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](#_Toc512234029)

[1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 4](#_Toc512234030)

[1.1 Основные направления развития 4](#_Toc512234031)

[1.2 Обзор существующих аналогов 10](#_Toc512234036)

[1.3 Постановка задачи 14](#_Toc512234042)

[2 Моделирование предметной области 15](#_Toc512234043)

[2.1 Разработка функциональной модели предметной области 15](#_Toc512234044)

[2.2 Спецификация функциональных требований 16](#_Toc512234045)

[2.3 Используемые технологии 16](#_Toc512234046)

[Заключение 28](#_Toc512234050)

[Список использованных источников 29](#_Toc512234051)

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.

### Криптовалюта

Криптовалюта – это один из видов цифровой валюты, электронных денег. Но в отличие от традиционных систем, где все данные хранятся на централизованном сервере, криптовалюты децентрализованы. И если обычный сервер в теории кто-то может взломать или вывести из строя, то с криптовалютой такой опасности нет: копии базы транзакций лежат на компьютерах всех участников системы, и они постоянно друг с другом автоматически сверяются по специальным алгоритмам.

Все криптовалюты основаны на криптографии: очень надежных механизмах шифрования. Взломать такую систему практически невозможно. У криптовалют есть одно очень интересное свойство – они не подвержены инфляции, потому что при создании изначально закладывается ограниченное количество «монет»: допечатать их никак нельзя. По поводу «обеспечения» тут все достаточно просто – в некотором смысле криптовалюты обеспечены желающими их создавать и покупать, а также компаниями, которые готовы продавать товары и услуги за соответствующую криптовалюту.

Стоимость той или иной криптовалюты определяется спросом и предложением на рынке. При этом факторов, влияющих на расстановку сил продавцов и покупателей большое количество: объемы эмиссии валюты, технологические аспекты, лежащие в основе того или иного криптоактива, возможность использования криптовалюты для приобретения благ, регулирование различными странами, новостной ажиотаж (зачастую, искусственно созданный) и т.д. Как и любой другой финансовый актив, криптовалюты можно покупать и продавать на специальных биржах, где и определяются те самые курсы криптовалют.

Сегодня в мире насчитывается более 800 видов криптовалют. Самые популярные виды криптовалют:

* **Bitcoin** – первая криптовалюта в истории, текущая капитализация которой составляет сегодня $95,546,108,543;
* **Ethereum** часто называют «цифровым аналогом нефти». Основное отличие его от того же биткоина состоит в том, что эфиры («монетки» Ethereum) можно использовать как «топливо» для исполнения умных контрактов: действий в блокчейне, которые автоматически будут исполнены при достижении определенного условия;
* **Ripple** часто используется для международных расчетов в реальном времени, а также для обмена валют и денежных переводов;
* **Bitcoin Cash** – альтернатива биткоину. Каждая криптовалюта создается по определенному алгоритму. Алгоритм биткоина в определенный момент времени разделился на два направление, одно из которых и получило название Bitcoin Cash;
* **Litecoin** часто называют «цифровым серебром» (в этой системе «цифровое золото» – это Bitcoin). Отличается меньшим временем транзакций, чем биткоин.

### Транзакция. Хеширование и подпись транзакции

**Транзакция** - операция, состоящая в переводе денежных средств с одного счёта на другой; сделка купли-продажи. Каждая транзакция должна быть защищена от злоумышленников, чтобы никто ее не смог изменить или от добавления в реестр операции, которой не было на самом деле. Таким образом, транзакции должны иметь цифровую подпись.

Каждый участник получает закрытый и открытый ключ, каждый из которых выглядит как цепочка битов. Секретный ключ нужно хранить в тайне. Цифровая подпись намного надежнее ручной подписи, так как в создании электронной подписи участвует функция, которая зависит от сообщения и закрытого ключа, но даже малейшее изменение в сообщении полностью перестраивает соответствующую ему подпись. Она выглядит как цепочка битов, чаще всего длинной в 256 бит. Благодаря закрытому ключу подпись может сделать только сам участник. Дополнительной защитой является то, что подпись зависит от сообщения, значит никто не сможет ее скопировать из одного сообщения и вставить в другое. Представим схематичную формулу цифровой подписи:

Sign(Message, private\_key) = Signature

Также необходимо проверить полученную подпись, для этого существует следующая формула, которая возвращает true или false в зависимости от того была ли подпись создана на основе закрытого ключа, который соответствует открытому ключу:

Verify(Message, Signature, public\_key) = true/false

### Технология «Блокчейн» и процесс шифрования

Блокчейн (от английского слова blockchain) – это способ хранения данных или цифровой реестр транзакций, сделок, контрактов. Всего что нуждается в отдельной независимой записи и, при необходимости, в проверке. Главным отличием и неоспоримым преимуществом данного реестра является то, что этот он не хранится в каком-то одном месте. Он распределён среди нескольких сотен и даже тысяч компьютеров во всем мире. Любой Пользователь этой сети может иметь свободный доступ к актуальной версии реестра, что делает его прозрачным абсолютно для всех участников.

Цепочка блоков транзакций — выстроенная по определённым правилам цепочка из формируемых блоков транзакций (рис. 1.1). Впервые термин появился как название распределённой базы данных, реализованной в криптовалюте Bitcoin.

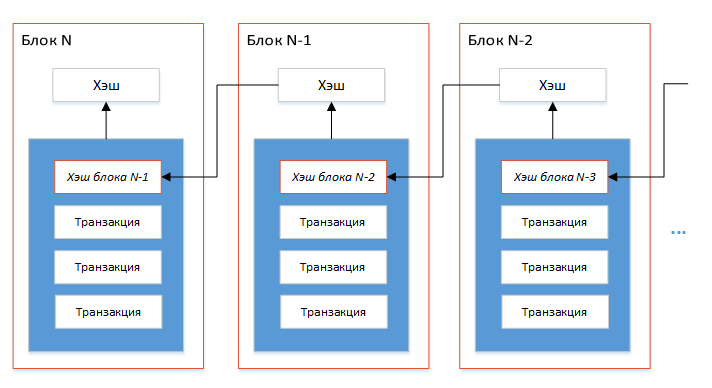


Рис. 1.1 – Цепочка блокчейнов

Цифровые записи, транзакции объединяются в «блоки», которые потом связываются криптографически и хронологически в «цепочку» с помощью сложных математических алгоритмов. Каждый блок связан с предыдущим и содержит в себе набор записей, например, в криптовалюте Bitcoin в одном блоке содержится около 2400 транзакций. Новые блоки всегда добавляются строго в конец цепочки. Чтобы добавить новый блок в цепочку, необходимо найти нужный хеш, с помощью математических вычислений. Процесс шифрования, известный как хеширование, выполняется большим количеством разных компьютеров.

Шифрование подразделяют на симметричное, ассиметричное и комбинированное.  Рассмотрим, в чем суть каждого из них.

**Симметричное шифрование** отличается наличием ключа – некоторой последовательности чисел, которая используется для шифрования и расшифровывания. При этом, каждая из обменивающихся информацией сторон должна этот ключ знать и хранить в секрете. Огромным плюсом такого подхода является скорость шифрования: ключ, по сути, является достаточно простой и короткой инструкцией, какой символ, когда, и на какой надо заменять. И работает данный ключ в обе стороны (то есть с его помощью можно как заменить все символы на новые, так и вернуть все как было), за что такой способ шифрования и получил название симметричного. Столь же огромным минусом является именно то, что обе стороны, между которыми информация пересылается, должны ключ знать. При этом, стоит нехорошему человеку заполучить ключ, как он тут же прочитает наши защищаемые данные.

**Асимметричное шифрование**. Здесь и у отправителя, и у получателя есть уже два ключа, которые называют открытый и закрытый. Закрытый ключ мы и получатель храним у себя, а открытый мы и получатель можем спокойно передавать кому угодно – наш закрытый, секретный, по нему восстановить нельзя. Итого, мы используем открытый ключ получателя для шифрования, а получатель, в свою очередь, использует свой закрытый ключ для расшифровывания. Схема ассиметричного шифрования приведена на рисунке 1.2.

