для работы с майнинг пулом;
- InitStatisticsServices - инициализировать сервисы статистики для работы с майнинг пулом;
- InitCoreServices - инициализировать сервисы ядра для работы с майнинг пулом;
- InitStorage - инициализировать хранилище для работы с майнинг пулом;
- InitNetworkServers – инициализировать сеть для работы с пулом;
- Recache – сбросить значения статистики;
- CalculateHashrate – количество хешей, которое майнер (майнинг пул) находит в секунду.

Среднее время нахождения *time* блока в майнинг пулом рассчитывается по формуле:

$$time = \frac{difficulty \cdot 2^{32}}{hashrate}, \quad (3.1)$$

где *difficulty* – это текущая сложность сети, 32 первых бита хеша, которого являются 0, остальную часть хеша составляют единицы, *byte*; *hashrate* – количество хешей, которое майнер находит в секунду[19].

Пример реализации данного программного модуля представлен в приложении А.

Схема алгоритма инициализации майнинг пула, представлена на рисунке 3.9.

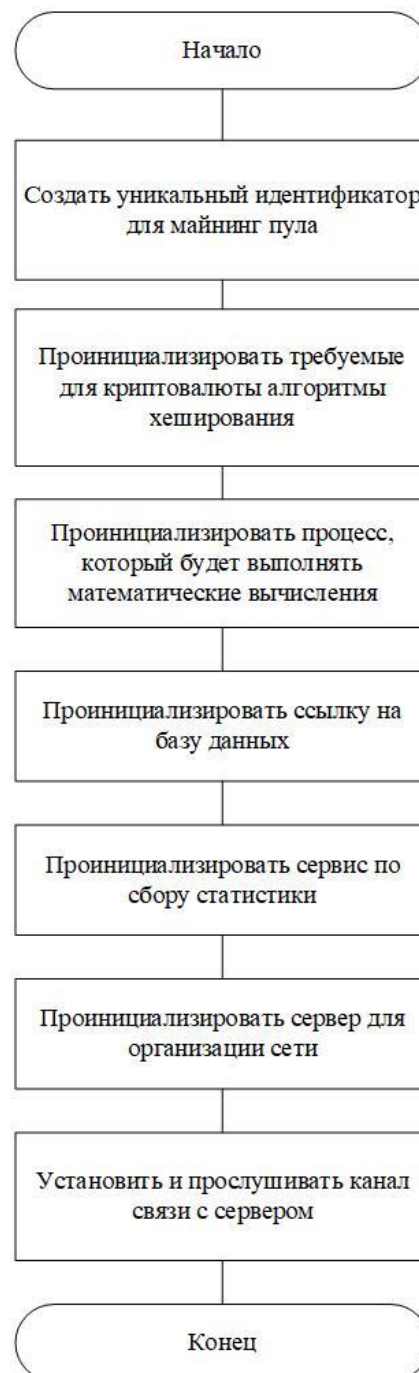


Рисунок 3.9 – Схема алгоритма инициализации майнинг пула

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 процесс, который обеспечивает контроль качества программного продукта. Это процесс, который длится на протяжении всего жизненного цикла ПО [20].

Виды тестирования представлены на рисунке 4.1.



Рисунок 4.1 – Виды тестиирования

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы. Юнит-тестирование помогает быстрым и качественным способом обнаружить ошибки.

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.

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 запрос	Ожидаемый результат	Полученный результат
Создать майнинг пул со всеми валидными параметрами	Создание майнинг пула	Создание майнинг пула
Создать майнинг пул заданными параметрами	Обработка ошибки	Обработка ошибки
Запрос майнера на подключения к майнинг пулу	Майнер подключился	Майнер подключился
Запрос майнера на подключения к майнинг пулу с необязательным параметром «signature»	Майнер подключился	Майнер подключился
Генерирование Share, которая будет являться кандидатом blockchain цепи.	В объекте Share установлен флаг IsBlockCandidate в true	В объекте Share установлен флаг IsBlockCandidate в true
Генерирование Share, которая будет являться кандидатом blockchain цепи.	В объекте Share установлен флаг IsBlockCandidate в true	В объекте Share установлен флаг IsBlockCandidate в false
Распределение прибыли между майнерами	Прибыль распределена корректна	Прибыль распределена корректна
Распределение прибыли майнинг пулу	Прибыль распределена корректна	Прибыль распределена корректна

Продолжение таблицы 4.1

Тестируемый запрос	Ожидаемый результат	Полученный результат
Запуск сервера с не прописанными параметрами в конфигурационном файле	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	Майнинг пул подключен к сети	Майнинг пул подключен к сети
Подключение майнинг пула, при условии, что StratumServer не включен или недоступен	Майнинг пул не подключен к сети. Вывод сообщения: "No connected servers to network!"	Майнинг пул не подключен к сети. Вывод сообщения: "No connected servers to network!"
Запрос на аутентификацию незарегистрированного майнера	Обработка исключения. Вывод сообщения: "Miner authentication failed"	Обработка исключения. Вывод сообщения: "Miner authentication failed"
Майнер отправляет запрос удалить соединение с сервером	Соединение разорвано	Соединение разорвано
Майнер отправляет запрос удалить соединение с сервером, но соединение установлено не было до этого	Обработка ошибки и вывод сообщения: «Exception connection does not exist.»	Обработка ошибки и вывод сообщения: «Exception connection does not exist.»
Создание нового TCP сокета	Сокет создан	Сокет создан

Продолжение таблицы 4.1

Тестируемый запрос	Ожидаемый результат	Полученный результат
Создание нового TCP сокета с недоступным IP-адресом	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."
Начать прослушивать порт хоста	Сервер слушает канал	Сервер слушает канал
Вызвать функцию, которая прослушивает порт хоста, когда сервер уже прослушивает канал	Обработка исключения, вывод текста ошибки: "Server is already listening."	Обработка исключения, вывод текста ошибки: "Server is already listening."
Вызвать функцию, которая обеспечивает, чтобы сокет принимал все входящие соединения	Сокет принимает все входящие соединения	Сокет принимает все входящие соединения
Вызвать функцию без обязательного параметра: ссылка на соединение, которая обеспечивает, чтобы сокет принимал все входящие соединения	Обработка исключения	Обработка исключения
Вызвать функцию, которая обеспечивает, чтобы сокет принимал все входящие соединения с клиентом, которое заблокировано	С клиентом соединение не установлено, показать сообщение ошибки. Сокет принимать другие входящие вызовы не прекратил	С клиентом соединение не установлено, показать сообщение ошибки. Сокет принимать другие входящие вызовы не прекратил
Запуск сервера	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"

Продолжение таблицы 4.1

Тестируемый запрос	Ожидаемый результат	Полученный результат
Авторизация существующего майнера на основании его имени и пароля	Авторизация прошла успешно	Авторизация прошла успешно
Авторизация несуществующего майнера на основании его имени и пароля	Обработка ошибки. Вывод текста сообщения: "Unauthorized worker: {username}"	Обработка ошибки. Вывод текста сообщения: "Unauthorized worker: {username}"
Майнер сгенерировал хеш код, который уже существует	Обработка ошибки и вывод текста: "Duplicate share".	Обработка ошибки и вывод текста: "Duplicate share".
Сгенерированный хеш на содержит ссылку на работу, которая выполнила хеш код	Обработка ошибки и вывод текста: "Job not found"	Обработка ошибки и вывод текста: "Job not found"
Хеш код был сгенерирован с низкой сложностью	Обработка исключения и вывод текста ошибка: "Low difficulty share"	Обработка исключения и вывод текста ошибка: "Low difficulty share"
Создать майнинг пул	Майнинг пул создан	Майнинг пул создан
Запустить все майнинг пулы	Все майнинг пулы запущены	Все майнинг пулы запущены
Установить начальное значение диапазона чисел	Начальное значение установлено	Начальное значение установлено
Установить конечное значение диапазона чисел	Конечное значение установлено	Конечное значение установлено
Установить конечное значение диапазона чисел, которое меньше начального	Конечное значение установлено. Установить флаг, который показывает прибавлять или отнимать шаг	Конечное значение установлено. Установить флаг, который показывает прибавлять или отнимать шаг
Установить шаг диапазона чисел	Шаг установлен	Шаг установлен
Получить каждое число из диапазона чисел от начала до конца через заданный шаг	Каждое число получено	Каждое число получено

Продолжение таблицы 4.1

Тестируемый запрос	Ожидаемый результат	Полученный результат
Получить каждое число из диапазона чисел от начала до конца через заданный шаг, если конечном числе меньше, чем начальное	Каждое число получено	Каждое число получено
Вызвать функцию, которая высчитывает хеш код	Хеш код получен	Хеш код получен
Вызвать событие, которое обрабатывает, что хеш код неверный	Инкрементировать счетчик. Показать причину, почему данный хеш код неверный	Инкрементировать счетчик. Показать причину, почему данный хеш код неверный
Вызвать событие, которое обрабатывает, что хеш код верный	Инкрементировать счетчик. Отправить блок.	Инкрементировать счетчик. Отправить блок.
Вызвать событие, которое обработает событие: блок найден	В сеть отправлено сообщение о том, что новый блок найден	В сеть отправлено сообщение о том, что новый блок найден

5 МЕТОДИКА ИСПОЛЬЗОВАНИЯ РАЗРАБОТАННОГО ПРОГРАММНОГО СРЕДСТВА

Данное программное средство является кроссплатформенным и может быть развернуто на следующих операционных системах:

- Linux/Unix;
- Windows;
- MacOS.

Также данное приложение поддерживается во всех браузерах.

Программное средство после авторизации/регистрации пользователя имеет переходит на главную страницу веб-приложения, которая представлена на рисунке 5.1. На данном рисунке представлена главная страница со списком подключенных майнинг пулов и вычислительных алгоритмов. Слева можно увидеть навигационное меню. Также в верхней части страницы можно наблюдать блоки, с помощью которых можно узнать общую информацию сервера:

- общую мощность майнинг пула;
- общее число майнеров;
- общее число подключенных пулов;
- общее число подключенных алгоритмов.

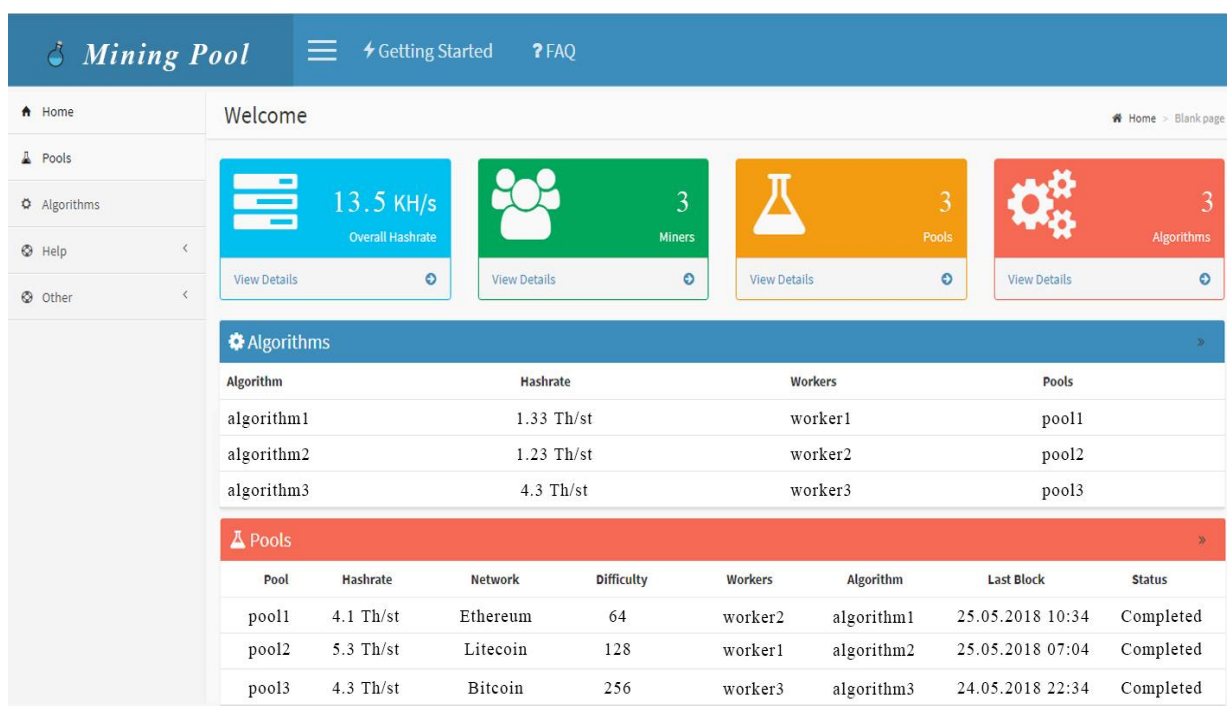


Рисунок 5.1 – Главная страница веб-приложения

На следующем рисунке 5.2 представлена веб-страница «Майнеры». В центре страницы расположена таблица, в которой находятся добытчики криптовалюты и их информация:

- майнинг пул;
- имя майнера;
- статус;
- мощность майнинг пула;
- мощность майнера за 1 час;
- ограничитель мощности майнера;
- минимальная сложность.

