Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

|  |  |
| --- | --- |
|  | Номер зачетной книжки \_\_\_\_\_4505008\_\_\_\_\_ |
|  | Преддипломная практика зачтена с оценкой  \_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)  (цифрой) (прописью) |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись руководителя практики от БГУИР)  \_\_\_\_.\_\_\_\_\_.2018 |

**ОТЧЕТ**

**по преддипломной практике**

**ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ДОБЫЧИ КРИПТОВАЛЮТЫ**

Место прохождения практики: ООО «ФП Трэйд», г. Минск

Сроки прохождения практики: с 23.03.2018 по 19.04.2018

|  |  |  |
| --- | --- | --- |
| Руководитель практики от предприятия:  \_\_\_\_\_\_\_\_\_\_\_ Р.Х.Фомин  (подпись руководителя)  М.П. |  | Студент группы 351004  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Борикова Е.В. (подпись студента)  Руководитель практики от БГУИР  Трус В.В. – ассисцент  кафедры ПОИТ |

Минск 2018

СОДЕРЖАНИЕ

[Введение 3](#_Toc513811246)

[1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 4](#_Toc513811247)

[1.1 Основные направления развития 4](#_Toc513811248)

[1.2 Обзор существующих аналогов 10](#_Toc513811252)

[1.3 Постановка задачи 14](#_Toc513811258)

[2 Моделирование предметной области 16](#_Toc513811259)

[2.1 Разработка функциональной модели предметной области 16](#_Toc513811260)

[2.2 Спецификация функциональных требований 17](#_Toc513811261)

[2.3 Используемые технологии 17](#_Toc513811262)

[3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 25](#_Toc513811266)

[3.1 Проектирование архитектуры программного средства 25](#_Toc513811267)

[Заключение 26](#_Toc513811268)

[Список использованных источников 27](#_Toc513811269)

Введение

В настоящее время слова: «криптовалюта», «блокчейн», «майнинг» у всех на слуху и существуют большой спрос на программные продукты, охватывающие эти понятия.

Такое программное средство позволяет самим добывать криптовалюту, совершать вывод электронных денег по рыночному курсу, объединяться участникам для распределения вычислительной нагрузки, что приведет к ускоренному процессу добычи криптовалют.

На информационном рынке существует достаточно большой выбор аналогов, но довериться таким приложениям очень сложно. 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.

Таким образом можно выделить ряд существующих проблем:

* небезопасность данных;
* взимание большой комиссии за пользование программным продуктом;
* получение вознаграждения.

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

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

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

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

В основу проектирования были положены следующие принципы:

* DRY;
* SOLID;
* YAGNI;
* KISS.

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

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

# Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству

## 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.

### Майнинг пул

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

Однако к майнинг пулу может быть подключено несколько участников, следовательно, награда будет делиться между ними в зависимости от потраченной мощности ПК.

Чтобы найти нужный хеш, нужно методом перебора искать подходящий хеш. Цель майнинг пула ускорить процесс перебора, путем распределения вычислительной нагрузки, 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 специальное число. Специальное число – изменяющееся майнинг пулом число для подбора нужного значения. Майнинг пул распределяет диапазоны специального числа в зависимости от отдаваемой мощности клиента. Схематично это можно представить на рисунке 1.5