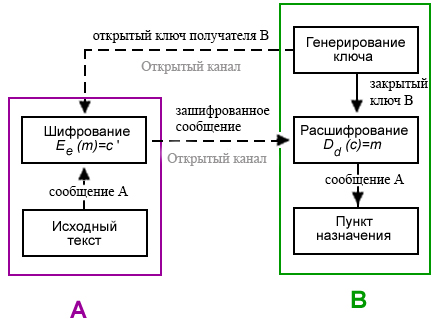


Рис. 1.2 – Схема ассиметричного шифрования

Плюс данного подхода очевиден: мы легко можем начать обмениваться секретной информацией с разными получателями, практически ничем (принимая условие, что наш получатель свой закрытый ключ не потерял/отдал и т.п., то есть не передал его в руки нехорошего человека) не рискуем при передаче информации. Но, без огромного минуса не обойтись. И здесь он в следующем: шифрование и расшифровывание в данном случае идут очень, очень, очень медленно, на два-три порядка медленнее, чем аналогичные операции при симметричном шифровании. Кроме того, ресурсов на это шифрование тратится также значительно больше. Да и сами ключи для данных операций существенно длиннее аналогичных для операций симметричного шифрования, так как требуется максимально обезопасить закрытый ключ от подбора по открытому. А значит, большие объемы информации данным способом шифровать просто невыгодно.

Чтобы решить эту проблему, делать нужно следующее: скомбинировать оба способа. Собственно, так и получаем комбинированное шифрование. Наш большой объем данных кодируем синхронным шифрованием, а чтобы донести ключ до получателя, сам ключ зашифруем по второму способу. Тогда и получим асимметричное медленное шифрование, но объем зашифрованных данных (то есть ключа, на котором зашифрованы большие данные) будет маленьким, а значит расшифровывание пройдет достаточно быстро, и дальше уже в дело вступит более быстрое симметричное шифрование.

В криптовалюте Bitcoin используется асинхронный алгоритм хеширования SHA256.

После нахождения определенным участником нужного хеша, блок присоединяется в цепочку блоков, а участник получает награду за нахождения нового блока. Как только реестр будет обновлён и образован новый блок, он уже больше не может быть изменён. Таким образом подделать его невозможно. К нему можно только добавлять новые записи. Важно учесть то, что реестр обновляется на всех компьютерах в сети одновременно.

### Добыча криптовалюты и участники сети

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» цифровую подпись, 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.

В связи с растущей сложностью добычи криптовалют начали формироваться пулы – сервера, где между участниками распределяется вычислительная нагрузка для добычи криптовалюты. А после нахождения цифровой подписи прибыль делится на всех участников, в зависимости от мощности персонального компьютера.

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, 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 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, исполняющая роль своего рода каркаса для коммуникаций в приложениях P2P. 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.

Технология P2P представляет собой альтернативный подход к организации сетевых коммуникаций. На рисунке 1.3 продемонстрированы отличия, чем P2P отличается от "стандартного" подхода к обеспечению коммуникаций связь типа "клиент-сервер".

Традиционно взаимодействие с приложениями по сети (в том числе Интернет) организуется с использованием архитектуры типа "клиент-сервер". 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. Такой архитектуре присуща проблема с масштабируемостью, что продемонстрировано на рисунке 1.3.

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. Для этого доступна масса решений, таких как продукты, позволяющие создавать веб-фермы или фермы серверов.

## 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:

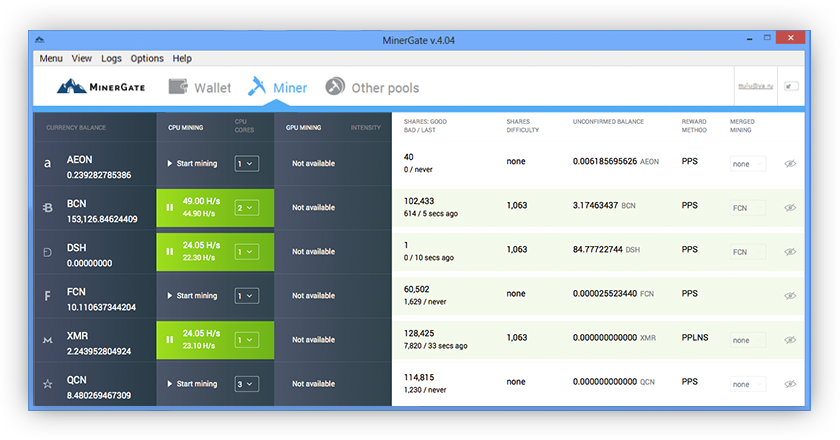
* высокая цена на ПО;
* высокий курс конвертации валют;
* ПС доступно только на английском языке.

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 3 вида актеров: администратор, майнер 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 рядовым пользователем системы. Для него открыт доступ ко всем основным функциям, а именно: добыча криптовалюты, конвертация монет в другие валюты, получать награду за добытые монеты, изменить личную информацию.