В верхней части страницы находятся 2 блока: общая информация и фильтр. С помощью фильтра можно отобразить записи в таблице по нужным значениям атрибутов.

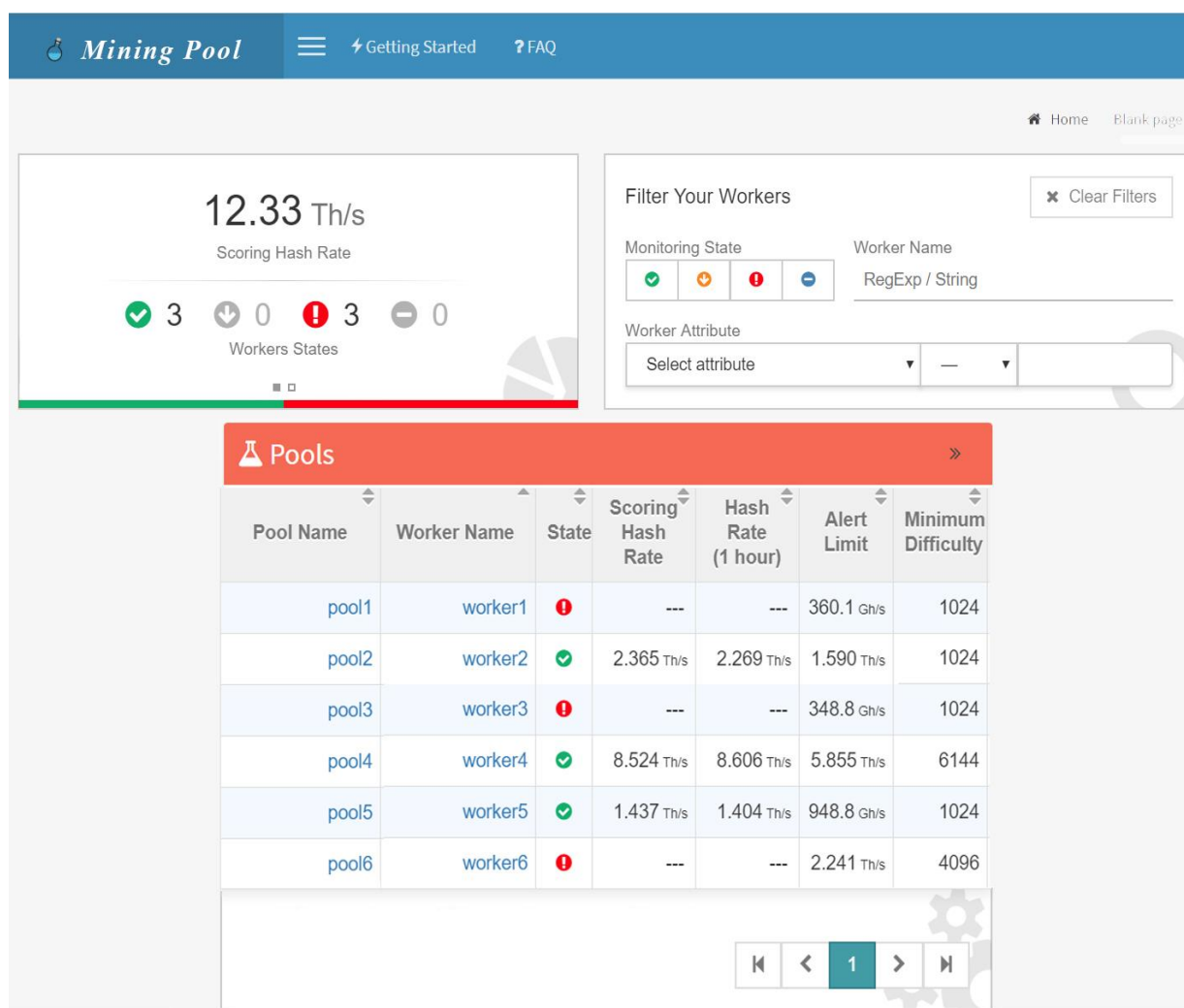


Рисунок 5.2 – Веб-страница «Майнеры»

На рисунке 5.3 представлена форма регистрации клиента. Дальнейшие настройки для подключения электронного кошелька редактируются в личном кабинете.

The image shows a web form titled "Register an account". It contains four input fields: "Username" with a person icon, "Email" with an envelope icon, "Required password" with a padlock icon, and "Password again" with a padlock icon. Below these fields is a checkbox labeled "I agree with terms of service.". At the bottom of the form is a large teal button labeled "Register now". Below the button, there is a link "Already have an account?" followed by a user icon and the text "Log in here".

Рисунок 5.3 – форма регистрации клиента

Веб-страница «История наград» представлена на рисунке 5.4. В центре старанице расположена таблица. Запись в данную таблицу добавляется тогда, когда генерируются новый блок. Запись содержит следующую информацию о блоке и награде за блок:

- идентификатор блока;
- момент времени нахождения блока;
- промежуток времени нахождения блока;
- объем работы майнинг пулом для нахождения данного блока;
- объем работы майнером для нахождения данного блока;
- прибыль майнера;
- награда за сгенерированный блок;
- уровень подтверждения данного блока.

Рядом с некоторыми колонками можно увидеть символ «?», при нажатии на иконку покажется подсказка, что означает данное понятие, что улучшает пользователю понимание системы.

В шапке таблицы находятся кнопки с иконками криптовалют, которые символизируют майнинг пулы, к которым майнер подключился. Таким образом майнер может переключить криптовалюту и посмотреть свою историю наград в других майнинг пулах.

Mining Pool							
Reward History							
Visit User Manual for more information about how your rewards are calculated. User Manual							
Block ID	Block Found At	Duration	Pool Scoring Hash Rate	Your Scoring Hash Rate	Your Reward	Block Value	Confirmations Left
34645	2018-05-26 21:41	00:38:06	3.733 Eh/s	12.77 Th/s	0.00004283 BTC	12.77785820 BTC	97
34644	2018-05-26 21:03	00:13:18	3.711 Eh/s	11.71 Th/s	0.00003875 BTC	12.52713431 BTC	93
34643	2018-05-26 20:50	00:02:41	3.725 Eh/s	12.06 Th/s	0.00004003 BTC	12.61193416 BTC	90
34642	2018-05-26 20:47	00:38:44	3.739 Eh/s	12.36 Th/s	0.00004223 BTC	13.04055461 BTC	89
34641	2018-05-26 20:08	04:27:30	3.771 Eh/s	12.95 Th/s	0.00004279 BTC	12.71154439 BTC	88
34640	2018-05-26 15:41	00:45:51	3.740 Eh/s	11.50 Th/s	0.00003864 BTC	12.81960288 BTC	67
34639	2018-05-26 14:55	00:13:45	3.768 Eh/s	11.62 Th/s	0.00003778 BTC	12.50329485 BTC	59
34638	2018-05-26 14:41	00:25:20	3.763 Eh/s	12.48 Th/s	0.00004143 BTC	12.74912019 BTC	57
34637	2018-05-26 14:16	00:12:36	3.767 Eh/s	12.59 Th/s	0.00004108 BTC	12.53739165 BTC	54
34636	2018-05-26 14:03	02:52:52	3.780 Eh/s	12.87 Th/s	0.00004211 BTC	12.61493395 BTC	51
34635	2018-05-26 11:10	05:16:14	3.889 Eh/s	12.96 Th/s	0.00004148 BTC	12.70382541 BTC	34

Рисунок 5.4 – веб-страница «История наград»

Все валидационные сообщения, обработки ошибок пользователь видит в браузере в виде выскакивающих окон. Это сделано с целью, чтобы пользователь обязательно увидел и подтвердил увиденную ошибку, так как веб-приложение связано с финансами и важно, чтобы ошибка была увиденна. Пример сообщения об ошибке представлен на рисунке 5.5.

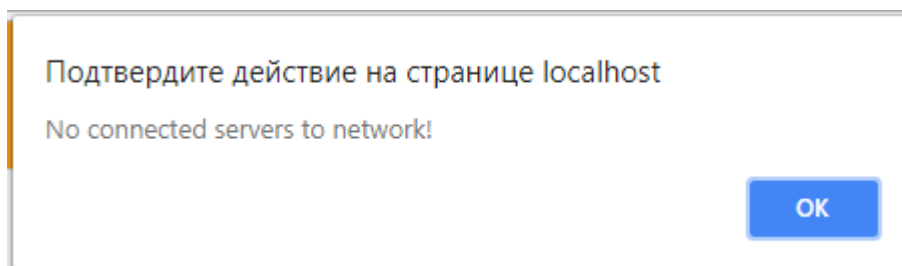


Рисунок 5.5 - Пример сообщения об ошибке

Данная предметная область сложная для понимания, поэтому чтобы пользователю было легче разобраться в системе, была сделана веб-страница помощь, которая представлена на рисунке 5.6. На данной веб-странице описаны основные процессы, понятия, ссылки на полезные видео или литературные источники. Ознакомившись с данной веб-страницей пользователь лучше начнет разбираться в системе, процессах, понятиях и начнет больше доверять веб-приложению.