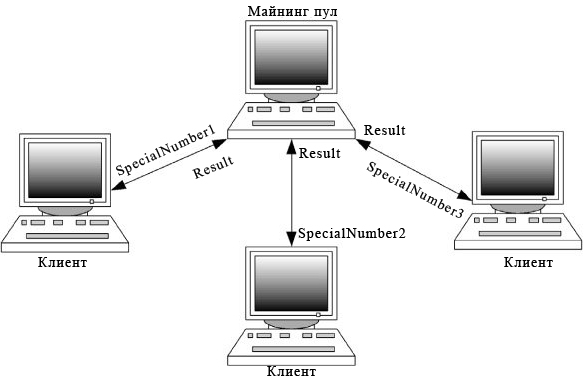


Рисунок 1.5 – Майнинг пул

### Майнинг

Процесс добычи любой криптовалюты называется майнингом. Майнинг нельзя сравнить с обычной печатью денег, потому что в него заложены определенные функции. Сам термин майнинг происходит от английского слова Mine (добыча). Если говорить сложными словами, то этот процесс представляет собой деятельность по поддержанию работы сети путем закрытия и создания блоков в «blockchain» с использованием вычислительных мощностей. Участник (майнер) следит за транзакциями и операциями, создают блоки, использует мощности персональных компьютеров для выполнения специальных вычислений по поиску цифровой подписи (хеша), которая 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 добыча криптовалюты с использованием мощностей специального оборудования. Майнеры получают вознаграждение, так как их деятельность обеспечивает функционирование и целостность всей системы. В этом и заключается основная задача майнинга.

На начальных этапах добывать крипотовалюту 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:

* **Майнинг на процессорах компьютеров (CPU)** – неэффективный способ добычи криптовалюты. Актуально у кого есть доступ к большому количеству компьютеров и бесплатному электричеству.
* **Майнинг на видеокартах (GPU)** актуален для большинства криптомонет, включая эфир, dash и другие. Эффективен при использовании мощных видеокарт.
* **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.
* **Фермы** – установка, объединяющая в себе мощные видеокарты (GPU). Подключается к одному или нескольким компьютерам. Показывает высокую эффективность, при этом оборудование реально продать на вторичном рынке. Увеличение количества майнеров повысило спрос и, как следствие, цены на карты.
* **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.
* **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.
* **Майнинг на телефонах и ноутбуках** – даже самые мощные модели показывают минимальную эффективность. Зарабатывать на таком способе майнинга не эффективно, также как и с ноутбуками.
* **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.
* **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.

### Организация сети между участниками

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

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

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

Рассмотрим два типа архитектур организации сети: технология P2P и "клиент-сервер". Архитектура типа «клиент-сервер», представлена на рисунке 1.3, используется в приложениях для взаимодействия 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 – Архитектура 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 мощи 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 удается улучшать 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.

## 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.

### CG MINER

Консольный клиент для майнинга CG MINER (рис 1.3). Главное ее преимущество – высокая стабильность и эффективная работа в фоновом режиме. Иными словами, вам не нужно постоянно следить за работой программы – запустили вычисления и можете заниматься своими делами.

 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.

Рис. 1.3 – Внешний вид CGMINER



### 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 1.4.

Достоинства:

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

Недостатки:

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



Рис. 1.4 – Внешний вид GUI Miner

### 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 1.5.

Достоинства:

* добывать можно около 14 видов криптовалют;
* удобный графический интерфейс;
* кроссплатформенное ПО;
* смарт-режим.

Недостатки:

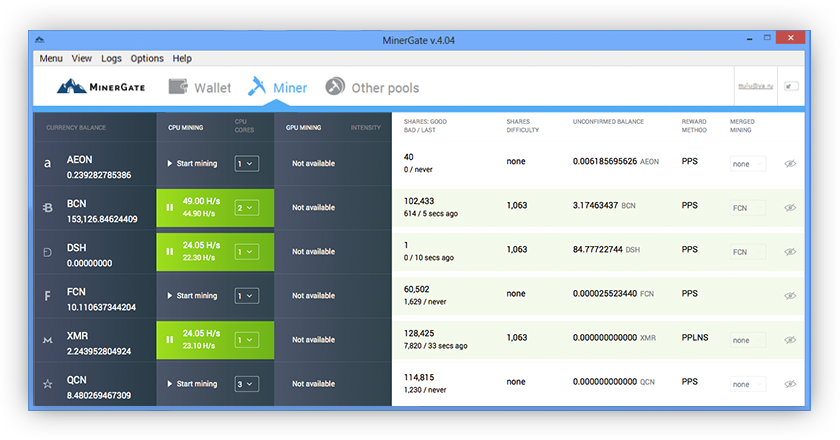
* высокая цена на ПО;
* высокий курс конвертации валют;
* ПС доступно только на английском языке.

Рис. 1.5 – Внешний вид Miner Gate

### 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 1.5.

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.

Рис. 1.6 - Внешний вид Ufasoft Miner

### 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 1.7.

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.



Рисунок 1.7 - Внешний вид Nice Hash Miner

## Постановка задачи

В результате анализа вышеприведённых аналогов можно выделить ряд недостатков, характерных для большинства из них: неудобный интерфейс, отсутствие возможности настройки интерфейса, отсутствия необходимой документации, конвертация валют т.д. Данные позиции играют важную роль в ПС данной предметной области.

Но главной проблемой 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 вовсе.

Поэтому целью проекта является разработка программного средства, которое позволит обеспечить оптимальный уровень управления добычи криптовалют.

Проектируемое средство должно отвечать следующим функциональным требованиям:

* обладать удобным пользовательским интерфейсом;
* иметь встроенную документацию;
* иметь возможность настройки интерфейса;
* дать пользователю возможность регулировать процесс добычи криптовалют;
* поддержка многих пулов и криптовалют для их добычи;
* кроссплатформенность;

# Моделирование предметной области

## Разработка функциональной модели предметной области

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

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

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

Майнер является пользователем системы. Для него открыт доступ ко всем основным функциям, а именно: добыча криптовалюты, конвертация монет в другие валюты, получать награду за добытые монеты, просмотр/редактирование информации.

Майнинг пул также является пользователем системы, который распределяет вычислительную нагрузку между участниками, распределяет прибыль, подключить к сети, добываемой криптовалюты.

Администратор является пользователем с особыми полномочиями, ему доступны функции конфигурирования самого сервера.

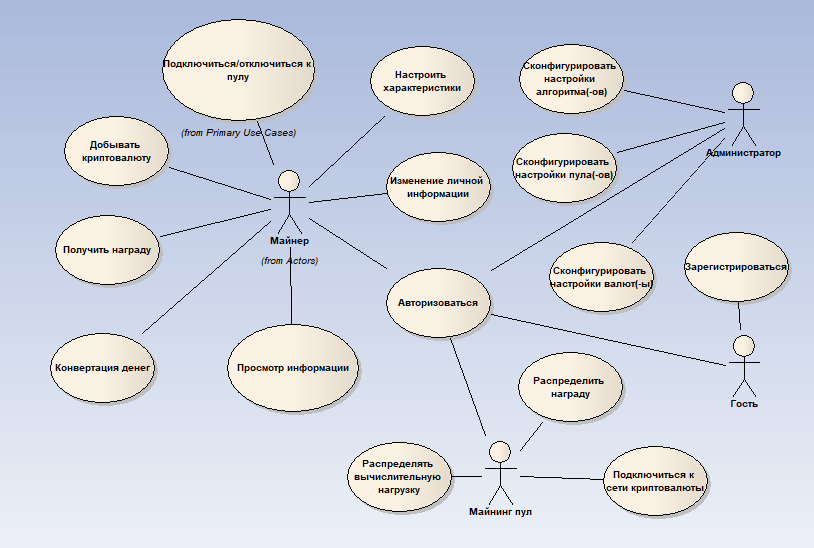


Рисунок 2.1 – Диаграмма вариантов использования

## Используемые технологии

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

– Разрабатываемое ПО должно запускаться как веб-сайт;

– Среди различных платформ разработки имеющийся программист лучше всего знаком с разработкой на платформе ASP .NET;

– Дальнейшей поддержкой проекта, возможно, будут заниматься разработчики, не принимавшие участие в выпуске первой версии;

– Имеющийся разработчик имеет опыт работы с объекто-ориентированными и с функциональными языками программирования.

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

– платформа разработки ASP .NET;

– языки программирования C# и Nancy framework (для клиентской части).

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

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

### Программная платформа ASP.Net

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

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

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

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

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

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

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.

### Язык программирования С# и платформа .Net Framework

Язык программирования C# и среду программирования .NET Framework можно назвать одными 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 мнению Microsoft, распределенные приложения, в которых обработка распределяется между клиентом и сервером, являются шагом вперед. Однако важно понимать, что C# 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 C# и среда .NET привели к революционным изменениям в способе написания разработчиками программ и сделали программирование приложений для Windows гораздо более простым, чем когда-либо.