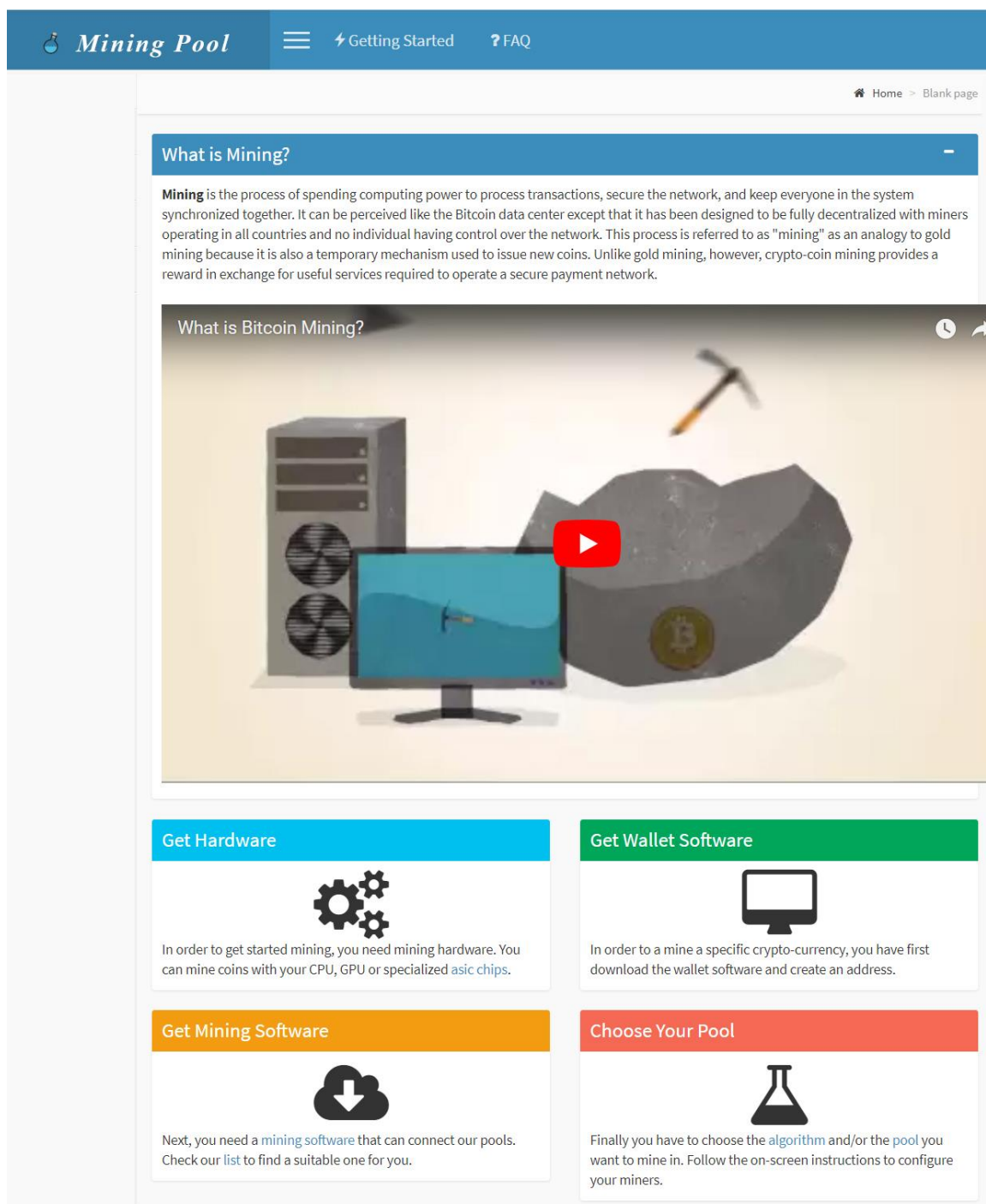


Рисунок 5.6 – веб-страница «Помощь»

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 заработ-

ную 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 ($K_n = 0,7$) [21].

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:

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

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:

$$T_o = 312 \cdot 1,27 \cdot 0,8 \cdot 0,7 \approx 194 \text{ чел./дн.} \quad (6.4)$$

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.;

$\Phi_{\text{эф}}$ – 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:

$$\Phi_{\text{эф}} = D_{\Gamma} - D_{\Pi} - D_{\text{в}} - D_{\text{о}}, \quad (6.6)$$

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

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

$D_{\text{в}}$ – выходных дней в году, дн.;

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

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

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

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:

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

6.3 Расчёт заработной платы исполнителей

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

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

где $З_{ci}$ – 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 $З_{ci}$ применяется формула:

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

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:

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

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:

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

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

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

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

Размер затрат на расходные материалы (3%) составит:

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

Расходы по статье «Машинное время» определяются:

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

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

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

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

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

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 по формуле 6.22:

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

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

$$C_{\text{пi}} = Z_{\text{oi}} + Z_{\text{дi}} + Z_{\text{сzi}} + P_{\text{ci}} + P_{\text{mi}} + M_i + P_{\text{нki}} + П_{\text{zi}} + P_{\text{нi}}, \quad (6.23)$$

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, %	$Y_{\text{рп}}$	40
Норматив НДС, %	$H_{\text{дс}}$	20
Норматив расходов на сопровождение программного средства, %	$H_{\text{с}}$	20
Норматив расходов на освоение программного средства, %	H_{o}	10

Прибыль от программного средства рассчитывается по формуле:

$$П_{\text{oi}} = \frac{C_{\text{пi}} \cdot Y_{\text{рпi}}}{100}, \quad (6.25)$$

где $Y_{\text{рпi}}$ – уровень рентабельности программного средства;

$C_{\text{пi}}$ – 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:

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

Таким образом, прогнозируемая цена без налога равна:

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

Для расчета налога на добавленную стоимость применяется формула:

$$\text{НДС}_i = \frac{\Pi_{ni} \cdot \text{НДС}}{100}, \quad (6.29)$$

где НДС – норматив налога на добавленную стоимость.

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

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

Прогнозируемая отпускная цена на программное средство вычисляется по формуле:

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

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:

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

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

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

$K_{\text{пр}}$ – затраты пользователя на приобретение ПС по отпускной цене у разработчика с учетом стоимости услуг по эксплуатации;

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

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

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

Продолжение таблицы 6.3

Наименование показателей	Усл. обозн.	Ед. изм.	Значение показателя		Наименование источника информации
			в базовом варианте	в новом варианте	
7. Среднемесячная ЗП одного программиста	$З_{см}$	руб.	1730	1730	Расчётные данные пользователя
8. Коэффициент начислений на зарплату	$K_{нз}$		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	$З_{т1}, З_{т2}$	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	$T_ч$	ч.	8		Принято для расчета
15. Ставка налога на прибыль	$H_{п}$	%		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:

$$C_3 = C_{зе} \cdot A_2, \quad (6.39)$$

где $C_{зе}$ – 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 ($C_{зе}$) рассчитывается по формуле:

$$C_{зе} = \frac{Z_{см} \cdot (T_{с1} - T_{с2})}{D_p \cdot T_ч}, \quad (6.40)$$

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

$T_{с1}, T_{с2}$ – снижение трудоемкости;

$T_ч$ – количество часов работы в день;

D_p – среднемесячное количество рабочих дней.

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

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

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

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

Экономия с учётом начисления на зарплату ($C_н$) будет равна:

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

Экономия за счёт сокращения простоев сервиса ($C_с$) будет равна:

$$C_с = \frac{(П1 - П2) \cdot D_p \cdot C_p}{60}, \quad (6.44)$$

$$C_с = \frac{40 \cdot 21,5 \cdot 30,1}{60} = 431,43 \text{ руб.} \quad (6.45)$$

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

$$C_o = C_n + C_c, \quad (6.46)$$

И составит 32120,43 руб. по формуле:

$$C_o = 31689 + 431,43 = 32120,43 \text{ руб.} \quad (6.47)$$

Чистая прибыль от экономии текущих затрат высчитывается по формуле:

$$\Delta\Pi_q = C_o - \frac{C_o \cdot H_n}{100}, \quad (6.48)$$

где H_n – ставка налога на прибыль.

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

$$\Delta\Pi_q = 32120,43 - \frac{32120,43 \cdot 18}{100} = 26338,75 \text{ руб.} \quad (6.49)$$

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 (Π_q)	руб.	26338,75	26338,75

Продолжение таблицы 6.4

Показатели	Ед. изм.	2018	2019
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 ($\Delta\Pi_{\text{ч}} = 26338,75$ 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 разработки.

ЗАКЛЮЧЕНИЕ

В ходе работы над дипломным проектом была проделана работа, в ходе которой были изучены такие понятия как: «майнинг», «майнер», «майнинг пул», «одноранговая сеть», «криптовалюта», были рассмотрены алгоритмы распределения прибыли и организации сети. Также изучен цикл добычи криптовалют, а также некоторые принципы проектирования. Исследованы разные направления и подходы к решению задач, связанных с разработкой программного средства для данных систем. Основным результатом стало программное средство, которое позволяет добывать криптовалюту с помощью организации вычислительной сети.

Был проведен анализ предметной области и изучены существующие аналоги на IT-рынке, у которых были выявлены наиболее слабые и сильные стороны каждого. Наиболее часто встречающимся недостатком у имеющихся решений оказалось отсутствие графического интерфейса и высокая сложность системы.

В ходе выполненного анализа, чтобы предотвратить на этапе проектирования некоторые ошибки, были выявлены некоторые требования:

- 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.

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

- платформа ASP.Net;
- язык программирования для серверной части C#;
- библиотека Nancy.

На основе функциональных требований было произведено проектирование программного средства. В нем представлены разработка архитектуры ПС, разработка модели базы данных, разработка алгоритма программного средства и алгоритмов отдельных модулей. В разделе «Проектирование и разработка программного средства» приведена обобщенная схема взаимодействия клиент-сервер и общая архитектура системы. Детально описаны общие алгоритмы работы программы, алгоритмы добычи криптовалюты, распределения прибыли, описание также сопровождается схемами алгоритмов.

За период дипломной работы были разработаны:

- программный модуль «Организация сети»;
- программный модуль «Протокол Stratum»;
- программный модуль «Майнер»;
- программный модуль «Майнер-менеджер»;
- программный модуль «Операции с хеш кодом»;
- программный модуль «Распределение прибыли»;
- программный модуль «Майнинг пул».

Согласно объявленным требованиям были сформированы тестовые наборы, которые успешно пройдены в ходе тестовых испытаний программного средства. Успешность прохождения тестов показывает корректность работы программы с реальными данными и соответствие функциональным требованиям.

Также описана методика использования разработанного программного средства, которая позволяет за достаточно короткие сроки освоить работу с программой.

– Также в ходе работы над дипломным проектом рассмотрена экономическая сторона проектирования и разработки программного средства, рассчитаны экономический эффект от внедрения программного средства и показатели эффективности использования программного средства у пользователя. В результате расчётов подтвердилась целесообразность разработки. Так как разработанная система реализует новую концепцию процесса доставки и использует самые современные технологии, инвестиции, вложенные в разработку, окупаются за первый год использования программного продукта. экономический эффект за год работы программного средства составит 13173,52 руб;

В дальнейшем будет проводиться оптимизация программного средства, рефакторинг кода, добавление новых алгоритмов распределения прибыли, реализация настройки конфигураций в графическом интерфейсе, организовать систему сообщений между пользователями внутри программного средства, реализация нотификации о событиях в сети на удобный для клиента способ связи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Cryptocurrency trading: how to make money by trading bitcoin and another cryptocurrency / Devan Hansel, 2017 – 144с.
- [2] Свон, М. Блокчейн: Схема новой экономики / М. Свон: Олимп-Бизнес, 2017 - 240 с.
- [3] Что такое майнинг криптовалют – полный обзор [Электронный ресурс] – 2018, Режим доступа: <https://bitinfo.by/akademiya/chto-takoe-majning/> Дата доступа: 14.04.2018
- [4] Простокоеин - Ваш проводник в мире криптовалют [Электронный ресурс] – 2018, Режим доступа: <https://prostocoin.com/blog/what-is-mining> Дата доступа: 15.04.18
- [5] Рассел, Дж. Одноранговая сеть / Дж. Рассел, Р. Сохн //Книга по Требованию, 2012 – 101с.
- [6] Фримен, Э. Паттерны проектирования / Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс // Питер, 2018 – 656с.
- [7] Система выплат PROP: особенности / RBC - [Электронный ресурс] – 2018, Режим доступа: <https://probtc.info/materialy/34520/> Дата доступа: 16.04.2018
- [8] ASP.NET Documentation / Microsoft [Электронный ресурс] – 2018, Режим доступа: <https://docs.microsoft.com/en-us/aspnet/> Дата доступа: 20.04.2018
- [9] METANIT.COM Сайт о программировании [Электронный ресурс] – 2018, Режим доступа: <https://metanit.com/sharp/mvc5/1.2.php> Дата доступа: 16.04.18
- [10] Рихтер, Дж., CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер // Питер, 2016 – 896с
- [11] NancyFx/Nancy: Lightweight, low-ceremony, framework for building HTTP based services on .Net and Mono / GitHub [Электронный ресурс] – 2018, Режим доступа: <https://github.com/NancyFx/Nancy> Дата доступа: 19.04.2018
- [12] Хант, Кр. TCP/IP. Сетевое администрирование / Кр. Хант // Символ, 2015 – 816 с.
- [13] Класс Socket (System.Net.Sockets) / Microsoft 2018 [Электронный ресурс] – 2018, Режим доступа: [https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket\(v=vs.110\).aspx](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket(v=vs.110).aspx) Дата доступа: 24.04.2018
- [14] Протоколы TCP и UDP / Professor web. net & web programming [Электронный ресурс] – 2018, Режим доступа: https://professor-web.ru/my/csharp/web/level1/1_4.php Дата доступа: 25.04.2018

[15] Stratum Bitcoin Protocol / bitcoinwiki [Электронный ресурс] – 2018, Режим доступа: https://en.bitcoin.it/wiki/Stratum_mining_protocol Дата доступа: 27.04.2018

[16] МТБлог. О деньгах, бизнесе, банках и идеях [Электронный ресурс] – 2018, Режим доступа: <http://mtblog.mtbank.by/chto-takoe-kriptovalyuta-i-kak-ee-zarabotat-razvernutyj-putevoditel-v-voprosah-i-otvetah/> Дата доступа: 16.04.18

[17] Cryptocurrency Mining: The Complete Guide to Mining Bitcoin, Ethereum and other Cryptocurrency. - Devan Hansel, 2017 – 135 с.