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

* C# спроектирован и разработан специально для применения с Microsoft .NET Framework (развитой платформой разработки, развертывания и выполнения распределенных приложений).
* C# — 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 C# — это полноценный язык программирования. Хотя он и предназначен для генерации кода, выполняемого в среде .NET, сам по себе он не является частью .NET. Существует ряд средств, которые поддерживаются .NET, но не поддерживаются C#, и, возможно, вас удивит, что есть также средства, поддерживаемые C# и не поддерживаемые .NET (например, некоторые случаи перегрузки операций). Однако поскольку язык C# предназначен для применения на платформе .NET, важно иметь представление о .NET Framework, чтобы эффективно разрабатывать приложения на C#.

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

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

Базовые функциональные возможности платформы .NET включают в себя:

* 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.
* 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).
* 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.
* 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.
* 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.
* 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.

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.

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 СОМ для Microsoft Office.

### Библиотека Nancy

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

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

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

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

* 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;
* 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.
* 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 (рис. 2.2). С помощью Nancy API-интерфейсы должны помогают добраться в нужный модуль к нужному методы.

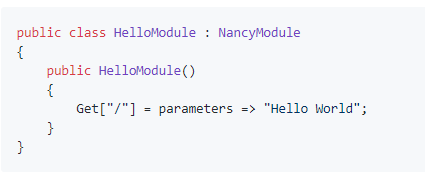


Рисунок 2.2 – пример использования библиотеки Nancy

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) параметр см. рисунок 2.3.

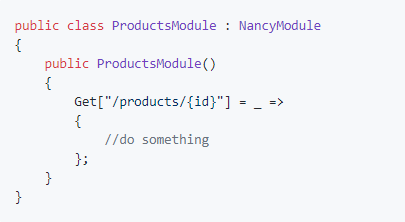


Рисунок 2.3 – составление маршрута в Nancy

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 озночает 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 выражения лямбда (рис 2.4).

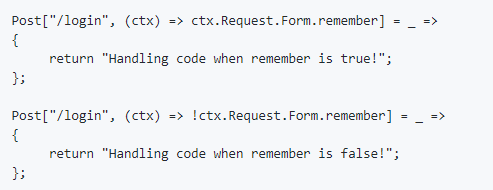


Рисунок 2.4 – Демонстрация необязательного условия в маршруте

## Разработка информационной модели базы данных

### Таблицы, связанные с аккаунтом

Таблица 2.3 – Атрибуты таблицы «Title»

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| Id | Идентификатор аккаунта |
| Username | Имя пользователя |
| Address | Адрес пользователя |

### Таблицы, связанные с BanConfig

Таблица 2.3 – Атрибуты таблицы «BanConfig»

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| Id | Идентификатор запрета |
| Enabled | Статус включенности запрета |
| Duration | Длительность запрета |
| InvalidPercent | Процент неправильных хешей, чтобы вызвать запрет |
| CheckThreshold | Количество сгенерированных хешей, перед тем, как майнер будет рассмотрен на блокировкку |
| PurgeInterval | Интервал времени в секундах, через которое проверяется истекло ли время блокировки, чтобы снять ее |

## Спецификация функциональных требований

Была составлена функциональная спецификация требований к программному средству добычи криптовалют.

Основными функциями ПС являются:

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

Нефункциональные требования для разрабатываемого программного средства:

* работа в современных браузерах (Chrome 60+, Firefox 47+, Opera 50+);
  + поддержка локализации;
  + интуитивно понятный интерфейс, который может быть самостоятельно освоен широким кругом пользователей.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## Проектирование архитектуры программного средства

Архитектура программного средства состоит из трёх основных компонентов:

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

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 состоит из двух архитектур: архитектура типа «Клиент-Сервер» и «Одноранговая сеть». Первый тип архитектуры предназначен для обеспечения взаимодействий между майнинг пулом и клиентами, которые к нему подключились, а второй тип архитектуры, где «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 была выбрана диаграмма развёртывания, представленная на рисунке 3.1.

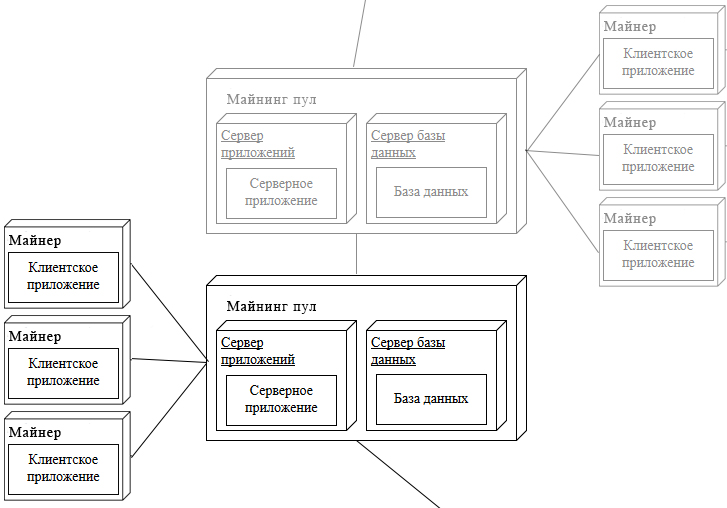


Рисунок 3.1 – Диаграмма развёртывания

### Организация сети между участниками. Программный модуль SocketServer.cs

Для обеспечения обмена данными между участниками сети существуют два протокала 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 произошел сбой или данные искажены, то запрашивает отправку пакета еще раз, также гарантировано, что пакеты придут в том же порядке, в котором и были отправлены.

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

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

* AddressFamily: возвращает все адреса, используемые сокетом;
* Available: возвращает объем данных, которые доступны для чтения;
* Connected: возвращает true, если сокет подключен к удаленному хосту;
* LocalEndPoint: возвращает локальную точку, по которой запущен сокет и по которой он принимает данные;
* ProtocolType: возвращает одно из значений перечисления ProtocolType, представляющее используемый сокетом протокол;
* RemoteEndPoint: возвращает адрес удаленного хоста, к которому подключен сокет;
* SocketType: возвращает тип сокета.

При работе с сокетами я опиралась на методы класса Socket:

* Accept(): создает новый объект Socket для обработки входящего подключения;
* Bind(): связывает объект Socket с локальной конечной точкой;
* Close(): закрывает сокет;
* Connect(): устанавливает соединение с удаленным хостом;
* Listen(): начинает прослушивание входящих запросов;
* Poll(): определяет состояние сокета;
* Receive(): получает данные;
* Send(): отправляет данные;
* Shutdown(): блокирует на сокете прием и отправку данных.

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

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

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

### Протокол Stratum. Программные модули StratumServer.cs, StratumMiner.cs и StratumService.cs

Протокол Stratum представляет собой линейный протокол, использующий простой TCP-сокет, с полезной нагрузкой, закодированной как сообщения JSON. Клиент просто открывает TCP-сокет и пишет запросы на сервер в виде сообщений JSON, завершенных символом новой строки. Каждая полученная клиентом строка снова является действительным фрагментом JSON, содержащим ответ.

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

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

Программный модуль StratumServer.cs вызывается через WebServer метод Start(), в котором, как мы можем увидеть, устанавливается соединение через порт. Также в данном модуле реализованы такие события как:

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

### Разработка программного модуля MinerManager.cs

Заключение

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

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

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

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

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

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

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

Список использованных источников

x

|  |  |
| --- | --- |
| 1. | https://prostocoin.com/blog/what-is-mining |
| 2. | https://learn.javascript.ru/intro  https://metanit.com/sharp/mvc5/1.2.php  http://mtblog.mtbank.by/chto-takoe-kriptovalyuta-i-kak-ee-zarabotat-razvernutyj-putevoditel-v-voprosah-i-otvetah/ - криптовалюта 1 раздел  https://golos.io/ru--golos/@aleco/prosto-i-dostupno-o-blockchain-chto-eto-i-kak-rabotaet - блокчейн 1 раздел  https://habrahabr.ru/post/98323/ - шифрование 1 раздел |
| 3. | Radeck K. C# and Java: Comparing Programming Languages [Электронный ресурс] // MSDN: [сайт]. URL: http:/​/​msdn.microsoft.com/​en-us/​library/​ms836794.aspx |
| 4. | Kurniawan B. Comparing C# and Java [Электронный ресурс] // O’Reilly Media: [сайт]. URL: http:/​/​www.windowsdevcenter.com/​lpt/​a/​889 |
| 5. | Chandra S.S., Chandra K. A comparison of Java and C# // Journal of Computing Sciences in Colleges, No. 20, 2005. pp. 238-254. |
| 6. | Rowe G.W. From Java to C#. Addison Wesley, 2004. 204-206 pp. |
| 7. | Johnson M. C#: A language alternative or just J--?, Part 2 [Электронный ресурс] // JavaWorld: [сайт]. [2000]. URL: http:/​/​www.javaworld.com/​cgi-bin/​mailto/​x\_java.cgi?pagetosend=/​export/​home/​httpd/​javaworld/​javaworld/​jw-12-2000/​jw-1221-csharp2.html&pagename=/​javaworld/​jw-12-2000/​jw-1221-csharp2.html&pageurl=http:/​/​www.javaworld.com/​javaworld/​jw-12-2000/​jw-1221-csharp2. |
| 8. | Krikorian R. Contrasting C# and Java Syntax [Электронный ресурс] // O’Reilly Media: [сайт]. URL: http:/​/​www.windowsdevcenter.com/​lpt/​a/​929 |
| 9. | sealed [Электронный ресурс] // Справочник по C#: [сайт]. URL: http:/​/​msdn.microsoft.com/​ru-ru/​library/​88c54tsw.aspx |
| 10. | Generics in C#, Java, and C++ [Электронный ресурс] URL: http:/​/​www.artima.com/​intv/​genericsP.html |
| 11. | Balagurusamy E. Programming in C#: A Primer. Tata McGraw-Hill, 2008. 8 pp. |
| 12. | Johnson M. C#: A language alternative or just J--?, Part 1 [Электронный ресурс] // JavaWorld: [сайт]. [2000]. URL: http:/​/​www.javaworld.com/​cgi-bin/​mailto/​x\_java.cgi?pagetosend=/​export/​home/​httpd/​javaworld/​jw-11-2000/​jw-1122-csharp1.html&pagename=/​jw-11-2000/​jw-1122-csharp1.html&pageurl=http:/​/​www.javaworld.com/​jw-11-2000/​jw-1122-csharp1.html&site=jw\_core |
| 13. | Anders Hejlsberg B.V.B.E. The Trouble with Checked Exceptions [Электронный ресурс] [2003]. URL: http:/​/​www.artima.com/​intv/​handcuffs.html |
| 14. | cs2j [Электронный ресурс] // github.com: [сайт]. URL: https:/​/​github.com/​twiglet/​cs2j |
| 15. | CS2J: The User Guide [Электронный ресурс] URL: http:/​/​www.cs2j.com/​documentation/ |
| 16. | var (справочник по C#) [Электронный ресурс] // MSDN: [сайт]. URL: https:/​/​msdn.microsoft.com/​ru-ru/​library/​bb383973.aspx |

x

|  |  |
| --- | --- |
| 17. | ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. - Введ. 01.01.1992, М: Изд-во стандартов, 1991. |
| 18. | Глухова, Л.А. Основы алгоритмизации и программирования: лаб. практикум для студ. спец. 1-40 01 01 "Программное обеспечение информационных технологий" дневной формы обуч. В 4 ч / Л. А. Глухова, Е. П. Фадеева, Е. Е. Фадеева. - Минск: БГУИР, 2004. |
| 19. | Доманов, А.Т. Стандарт предприятия. Дипломные проекты (работы). Общие требования / А. Т. Доманов, Н. И. Сорока - Минск: БГУИР, 2010 |