[18] Что выгоднее PPS или PPLNS / Russian Bitcoin Community [Электронный ресурс] – 2018, Режим доступа: <https://probtc.info/materialy/30642/> Дата доступа: 16.04.18

[19] Pool Terminology / SlushPool [Электронный ресурс] – 2018, Режим доступа: <https://slushpool.com/help/topic/terminology/> Дата доступа: 16.04.18

[20] Виды тестирования / STORMNET [Электронный ресурс] – 2018, Режим доступа: <http://www.it-courses.by/all-software-testing-types/> Дата доступа: 15.05.18

[21] Палицын, В. А. Техничко-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В.А. Палицын. – Мн.: БГУИР, 2006. – 76 с

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный текст программы

Текст программного модуля запросы для отображения веб-страниц:

```
public class PoolModule : NancyModule
{
    public PoolModule(IPoolManager poolManager)
        : base("/pool")
    {
        Get["/{slug}"] = _ =>
        {
            var pool = poolManager.Get(HttpUtility.HtmlEncode(_slug));
            // find the requested pool.

            if (pool == null)
            {
                return View["error", new ErrorViewModel
                {
                    Details = string.Format("The requested pool does not
exist: {0}", _slug)
                }];
            }

            ViewBag.Title = string.Format("{0} Pool", pool.Con-
fig.Coin.Name);
            ViewBag.Heading = string.Format("{0} Pool", pool.Con-
fig.Coin.Name);

            // return our view
            return View["pool", new PoolModel
            {
                Pool = pool
            }];
        };

        Get["/{slug}/workers"] = _ =>
        {
            var pool = poolManager.Get(HttpUtility.HtmlEncode(_slug));
            // find the requested pool.

            if (pool == null) // make sure queried pool exists.
            {
                return View["error", new ErrorViewModel
                {
                    Details = string.Format("The requested pool does not
exist: {0}", _slug)
                }];
            }

            ViewBag.Header = string.Format("{0} Workers", pool.Con-
fig.Coin.Name);

            // return our view
            return View["workers", new WorkersModel
            {
                Workers = pool.MinerManager.Miners
            }];
        };
    }
}
```

```

};

Get["/{slug}/round"] = _ =>
{
    var pool = poolManager.Get(HttpUtility.HtmlEncode(_slug));
    // find the requested pool.

    if (pool == null) // make sure queried pool exists.
    {
        return View["error", new ErrorViewModel
        {
            Details = string.Format("The requested pool does not
exist: {0}", _slug)
        }
    ];

    ViewBag.Header = string.Format("{0} Current Round", pool.Con-
fig.Coin.Name);

    // return our view
    return View["round", new RoundModel
    {
        Round = pool.NetworkInfo.Round,
        Shares = pool.RoundShares
    }
    ];
};

Get["/{slug}/blocks/{page?1}"] = _ =>
{
    var pool = (IPool)poolManager.Get(HttpUtility.HtmlEn-
code(_slug)); // find the requested pool.

    if (pool == null) // make sure queried pool exists.
    {
        return View["error", new ErrorViewModel
        {
            Details = string.Format("The requested pool does not
exist: {0}", _slug)
        }
    ];

    int page;
    if (!Int32.TryParse(_page, out page))
        page = 1;

    var paginationQuery = new PaginationQuery(page);

    var blocks = pool.BlockRepository.GetBlocks(paginationQuery);

    if (blocks.Count == 0)
    {
        return View["error", new ErrorViewModel
        {
            Details = "No more blocks exist"
        }
    ];

    var model = new BlocksModel
    {
        Blocks = blocks,
        Coin = pool.Config.Coin,
        Filter = BlockFilter.All,
    };
};

```

```

        PaginationQuery = paginationQuery
    };

    return View["blocks", model];
};

Get["/{slug}/blocks/paid/{page?1}"] = _ =>
{
    var pool = (IPool)poolManager.Get(HttpUtility.HtmlEn-
code(_slug)); // find the requested pool.

    if (pool == null) // make sure queried pool exists.
    {
        return View["error", new ErrorViewModel
        {
            Details = string.Format("The requested pool does not
exist: {0}", _slug)
        }
    ];
    }

    int page;
    if (!Int32.TryParse(_page, out page))
        page = 1;

    var paginationQuery = new PaginationQuery(page);

    var blocks = pool.BlockRepository.GetPaidBlocks(pagination-
Query);

    if (blocks.Count == 0)
    {
        return View["error", new ErrorViewModel
        {
            Details = "No more blocks exist"
        }
    ];
    }

    var model = new BlocksModel
    {
        Blocks = blocks,
        Coin = pool.Config.Coin,
        Filter = BlockFilter.PaidOnly,
        PaginationQuery = paginationQuery
    };

    return View["blocks", model];
};

Get["/{slug}/block/{height:int}"] = _ =>
{
    var pool = (IPool)poolManager.Get(HttpUtility.HtmlEn-
code(_slug)); // find the requested pool.

    if (pool == null) // make sure queried pool exists.
    {
        return View["error", new ErrorViewModel
        {
            Details = string.Format("The requested pool does not
exist: {0}", _slug)
        }
    ];
    }
}

```

```

        var block = pool.BlockRepository.Get((uint)_.height);

        if (block == null)
        {
            return View["error", new ErrorViewModel
            {
                Details = string.Format("The requested block does not
exist: {0}", _.height)
            }];
        }

        var model = new BlockModel
        {
            Block = block,
            Coin = pool.Config.Coin,
            Payments = pool.PaymentRepository.GetPay-
mentDetailsForBlock((uint)_.height)
        };

        ViewBag.Header = string.Format("Block {0}", block.Height);
        ViewBag.SubHeader = string.Format("{0} block", pool.Con-
fig.Coin.Name);

        return View["block", model];
    };

    Get["/{slug}/tx/{id:int}"] = _ =>
    {
        var pool = (IPool)poolManager.Get(HttpUtility.HtmlEn-
code(_.slug)); // find the requested pool.

        if (pool == null) // make sure queried pool exists.
        {
            return View["error", new ErrorViewModel
            {
                Details = string.Format("The requested pool does not
exist: {0}", _.slug)
            }];
        }

        var details = pool.PaymentRepository.GetPaymentDetails-
ByTransactionId((uint)_.id);

        if (details == null)
        {
            return View["error", new ErrorViewModel
            {
                Details = string.Format("The requested transaction
does not exist.")
            }];
        }

        var account = pool.AccountManager.GetAccountById(details.Ac-
countId);
        var block = pool.BlockRepository.Get((uint) details.Block);

        ViewBag.Header = string.Format("Transaction Details");
        ViewBag.SubHeader = string.Format("{0}", details.Transac-
tionId);

        var model = new PaymentDetailsModel
        {

```

```

        Details = details,
        Account = account,
        Block = block,
        Coin = pool.Config.Coin
    };

    return View["paymentdetails", model];
};

Get["/{slug}/payment/{id:int}"] = _ =>
{
    var pool = (IPool)poolManager.Get(HttpUtility.HtmlEn-
code(_slug)); // find the requested pool.

    if (pool == null) // make sure queried pool exists.
    {
        return View["error", new ErrorViewModel
        {
            Details = string.Format("The requested pool does not
exist: {0}", _slug)
        }
    ];
    }

    var details = pool.PaymentRepository.GeyPaymentDetailsByPay-
mentId((uint)_id);

    if (details == null)
    {
        return View["error", new ErrorViewModel
        {
            Details = string.Format("The requested payment does
not exist.")
        }
    ];
    }

    var account = pool.AccountManager.GetAccountById(details.Ac-
countId);
    var block = pool.BlockRepository.Get((uint)details.Block);

    ViewBag.Header = string.Format("Payment Details");
    ViewBag.SubHeader = string.Format("{0}", details.PaymentId);

    var model = new PaymentDetailsModel
    {
        Details = details,
        Account = account,
        Block = block,
        Coin = pool.Config.Coin
    };

    return View["paymentdetails", model];
};

Get["/{slug}/account/address/{address:length(26,34)}/{page?1}"] =
_ =>
{
    var pool = (IPool)poolManager.Get(HttpUtility.HtmlEn-
code(_slug)); // find the requested pool.

    if (pool == null)
    {
        return View["error", new ErrorViewModel

```

```

        {
            Details = string.Format("The requested pool does not
exist: {0}", HttpUtility.HtmlEncode(_slug))
        }];
    }

    var account = (IAccount)pool.AccountManager.GetAc-
countByAddress(_address);

    if (account == null)
    {
        return View["error", new ErrorViewModel
        {
            Details = string.Format("The requested account does
not exist: {0}", _address)
        }];
    }

    int page;
    if (!Int32.TryParse(_page, out page))
        page = 1;

    var paginationQuery = new PaginationQuery(page);

    // get the payments for the account.
    var payments = pool.AccountManager.GetPaymentsForAccount(ac-
count.Id, paginationQuery);

    ViewBag.Header = string.Format("Account Details");
    ViewBag.SubHeader = account.Username;

    // return our view
    return View["account", new AccountModel
    {
        Account = account,
        Coin = pool.Config.Coin,
        Payments = payments,
        PaginationQuery = paginationQuery
    }];
}

}

public class ApiModule: NancyModule
{
    private static readonly Response PoolNotFound = JsonConvert.Serial-
izeObject(new JsonError("Pool not found!"));
    private static readonly Response AlgorithmNotFound = JsonConvert.Se-
rializeObject(new JsonError("Algorithm not found!"));

    public ApiModule(IStatisticsManager statisticsManager, IPoolManager
poolManager, IAlgorithmManager algorithmManager)
        :base("/api")
    {
        Get["/"] = _ =>
        {
            // include common data required by layout.
            ViewBag.Header = "Public API";

            // return our view
            return View["api", new ApiModel
            {

```

```

        baseUrl = Request.Url.SiteBase,
        Coin = poolManager.First().Config.Coin
    }];

    Get["/pools"] = _ =>
    {
        var response = (Response) poolManager.ServiceResponse;
        response.ContentType = "application/json";
        return response;
    };

    Get["/pool/{slug}"] = _ =>
    {
        var pool = poolManager.Get(HttpUtility.HtmlEncode(_.slug));
        // query the requested pool.

        var response = pool != null ? (Response)pool.ServiceResponse
: PoolNotFound;
        response.ContentType = "application/json";
        return response;
    };

    Get["/algorithms"] = _ =>
    {
        var response = (Response)algorithmManager.ServiceResponse;
        response.ContentType = "application/json";
        return response;
    };

    Get["/algorithm/{slug}"] = _ =>
    {
        var algorithm = algorithmManager.Get(HttpUtility.HtmlEn-
code(_.slug)); // query the requested pool.

        var response = algorithm != null ? (Response)algorithm.Ser-
viceResponse : AlgorithmNotFound;
        response.ContentType = "application/json";
        return response;
    };

    Get["/global"] = _ =>
    {
        var response = (Response) statisticsManager.ServiceResponse;
        response.ContentType = "application/json";
        return response;
    };
}
}

```

Текст программного модуля организация сети:

```

public class SocketServer : ISocketServer, IDisposable
{
    /// <summary>
    /// The IP address of the interface the server binded.
    /// </summary>
    public string BindInterface { get; protected set; }

    /// <summary>

```



```

    /// The listening port for the server.
    /// </summary>
    public int Port { get; protected set; }

    /// <summary>
    /// Is server currently listening for connections?
    /// </summary>
    public bool IsListening { get; private set; }

    /// <summary>
    /// Listener socket.
    /// </summary>
    protected Socket Listener;

    /// <summary>
    /// List of connections.
    /// </summary>
    protected List<IConnection> Connections = new List<IConnection>();

    /// <summary>
    /// Used for locking connections list.
    /// </summary>
    protected object ConnectionLock = new object();

    // connection event handlers.
    public delegate void ConnectionEventHandler(object sender,
ConnectionEventArgs e);
    public delegate void BannedConnectionEventHandler(object sender,
BannedConnectionEventArgs e);
    public delegate void ConnectionDataEventHandler(object sender,
ConnectionDataEventArgs e);

    // connection events.
    public event ConnectionEventHandler ClientConnected;
    public event ConnectionEventHandler ClientDisconnected;
    public event BannedConnectionEventHandler BannedConnection;
    public event ConnectionDataEventHandler DataReceived;
    public event ConnectionDataEventHandler DataSent;

    protected ILogger _logger;

    /// <summary>
    /// Is the instance disposed?
    /// </summary>
    private bool _disposed;

    #region listener & accept callbacks.

    /// <summary>
    /// Start listening on given interface and port.
    /// </summary>
    /// <param name="bindIP">The interface IP to listen for
connections.</param>
    /// <param name="port">The port to listen for connections.</param>
    /// <returns></returns>
    protected virtual bool Listen(string bindIP, int port)
    {
        // Check if the server instance has been already disposed.
        if (_disposed)
            throw new ObjectDisposedException(GetType().Name, "Server
instance has been already disposed.");
    }

```

```

        // Check if the server is already listening.
        if (IsListening)
            throw new InvalidOperationException(string.Format("Server is
already listening on {0}:{1}", bindIP, port));

        try
        {
            // Create new TCP socket and set socket options.
            Listener = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);

            // Setup our options:
            // * NoDelay - true - don't use packet coalescing
            // * DontLinger - true - don't keep sockets around once
they've been disconnected
            Listener.SetSocketOption(SocketOptionLevel.Tcp,
SocketOptionName.NoDelay, true);
            Listener.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.DontLinger, true);

            // try the actual bind.
            Listener.Bind(new IPEndPoint(IPAddress.Parse(bindIP), port));
            Port = port;

            // Start listening for incoming connections.
            Listener.Listen(int.MaxValue); // let the maximum amount of
accept backlog - we are basically leaving it to OS to determine the value.
// by setting the maximum
available value, we can make sure that we can handle large amounts of
concurrent
// connection requests (maybe
after a server restart).

            // http://blog.stephencleary.com/2009/05/using-socket-as-
server-listening-socket.html
            // The "backlog" parameter to Socket.Listen is how many
connections the OS may accept on behalf of the application. This is not
            // the total number of active connections; it is only how
many connections will be established if the application "gets behind".
            // Once connections are Accepted, they move out of the
backlog queue and no longer "count" against the backlog limit.

            // The .NET docs fail to mention that int.MaxValue can be
used to invoke the "dynamic backlog" feature (Windows Server systems only),
            // essentially leaving it up to the OS. It is tempting to set
this value very high (e.g., always passing int.MaxValue), but this would
            // hurt system performance (on non-server machines) by pre-
allocating a large amount of scarce resources. This value should be set to a
            // reasonable amount (usually between 2 and 5), based on how
many connections one is realistically expecting and how quickly they can be
            // Accepted.

            IsListening = true;

            // Begin accepting any incoming connections asynchronously.
            Listener.BeginAccept(AcceptCallback, null);

            return true;
        }
        catch (SocketException exception)
        {

```

```

        _logger.Fatal("{0} can not bind on {1}, server shutting
down.. Reason: {2}", GetType().Name, bindIP, exception);
        Shutdown();
        return false;
    }
}

/// <summary>
/// Accept callbacks from listener socket.
/// </summary>
/// <param name="result"></param>
private void AcceptCallback(IAsyncResult result)
{
    if (Listener == null)
        return;

    try
    {
        var socket = Listener.EndAccept(result); // Finish accepting
the incoming connection.
        var banned = IsBanned(socket);

        if (banned)
        {
            var endpoint = socket.RemoteEndPoint;
            socket.Disconnect(true);
            OnBannedConnection(new
BannedConnectionEventArgs(endpoint));
        }
        else
        {
            var connection = new Connection(this, socket); // Track
the connection.

            lock (ConnectionLock)
                Connections.Add(connection); // Add the new
connection to the active connections list.

            OnClientConnection(new ConnectionEventArgs(connection));
// Raise the ClientConnected event.
            connection.BeginReceive(ReceiveCallback, connection); //
Begin receiving on the new connection connection.
        }
    }
    catch (Exception exception)
    {
        _logger.Error(exception, "Can not accept connection");
    }
    finally
    {
        // no matter we were able to accept last connection request,
make sure we continue to listen for new connections.
        Listener.BeginAccept(AcceptCallback, null);
    }
}

public virtual bool IsBanned(Socket socket)
{
    return false;
}

#endregion

```

```

#region recieve callback

private void ReceiveCallback(IAsyncResult result)
{
    var connection = result.AsyncState as Connection; // Get the
connection connection passed to the callback.

    if (connection == null)
        return;

    try
    {
        var bytesRecv = connection.EndReceive(result); // Finish
receiving data from the socket.

        if (bytesRecv > 0)
        {
            OnDataReceived(new ConnectionDataEventArgs(connection,
connection.RecvBuffer.Enumerate(0, bytesRecv))); // Raise the DataReceived
event.

            // Begin receiving again on the socket, if it is
connected.
            if (connection.IsConnected)
                connection.BeginReceive(ReceiveCallback, connection);
        }
        else
            RemoveConnection(connection); // Connection was lost.
    }
    catch (SocketException e)
    {
        _logger.Error(e, "SocketException on ReceiveCallback");
        RemoveConnection(connection); // An error occured while
receiving, connection has disconnected.
    }
}

#endregion

#region send methods

public virtual int Send(Connection connection, byte[] buffer, int
start, int count, SocketFlags flags)
{
    if (connection == null)
        throw new ArgumentNullException("connection");

    if (buffer == null)
        throw new ArgumentNullException("buffer");

    if (!connection.IsConnected)
        return 0;

    var totalBytesSent = 0;
    var bytesRemaining = buffer.Length;

    try
    {
        while (bytesRemaining > 0) // Ensure we send every byte.
        {

```

```

        int bytesSent = connection.Socket.Send(buffer, start,
count, flags);

        if (bytesSent > 0)
            OnDataSent(new ConnectionDataEventArgs(connection,
buffer.Enumerate(totalBytesSent, bytesSent))); // Raise the Data Sent event.

        // Decrement bytes remaining and increment bytes sent.
        bytesRemaining -= bytesSent;
        totalBytesSent += bytesSent;
    }
}
catch (SocketException socketException)
{
    RemoveConnection(connection); // An error occurred while
sending, connection has disconnected.
    _logger.Error(socketException, "Send");
}
catch (Exception e)
{
    RemoveConnection(connection); // An error occurred while
sending, it is possible that the connection has a problem.
    _logger.Error(e, "Send");
}

    return totalBytesSent;
}
public virtual int Send(Connection connection, IEnumerable<byte>
data, SocketFlags flags)
{
    if (connection == null)
        throw new ArgumentNullException("connection");

    if (data == null)
        throw new ArgumentNullException("data");

    var buffer = data.ToArray();
    return Send(connection, buffer, 0, buffer.Length,
SocketFlags.None);
}

#endregion

#region disconnect & shutdown handlers

public void RemoveConnection(IConnection connection)
{
    if (connection == null)
    {
        _logger.Error("Exception connection doesn't not exist.");
        return;
    }

    lock (connection)
    {
        if (connection.IsConnected) // disconnect the client
        {
            try
            {
                connection.Socket.Disconnect(true);
            } catch (Exception ex) {

```

```

        _logger.Error(ex, "Exception on client
disconnection");
    }
}

connection.Client = null;

// Remove the connection from the dictionary and raise the
OnDisconnection event.
lock (ConnectionLock)
{
    if (Connections.Contains(connection))
        Connections.Remove(connection);
}

OnClientDisconnect(new ConnectionEventArgs(connection)); // raise
the ClientDisconnected event.
}

/// <summary>
/// Shuts down the server instance.
/// </summary>
public virtual void Shutdown()
{
    // Check if the server has been disposed.
    if (_disposed)
        throw new ObjectDisposedException(GetType().Name, "Server has
been already disposed.");

    // Check if the server is actually listening.
    if (!IsListening)
        return;

    // Close the listener socket.
    if (Listener != null)
    {
        Listener.Close();
        Listener = null;
    }

    // Disconnect the clients.
    DisconnectAll();

    Listener = null;
    IsListening = false;
}

public virtual void DisconnectAll()
{
    lock (ConnectionLock)
    {
        foreach (var connection in Connections.ToList()) // using
ToList() to get a copy in order to prevent any modified collection
exceptions.
        {
            RemoveConnection(connection);
        }

        Connections.Clear();
    }
}

```

```

#endregion

#region service methods

public IEnumerable<IConnection> GetConnections()
{
    lock (ConnectionLock)
        foreach (IConnection connection in Connections)
            yield return connection;
}

#endregion

#region server control
public virtual bool Start()
{
    throw new NotImplementedException();
}

public virtual bool Stop()
{
    throw new NotImplementedException();
}

#endregion

#region events

private void OnClientConnection(ConnectionEventArgs e)
{
    var handler = ClientConnected;

    if (handler != null)
        handler(this, e);
}

private void OnClientDisconnect(ConnectionEventArgs e)
{
    var handler = ClientDisconnected;

    if (handler != null)
        handler(this, e);
}

private void OnBannedConnection(BannedConnectionEventArgs e)
{
    var handler = BannedConnection;

    if (handler != null)
        handler(this, e);
}

private void OnDataReceived(ConnectionDataEventArgs e)
{
    var handler = DataReceived;

    if (handler != null)
        handler(this, e);
}

```

```

private void OnDataSent(ConnectionDataEventArgs e)
{
    var handler = DataSent;

    if (handler != null)
        handler(this, e);
}

#endregion

#region de-ctor

public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

protected virtual void Dispose(bool disposing)
{
    if (_disposed) return;

    if (disposing)
    {
        Shutdown(); // Close the listener socket.
        DisconnectAll(); // Disconnect all users.
    }

    // Dispose of unmanaged resources here.

    _disposed = true;
}

#endregion
}

```

Текст программного модуля «Майнер» и «Майнер-менеджер»:

```

public class MinerManager : IMinerManager
{
    public int Count { get { return _miners.Count(kvp => kvp.Value.Authenticated); } }

    public IList<IMiner> Miners { get { return _miners.Values.ToList(); } }

    public event EventHandler MinerAuthenticated;

    private readonly Dictionary<int, IMiner> _miners;

    private int _counter = 0; // counter for assigning unique id's to miners.

    private readonly IPoolConfig _poolConfig;

    private readonly IStorageLayer _storageLayer;

    private readonly IAccountManager _accountManager;

    private readonly ILogger _logger;

    /// <summary>

```



```

    /// Used for locking miners list.
    /// </summary>
    private readonly object _minersLock = new object();

    public MinerManager(IPoolConfig poolConfig, IStorageLayer storageLayer, IAccountManager accountManager)
    {
        _poolConfig = poolConfig;
        _storageLayer = storageLayer;
        _accountManager = accountManager;

        _miners = new Dictionary<int, IMiner>();
        _logger = Log.ForContext<MinerManager>().ForContext("Component", poolConfig.Coin.Name);
    }

    public IMiner GetMiner(Int32 id)
    {
        return _miners.ContainsKey(id) ? _miners[id] : null;
    }

    public IMiner GetByConnection(IConnection connection)
    {
        return (from pair in _miners // returned the miner associated
with the given connection.
            let client = (IClient) pair.Value
            where client.Connection == connection
            select pair.Value).FirstOrDefault();
    }

    public T Create<T>(IPool pool) where T : IGetworkMiner
    {
        var @params = new object[]
        {
            _counter++,
            pool,
            this
        };

        var instance = Activator.CreateInstance(typeof(T), @params); //
create an instance of the miner.
        var miner = (IGetworkMiner)instance;

        lock(_minersLock) // lock the list before we modify the collec-
tion.
            _miners.Add(miner.Id, miner); // add it to our collection.

        return (T)miner;
    }

    public T Create<T>(UInt32 extraNonce, IConnection connection, IPool
pool) where T : IStratumMiner
    {
        var @params = new object[]
        {
            _counter++,
            extraNonce,
            connection,
            pool,
            this,
            _storageLayer
        };
    }

```

```

        var instance = Activator.CreateInstance(typeof(T), @params); //
create an instance of the miner.
        var miner = (IStratumMiner)instance;

        lock (_minersLock) // lock the list before we modify the collec-
tion.
            _miners.Add(miner.Id, miner); // add it to our collection.

        return (T)miner;
    }

    public void Remove(IConnection connection)
    {
        // find the miner associated with the connection.
        var miner = (from pair in _miners
            let client = (IClient) pair.Value
            where client.Connection == connection
            select pair.Value).FirstOrDefault();

        if (miner == null) // make sure the miner exists
            return;

        lock (_minersLock) // lock the list before we modify the collec-
tion.
            _miners.Remove(miner.Id); // remove the miner.
    }

    public void Authenticate(IMiner miner)
    {
        // if username validation is not on just authenticate the miner,
        else ask the current storage layer to do so.
        miner.Authenticated = !_poolConfig.Miner.ValidateUsername ||
        _storageLayer.Authenticate(miner);

        _logger.Debug(
            miner.Authenticated ? "Authenticated miner: {0:l} [{1:l}]" :
            "Miner authentication failed: {0:l} [{1:l}]",
            miner.Username, ((IClient) miner).Connection.RemoteEndPoint);

        if (!miner.Authenticated)
            return;

        if (miner is IStratumMiner) // if we are handling a stratum-
miner, apply stratum specific stuff.
        {
            var stratumMiner = (IStratumMiner) miner;
            stratumMiner.SetDifficulty(_poolConfig.Stratum.Diff); // set
the initial difficulty for the miner and send it.
            stratumMiner.SendMessage(_poolConfig.Meta.MOTD); // send the
motd.
        }

        miner.Account = _accountManager.GetAc-
countByUsername(miner.Username); // query the user.
        if (miner.Account == null) // if the user doesn't exists.
        {
            _accountManager.AddAccount(new Account(miner)); // create a
new one.
            miner.Account = _accountManager.GetAc-
countByUsername(miner.Username); // re-query the newly created record.
        }
    }

```

```

        OnMinerAuthenticated(new MinerEventArgs(miner)); // notify lis-
teners about the new authenticated miner.
    }

    // todo: consider exposing this event by miner object itself.
    protected virtual void OnMinerAuthenticated(MinerEventArgs e)
    {
        var handler = MinerAuthenticated;

        if (handler != null)
            handler(this, e);
    }
}

public class MiningSoftware :IMiningSoftware
{
    public string Name { get; private set; }

    public Version Version { get; private set; }

    public IList<IHashAlgorithmStatistics> Algorithms { get; private set; }

    public Platforms Platforms { get; private set; }

    public string Site { get; private set; }

    public IDictionary<string, string> Downloads { get; private set; }

    public MiningSoftware(IAlgorithmManager algorithmManager, IMin-
ingSoftwareConfig config)
    {
        Name = config.Name;

        if(config.Version!=null)
            Version = new Version(config.Version);

        Algorithms = new List<IHashAlgorithmStatistics>();
        foreach(var entry in config.Algorithms)
        {
            var algorithm = algorithmManager.Get(entry);

            if (algorithm == null)
                continue;

            Algorithms.Add(algorithm);
        }

        foreach (var entry in config.Platforms)
        {
            switch (entry)
            {
                case "ati":
                    Platforms = Platforms | Platforms.Ati;
                    break;
                case "asic":
                    Platforms = Platforms | Platforms.Asic;
                    break;
                case "cpu":
                    Platforms = Platforms | Platforms.Cpu;
                    break;
                case "nvidia":

```

```

        Platforms = Platforms | Platforms.Nvidia;
        break;
    }
}

Site = config.Site;
Downloads = config.Downloads.Where(x => x.Value != null).ToDictionary(x => x.Key, x => x.Value);
}

}

[JsonObject(MemberSerialization.OptIn)]
public interface IMiner
{
    /// <summary>
    /// Unique subscription id for identifying the miner.
    /// </summary>
    [JsonProperty("id")]
    int Id { get; }

    /// <summary>
    /// Account for the miner.
    /// </summary>
    IAccount Account { get; set; }

    /// <summary>
    /// Username of the miner.
    /// </summary>
    [JsonProperty("username")]
    string Username { get; }

    /// <summary>
    /// The pool miner is connected to.
    /// </summary>
    IPool Pool { get; }

    /// <summary>
    /// Is the miner authenticated.
    /// </summary>
    [JsonProperty("authenticated")]
    bool Authenticated { get; set; }

    int ValidShareCount { get; set; }

    int InvalidShareCount { get; set; }

    MinerSoftware Software { get; }

    Version SoftwareVersion { get; }

    /// <summary>
    /// Authenticates the miner.
    /// </summary>
    /// <param name="user"></param>
    /// <param name="password"></param>
    /// <returns></returns>
    bool Authenticate(string user, string password);
}

public class MiningSoftwareConfig:IMiningSoftwareConfig
{
    public string Name { get; private set; }
}

```

```

public string Version { get; private set; }
public IList<string> Platforms { get; private set; }
public IList<string> Algorithms { get; private set; }
public string Site { get; private set; }
public IDictionary<string, string> Downloads { get; private set; }
public bool Valid { get; private set; }
public MiningSoftwareConfig(dynamic config)
{
    try
    {
        Name = config.name;
        Version = config.version;
        Platforms = config.platforms;
        Algorithms = config.algorithms;
        Site = config.site;
        Downloads = new Dictionary<string, string>
        {
            {"windows", config.download.windows},
            {"linux", config.download.linux},
            {"macos", config.download.macos},
        };

        Valid = true;
    }
    catch (Exception e)
    {
        Valid = false;
        Log.Logger.ForContext<MiningSoftwareConfig>().Error(e, "Error
loading software configuration");
    }
}
}

```

Текст программного модуля «Обработка хеш кода»:

```

public class Share : IShare
{
    public bool IsValid { get { return Error == ShareError.None; } }
    public bool IsBlockCandidate { get; private set; }
    public Block Block { get; private set; }
    public Transaction GenerationTransaction { get; private set; }
    public bool IsBlockAccepted { get { return Block != null; } }
    public IMiner Miner { get; private set; }
    public ShareError Error { get; private set; }
    public UInt64 JobId { get; private set; }
    public IJob Job { get; private set; }
    public int Height { get; private set; }
    public UInt32 NTime { get; private set; }
    public UInt32 Nonce { get; private set; }
    public UInt32 ExtraNoncel { get; private set; }
    public UInt32 ExtraNonce2 { get; private set; }
    public byte[] CoinbaseBuffer { get; private set; }
    public Hash CoinbaseHash { get; private set; }
    public byte[] MerkleRoot { get; private set; }
    public byte[] HeaderBuffer { get; private set; }
    public byte[] HeaderHash { get; private set; }
    public BigInteger HeaderValue { get; private set; }
    public Double Difficulty { get; private set; }
    public double BlockDiffAdjusted { get; private set; }
    public byte[] BlockHex { get; private set; }
    public byte[] BlockHash { get; private set; }
}

```

```

    public Share(IStratumMiner miner, UInt64 jobId, IJob job, string extraNonce2, string nTimeString, string nonceString)
    {
        Miner = miner;
        JobId = jobId;
        Job = job;
        Error = ShareError.None;

        var submitTime = TimeHelpers.NowInUnixTimestamp(); // time we recieved the share from miner.

        if (Job == null)
        {
            Error = ShareError.JobNotFound;
            return;
        }

        // check size of miner supplied extraNonce2
        if (extraNonce2.Length/2 != ExtraNonce.ExpectedExtraNonce2Size)
        {
            Error = ShareError.IncorrectExtraNonce2Size;
            return;
        }
        ExtraNonce2 = Convert.ToUInt32(extraNonce2, 16); // set extra-Nonce2 for the share.

        // check size of miner supplied nTime.
        if (nTimeString.Length != 8)
        {
            Error = ShareError.IncorrectNTimeSize;
            return;
        }
        NTime = Convert.ToUInt32(nTimeString, 16); // read ntime for the share

        // make sure NTime is within range.
        if (NTime < job.BlockTemplate.CurTime || NTime > submitTime + 7200)
        {
            Error = ShareError.NTimeOutOfRange;
            return;
        }

        // check size of miner supplied nonce.
        if (nonceString.Length != 8)
        {
            Error = ShareError.IncorrectNonceSize;
            return;
        }
        Nonce = Convert.ToUInt32(nonceString, 16); // nonce supplied by the miner for the share.

        // set job supplied parameters.
        Height = job.BlockTemplate.Height; // associated job's block height.
        ExtraNonce1 = miner.ExtraNonce; // extra nonce1 assigned to miner.

        // check for duplicate shares.
        if (!Job.RegisterShare(this)) // try to register share with the job and see if it's duplicated or not.

```

```

        {
            Error = ShareError.DuplicateShare;
            return;
        }

        // construct the coinbase.
        CoinbaseBuffer = Serializers.SerializeCoinbase(Job, ExtraNonce1,
ExtraNonce2);
        CoinbaseHash = Coin.Coinbase.Utls.HashCoinbase(CoinbaseBuffer);

        // create the merkle root.
        MerkleRoot = Job.MerkleTree.WithFirst(CoinbaseHash).Reverse-
Buffer();

        // create the block headers
        HeaderBuffer = Serializers.SerializeHeader(Job, MerkleRoot,
NTime, Nonce);
        HeaderHash = Job.HashAlgorithm.Hash(HeaderBuffer);
        HeaderValue = new BigInteger(HeaderHash);

        // calculate the share difficulty
        Difficulty = ((double)new BigRational(AlgorithmManager.Diff1,
HeaderValue)) * Job.HashAlgorithm.Multiplier;

        // calculate the block difficulty
        BlockDiffAdjusted = Job.Difficulty * Job.HashAlgorithm.Multi-
plier;

        // check if block candidate
        if (Job.Target >= HeaderValue)
        {
            IsBlockCandidate = true;
            BlockHex = Serializers.SerializeBlock(Job, HeaderBuffer,
CoinbaseBuffer, miner.Pool.Config.Coin.Options.IsProofOfStakeHybrid);
            BlockHash = HeaderBuffer.DoubleDigest().ReverseBuffer();
        }
        else
        {
            IsBlockCandidate = false;
            BlockHash = HeaderBuffer.DoubleDigest().ReverseBuffer();

            // Check if share difficulty reaches miner difficulty.
            var lowDifficulty = Difficulty/miner.Difficulty < 0.99; //
share difficulty should be equal or more then miner's target difficulty.

            if (!lowDifficulty) // if share difficulty is high enough to
match miner's current difficulty.
                return; // just accept the share.

            if (Difficulty >= miner.PreviousDifficulty) // if the diffi-
culty matches miner's previous difficulty before the last vardiff triggered
difficulty change
                return; // still accept the share.

            // if the share difficulty can't match miner's current diffi-
culty or previous difficulty
            Error = ShareError.LowDifficultyShare; // then just reject
the share with low difficult share error.
        }
    }

    public void SetFoundBlock(Block block, Transaction genTx)

```

```

        {
            Block = block;
            GenerationTransaction = genTx;
        }
    }

public class ShareManager : IShareManager
{
    public event EventHandler BlockFound;

    public event EventHandler ShareSubmitted;

    private readonly IJobTracker _jobTracker;

    private readonly IDaemonClient _daemonClient;

    private readonly IStorageLayer _storageLayer;

    private readonly IPoolConfig _poolConfig;

    private string _poolAccount;

    private readonly ILogger _logger;

    /// <summary>
    /// Initializes a new instance of the <see cref="ShareManager" />
class.
    /// </summary>
    /// <param name="poolConfig"></param>
    /// <param name="daemonClient"></param>
    /// <param name="jobTracker"></param>
    /// <param name="storageLayer"></param>
    public ShareManager(IPoolConfig poolConfig, IDaemonClient daemonCli-
ent, IJobTracker jobTracker, IStorageLayer storageLayer)
    {
        _poolConfig = poolConfig;
        _daemonClient = daemonClient;
        _jobTracker = jobTracker;
        _storageLayer = storageLayer;
        _logger = Log.ForContext<ShareManager>().ForContext("Component",
poolConfig.Coin.Name);

        FindPoolAccount();
    }

    /// <summary>
    /// Processes the share.
    /// </summary>
    /// <param name="miner">The miner.</param>
    /// <param name="jobId">The job identifier.</param>
    /// <param name="extraNonce2">The extra nonce2.</param>
    /// <param name="nTimeString">The n time string.</param>
    /// <param name="nonceString">The nonce string.</param>
    /// <returns></returns>
    public IShare ProcessShare(IStratumMiner miner, string jobId, string
extraNonce2, string nTimeString, string nonceString)
    {
        // check if the job exists
        var id = Convert.ToUInt64(jobId, 16);
        var job = _jobTracker.Get(id);

        // create the share

```



```

        var share = new Share(miner, id, job, extraNonce2, nTimeString,
nonceString);

        if (share.IsValid)
            HandleValidShare(share);
        else
            HandleInvalidShare(share);

        OnShareSubmitted(new ShareEventArgs(miner)); // notify the lis-
teners about the share.

        return share;
    }

    public IShare ProcessShare(IGetworkMiner miner, string data)
    {
        throw new NotImplementedException();
    }

    private void HandleValidShare(IShare share)
    {
        var miner = (IStratumMiner) share.Miner;
        miner.ValidShareCount++;

        _storageLayer.AddShare(share); // commit the share.
        _logger.Debug("Share accepted at {0:0.00}/{1} by miner {2:1}",
share.Difficulty, miner.Difficulty, miner.Username);

        // check if share is a block candidate
        if (!share.IsBlockCandidate)
            return;

        // submit block candidate to daemon.
        var accepted = SubmitBlock(share);

        if (!accepted) // if block wasn't accepted
            return; // just return as we don't need to notify about it
and store it.

        OnBlockFound(EventArgs.Empty); // notify the listeners about the
new block.

        _storageLayer.AddBlock(share); // commit the block details to
storage.
        _storageLayer.MoveCurrentShares(share.Height); // move associated
shares to new key.
    }

    private void HandleInvalidShare(IShare share)
    {
        var miner = (IStratumMiner) share.Miner;
        miner.InvalidShareCount++;

        JsonRpcException exception = null; // the exception determined by
the stratum error code.
        switch (share.Error)
        {
            case ShareError.DuplicateShare:
                exception = new DuplicateShareError(share.Nonce);
                break;
            case ShareError.IncorrectExtraNonce2Size:
                exception = new OtherError("Incorrect extranonce2 size");

```

```

        break;
    case ShareError.IncorrectNTimeSize:
        exception = new OtherError("Incorrect nTime size");
        break;
    case ShareError.IncorrectNonceSize:
        exception = new OtherError("Incorrect nonce size");
        break;
    case ShareError.JobNotFound:
        exception = new JobNotFoundError(share.JobId);
        break;
    case ShareError.LowDifficultyShare:
        exception = new LowDifficultyShare(share.Difficulty);
        break;
    case ShareError.NTimeOutOfRange:
        exception = new OtherError("nTime out of range");
        break;
    }
    JsonRpcContext.SetException(exception); // set the stratum excep-
tion within the json-rpc reply.

    Debug.Assert(exception != null); // exception should be never
    null when the share is marked as invalid.
    _logger.Debug("Rejected share by miner {0:1}, reason: {1:1}",
miner.Username, exception.message);
}

private bool SubmitBlock(IShare share)
{
    // TODO: we should try different submission techniques and proba-
    bly more then once: https://github.com/ahmedbodi/stratum-mining/blob/master/lib/bitcoin\_rpc.py#L65-123

    try
    {
        if (_poolConfig.Coin.Options.SubmitBlockSupported) // see if
        submitblock() is available.
            _daemonClient.SubmitBlock(share.BlockHex.ToHexString());
        // submit the block.
        else
            _daemonClient.GetBlockTemplate(share.BlockHex.ToHex-
String()); // use getblocktemplate() if submitblock() is not supported.

        var block = _daemonClient.GetBlock(share.BlockHash.ToHex-
String()); // query the block.

        if (block == null) // make sure the block exists
            return false;

        if (block.Confirmations == -1) // make sure the block is ac-
        cepted.
        {
            _logger.Debug("Submitted block [{0}] is orphaned;
            [{1:1}]", block.Height, block.Hash);
            return false;
        }

        var expectedTxHash = share.CoinbaseHash.Bytes.Reverse-
Buffer().ToHexString(); // calculate our expected generation transactions's
        hash
        var genTxHash = block.Tx.First(); // read the hash of very
        first (generation transaction) of the block
    }
}

```

```

        if (expectedTxHash != genTxHash) // make sure our calculated
generated transaction and one reported by coin daemon matches.
        {
            _logger.Debug("Submitted block [{0}] doesn't seem to be-
long us as reported generation transaction hash [{1:1}] doesn't match our ex-
pected one [{2:1}]", block.Height, genTxHash, expectedTxHash);
            return false;
        }

        var genTx = _daemonClient.GetTransaction(block.Tx.First());
// get the generation transaction.

        // make sure we were able to read the generation transaction
        if (genTx == null)
        {
            _logger.Debug("Submitted block [{0}] doesn't seem to be-
long us as we can't read the generation transaction on our records [{1:1}]",
block.Height, block.Tx.First());
            return false;
        }

        var poolOutput = genTx.GetPoolOutput(_poolConfig.Wal-
let.Address, _poolAccount); // get the output that targets pool's central ad-
dress.

        // make sure the blocks generation transaction contains our
central pool wallet address
        if (poolOutput == null)
        {
            _logger.Debug("Submitted block [{0}] doesn't seem to be-
long us as generation transaction doesn't contain an output for pool's cen-
tral wallet address: {0:}", block.Height, _poolConfig.Wallet.Address);
            return false;
        }

        // if the code flows here, then it means the block was suc-
cessfully submitted and belongs to us.
        share.SetFoundBlock(block, genTx); // assign the block to
share.

        _logger.Information("Found block [{0}] with hash [{1:1}]",
share.Height, share.BlockHash.ToHexString());

        return true;
    }
    catch (RpcException e)
    {
        // unlike BlockProcessor's detailed exception handling and
decision making based on the error,
        // here in share-manager we only one-shot submissions. If we
get an error, basically we just don't care about the rest
        // and flag the submission as failed.
        _logger.Debug("We thought a block was found but it was re-
jected by the coin daemon; [{0:1}] - reason; {1:1}", share.BlockHash.ToHex-
String(), e.Message);
        return false;
    }
}

private void OnBlockFound(EventArgs e)
{
    var handler = BlockFound;

```

```

        if (handler != null)
            handler(this, e);
    }
    private void OnShareSubmitted(EventArgs e)
    {
        var handler = ShareSubmitted;

        if (handler != null)
            handler(this, e);
    }
    private void FindPoolAccount()
    {
        try
        {
            _poolAccount = !_poolConfig.Coin.Options.UseDefaultAccount //
if UseDefaultAccount is not set
                ? _daemonClient.GetAccount(_poolConfig.Wallet.Address) //
find the account of the our pool address.
                : ""; // use the default account.
        }
        catch (RpcException e)
        {
            _logger.Error("Error getting account for pool central wallet
address: {0:1} - {1:1}", _poolConfig.Wallet.Address, e.Message);
        }
    }
}

```

Текст программного модуля «Распределения прибыли»:

```

public GenerationTransaction(IExtraNonce extraNonce, IDaemonClient daemonCli-
ent, IBlockTemplate blockTemplate, IPoolConfig poolConfig)
{
    // TODO: we need a whole refactoring here.
    // we should use DI and it shouldn't really require daemonClient
connection to function.

    BlockTemplate = blockTemplate;
    ExtraNonce = extraNonce;
    PoolConfig = poolConfig;

    Version = blockTemplate.Version;
    TxMessage = Serializers.SerializeString(poolConfig.Meta.TxMes-
sage);

    LockTime = 0;

    // transaction inputs
    Inputs = new List<TxIn>
    {
        new TxIn
        {
            PreviousOutput = new OutPoint
            {
                Hash = Hash.ZeroHash,
                Index = (UInt32) Math.Pow(2, 32) - 1
            },
            Sequence = 0x0,
            SignatureScript =
                new SignatureScript(
                    blockTemplate.Height,

```

```

        blockTemplate.CoinBaseAux.Flags,
        TimeHelpers.NowInUnixTimestamp(),
        (byte) extraNonce.ExtraNoncePlaceholder.Length,
        "/CoiniumServ/")
    }
};

// transaction outputs
Outputs = new Outputs(daemonClient, poolConfig.Coin);

double blockReward = BlockTemplate.Coinbasevalue; // the amount
rewarded by the block.
// generate output transactions for recipients (set in config).
foreach (var pair in poolConfig.Rewards)
{
    var amount = blockReward * pair.Value / 100;
    // calculate the amount he recieves based on the percent of
his shares.
    blockReward -= amount;

    Outputs.AddRecipient(pair.Key, amount);
}
// send the remaining coins to pool's central wallet.
Outputs.AddPoolWallet(poolConfig.Wallet.Adress, blockReward);
}

```

Текст программного модуля «Диапазон чисел»

```

public class Range : IEnumerable<int>
{
    private readonly int _start;
    private int _stop;
    private int _step = 1;

    public Range(int start)
    {
        _start = _stop = start;
    }

    public static Range From(int startRange)
    {
        return new Range(startRange);
    }

    public Range To(int endRange)
    {
        _stop = endRange;
        return this;
    }

    public Range WithStepSize(int step)
    {
        _step = step;
        return this;
    }

    public IEnumerator<int> GetEnumerator()
    {
        for (var i = _start; _step > 0 ? i < _stop : i > _stop; i +=
_step)
        {

```

```

        yield return i;
    }
}

IEnumerator IEnumerable.GetEnumerator()
{
    return GetEnumerator();
}
}

```

Текст программного модуля «Майнинг пул»:

```

/// <summary>
/// Contains pool services and server.
/// </summary>
public class Pool : IPool
{
    public bool Initialized { get; private set; }

    public double Hashrate { get; private set; }

    public Dictionary<string, double> RoundShares { get; private
set; }

    public IPoolConfig Config { get; private set; }

    public IHashAlgorithm HashAlgorithm { get; private set; }

    public IMinerManager MinerManager { get; private set; }

    public INetworkInfo NetworkInfo { get; private set; }

    public IBlockRepository BlockRepository { get; private set; }

    public IPaymentRepository PaymentRepository { get; private set;
}

    public IDaemonClient Daemon { get; private set; }

    public IAccountManager AccountManager { get; private set; }

    public string ServiceResponse { get; private set; }

    private readonly IObjectFactory _objectFactory;

    private IJobManager _jobManager;

    private IShareManager _shareManager;

    private IBanManager _banningManager;

    private IStorageLayer _storage;

    private readonly IConfigManager _configManager;

    private Dictionary<IMiningServer, IRpcService> _servers;

    private double _shareMultiplier; // share multiplier to be used
in hashrate calculation.

    private readonly ILogger _logger;

```

```

    /// <summary>
    /// Instance id of the pool.
    /// </summary>
    public UInt32 InstanceId { get; private set; }

    /// <summary>
    /// Initializes a new instance of the <see cref="Pool" /> class.
    /// </summary>
    /// <param name="poolConfig"></param>
    /// <param name="configManager"></param>
    /// <param name="objectFactory"></param>
    public Pool(IPoolConfig poolConfig, IConfigManager config-
Manager, IObjectFactory objectFactory)
    {
        Initialized = false; // mark the pool as un-initiliazed un-
til all services are up and running.

        // ensure dependencies are supplied.
        Enforce.ArgumentNotNull(() => poolConfig);
        Enforce.ArgumentNotNull(() => configManager);
        Enforce.ArgumentNotNull(() => objectFactory);

        _configManager = configManager;
        _objectFactory = objectFactory;
        Config = poolConfig;

        _logger = Log.ForContext<Pool>().ForContext("Component",
poolConfig.Coin.Name);
    }

    public void Initialize()
    {
        if (Initialized)
            return;

        try
        {
            if (!Config.Valid) // make sure we have valid configura-
tion.
            {
                _logger.Error("Can't start pool as configuration is
not valid.");
                return;
            }

            GenerateInstanceId(); // generate unique instance id for
the pool.

            if (!InitHashAlgorithm()) // init the hash algorithm re-
quired by the coin.
                return;

            if (!InitDaemonClient()) // init the coin daemon client.
                return;

            if (!InitStorage()) // init storage support.
                return;

            if (!InitCoreServices()) // init core services.
                return;

```

```

        if (!InitStatisticsServices()) // init statistics ser-
vices.
            return;

        if (!InitNetworkServers()) // init network servers.
            return;

        Initialized = true;
    }
    catch (Exception e)
    {
        _logger.Error("Pool initialization failed; {0:l}", e);
        Initialized = false;
    }
}

private bool InitHashAlgorithm()
{
    try
    {
        HashAlgorithm = _objectFactory.GetHashAlgorithm(Config.Coin);
        _shareMultiplier = Math.Pow(2, 32) / HashAlgorithm.Multiplier; // will be used in hashrate calculation.
        return true;
    }
    catch (TinyIoCResolutionException)
    {
        _logger.Error("Unknown hash algorithm: {0:l}, pool initialization failed", Config.Coin.Algorithm);
        return false;
    }
}

private bool InitDaemonClient()
{
    if (Config.Daemon == null || Config.Daemon.Valid == false)
    {
        _logger.Error("Coin daemon configuration is not valid!");
        return false;
    }

    Daemon = _objectFactory.GetDaemonClient(Config.Daemon, Config.Coin);
    return true;
}

private bool InitStorage()
{
    // load the providers for the current storage layer.
    var providers =
        Config.Storage.Layer.Providers.Select(
            providerConfig =>
                _objectFactory.GetStorageProvider(
                    providerConfig is IMysqlProviderConfig ?
StorageProviders.MySql : StorageProviders.Redis,
                    Config, providerConfig)).ToList();

    // start the migration manager if needed
    if (Config.Storage.Layer is HybridStorageConfig)

```



```

        _objectFactory.GetMigrationManager((IMySqlProvider)pro-
viders.First(p => p is MySqlProvider), Config); // run migration manager.

        // load the storage layer.
        if (Config.Storage.Layer is HybridStorageConfig)
            _storage = _objectFactory.GetStorageLayer(StorageLay-
ers.Hybrid, providers, Daemon, Config);

            else if (Config.Storage.Layer is MposStorageConfig)
                _storage = _objectFactory.GetStorageLayer(StorageLay-
ers.Mpos, providers, Daemon, Config);

            else if (Config.Storage.Layer is NullStorageConfig)
                _storage = _objectFactory.GetStorageLayer(StorageLay-
ers.Empty, providers, Daemon, Config);

        return true;
    }

    private bool InitCoreServices()
    {
        AccountManager = _objectFactory.GetAccountManager(_storage,
Config);
        MinerManager = _objectFactory.GetMinerManager(Config, _stor-
age, AccountManager);

        var jobTracker = _objectFactory.GetJobTracker(Config);
        _shareManager = _objectFactory.GetShareManager(Config, Dae-
mon, jobTracker, _storage);
        _objectFactory.GetVardiffManager(Config, _shareManager);
        _banningManager = _objectFactory.GetBanManager(Config,
_shareManager);
        _jobManager = _objectFactory.GetJobManager(Config, Daemon,
jobTracker, _shareManager, MinerManager, HashAlgorithm);
        _jobManager.Initialize(InstanceId);

        var blockProcessor = _objectFactory.GetBlockProcessor(Con-
fig, Daemon, _storage);
        var blockAccounter = _objectFactory.GetBlockAccounter(Con-
fig, _storage, AccountManager);
        var paymentProcessor = _objectFactory.GetPaymentProces-
sor(Config, _storage, Daemon, AccountManager);
        _objectFactory.GetPaymentManager(Config, blockProcessor,
blockAccounter, paymentProcessor);

        return true;
    }

    private bool InitStatisticsServices()
    {
        NetworkInfo = _objectFactory.GetNetworkInfo(Daemon, Ha-
shAlgorithm, Config);
        BlockRepository = _objectFactory.GetBlockRepository(_stor-
age);
        PaymentRepository = _objectFactory.GetPaymentReposi-
tory(_storage);

        return true;
    }

    private bool InitNetworkServers()
    {

```

```

        _servers = new Dictionary<IMiningServer, IRpcService>();

        if (Config.Stratum != null && Config.Stratum.Enabled)
        {
            var stratumServer = _objectFactory.GetMiningServer("Stratum", Config, this, MinerManager, _jobManager, _banningManager);
            var stratumService = _objectFactory.GetMiningService("Stratum", Config, _shareManager, Daemon);
            stratumServer.Initialize(Config.Stratum);

            _servers.Add(stratumServer, stratumService);
        }

        if (Config.Getwork != null && Config.Getwork.Enabled)
        {
            var getworkServer = _objectFactory.GetMiningServer("Getwork", Config, this, MinerManager, _jobManager, _banningManager);
            var getworkService = _objectFactory.GetMiningService("Getwork", Config, _shareManager, Daemon);

            getworkServer.Initialize(Config.Getwork);

            _servers.Add(getworkServer, getworkService);
        }

        foreach (var server in _servers)
        {
            server.Key.Start();
        }

        if(_servers.Count == 0 )
            _logger.Error("No connected servers to network!");

        return true;
    }

    /// <summary>
    /// Generates an instance Id for the pool that is cryptographically random.
    /// </summary>
    private void GenerateInstanceId()
    {
        var rndGenerator = RandomNumberGenerator.Create(); // cryptographically random generator.
        var randomBytes = new byte[4];
        rndGenerator.GetNonZeroBytes(randomBytes); // create cryptographically random array of bytes.
        InstanceId = BitConverter.ToUInt32(randomBytes, 0); // convert them to instance Id.
        _logger.Debug("Generated cryptographically random instance Id: {0}", InstanceId);
    }

    public void Recache()
    {
        if (!Initialized)
            return;

        BlockRepository.Recache(); // recache the blocks.
        NetworkInfo.Recache(); // let network statistics recache.
        CalculateHashrate(); // calculate the pool hashrate.
    }

```

```

        RoundShares = _storage.GetCurrentShares(); // recache current round.

        // cache the json-service response
        ServiceResponse = JsonConvert.SerializeObject(this, Formatting.Indented, new JsonSerializerSettings { ReferenceLoopHandling = ReferenceLoopHandling.Ignore });
    }

    private void CalculateHashrate()
    {
        // read hashrate stats.
        var windowTime = TimeHelpers.NowInUnixTimestamp() - _configManager.StatisticsConfig.HashrateWindow;
        _storage.DeleteExpiredHashrateData(windowTime);
        var hashrates = _storage.GetHashrateData(windowTime);

        double total = hashrates.Sum(pair => pair.Value);
        Hashrate = Convert.ToUInt64(_shareMultiplier * total / _configManager.StatisticsConfig.HashrateWindow);
    }
}

public class PoolManager : IPoolManager
{
    public int Count { get { return _storage.Count; } }

    public string ServiceResponse { get; private set; }

    private readonly IList<IPool> _storage;

    private readonly ILogger _logger;

    public PoolManager(IObjectFactory objectFactory , IConfigManager configManager)
    {
        _logger = Log.ForContext<PoolManager>();
        _storage = new List<IPool>(); // initialize the pool storage.

        // loop through all enabled pool configurations.
        foreach (var config in configManager.PoolConfigs)
        {
            var pool = objectFactory.GetPool(config); // create pool for the given configuration.

            if(pool.Config.Enabled) // make sure pool was successfully initialized.
            {
                _storage.Add(pool); // add it to storage.
            }

            Run(); // run the initialized pools.
        }

        private void Run()
        {
            // run the initialized pools
            foreach (var pool in _storage)
            {
                // var t = new Thread(pool.Initialize);
                // t.Start();
                pool.Initialize();
            }
        }
    }
}

```

```

    }

    public IQueryable<IPool> SearchFor(Expression<Func<IPool, bool>>
predicate)
    {
        return _storage.AsQueryable().Where(predicate);
    }

    public IEnumerable<IPool> GetAll()
    {
        return _storage;
    }

    public IQueryable<IPool> GetAllAsQueryable()
    {
        return _storage.AsQueryable();
    }

    public IReadOnlyCollection<IPool> GetAllAsReadOnly()
    {
        return new ReadOnlyCollection<IPool>(_storage);
    }

    public void Recache()
    {
        try
        {
            foreach (var pool in _storage) // recache per-pool stats
            {
                pool.Recache();
            }

            // cache the json-service response
            var cache = _storage.ToDictionary(pool => pool.Con-
fig.Coin.Symbol.ToLower());
            ServiceResponse = JsonConvert.SerializeObject(cache, Format-
ting.Indented, new JsonSerializerSettings {ReferenceLoopHandling = Refer-
enceLoopHandling.Ignore});
        }
        catch (Exception e)
        {
            _logger.Error("Error recaching statistics; {0:1}", e.Mes-
sage);
        }
    }

    public IPool Get(string symbol)
    {
        return _storage.FirstOrDefault(p => p.Config.Coin.Sym-
bol.Equals(symbol, StringComparison.OrdinalIgnoreCase));
    }

    public IEnumerator<IPool> GetEnumerator()
    {
        return _storage.GetEnumerator();
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}

```