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

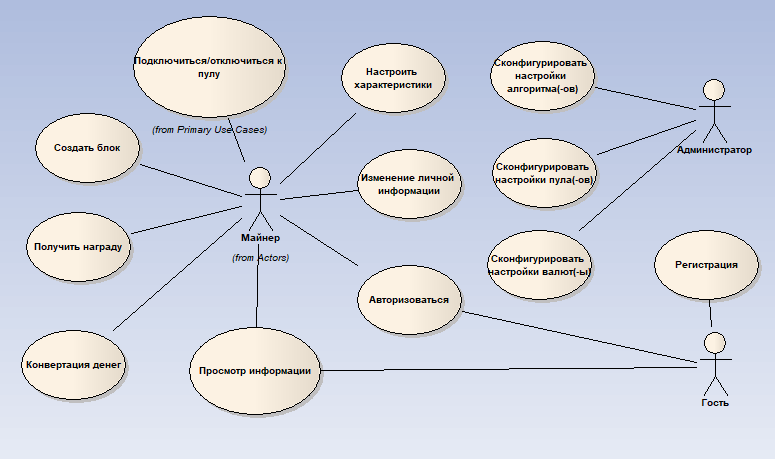


Рисунок 2.1 – Диаграмма вариантов использования

## 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.

## 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 2015: 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 JavaScript (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. Язык JavaScript больше подходит для создания высокоуровневого дизайна приложения (динамическое отображение элементов, гибкий интерфейс), язык С# — для реализации логики приложения, функций и методов, иерархия дерева классов, применения паттернов проектирования. В разрабатываемом программном продукте JavaScript используется для предоставления удобного программного интерфейса, C# — для проектирования и реализации вычислительной логики. Далее проводится характеристика используемых технологий и языков программирования более подробно.

### Программная платформа ASP.Net

Платформа ASP.NET MVC представляет собой фреймворк для создания сайтов и веб-приложений с помощью реализации паттерна MVC. Концепция паттерна (шаблона) MVC (model - view - controller) предполагает разделение приложения на три компонента:

Контроллер (controller) представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления.

Представление (view) определяет представление модели. Как правило, представление получает состояние и данные для отображения непосредственно от модели.

Модель (model) хранит все данные, информацию состояния и логику приложения. Она не знает о существовании представления и контроллера, хотя и предоставляет интерфейс для получения/изменения состояния, а также может отправлять оповещения об изменениях состояния наблюдателям.

Общую схему взаимодействия этих компонентов можно представить следующим образом на рисунке 2.1:

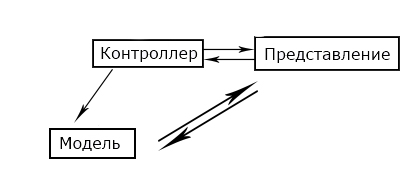


Рис. 2.1 - Общая схема взаимодействия MVC

В этой схеме модель является независимым компонентом - любые изменения контроллера или представления не затрагивают модель. Контроллер и представление являются относительно независимыми компонентами, и нередко их можно изменять независимо друг от друга.

Реализуется концепция разделение ответственности, в связи с чем легче построить работу над отдельными компонентами. Кроме того, вследствие этого приложение обладает лучшей тестируемостью. И если нам, допустим, важна визуальная часть, то мы можем тестировать представление независимо от контроллера. Либо мы можем сосредоточиться на серверной части и тестировать контроллер.

Конкретные реализации и определения данного паттерна могут отличаться, но в силу своей гибкости и простоты он стал очень популярным в последнее время, особенно в сфере веб-разработки.

#### Структура проекта MVC 5

Весь этот функционал обеспечивается следующей структурой проекта, которая продемонстрирована на рисунке 2.2:

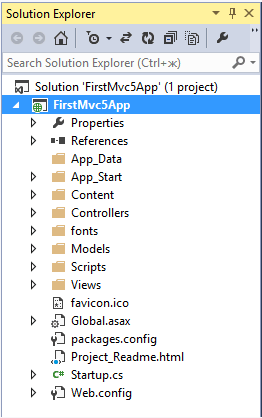


Рис.2.2 – Структура проекта

Рассмотрим, для чего предназначены все эти папки и файлы:

* **App\_Data**: содержит файлы, ресурсы и базы данных, используемые приложением;
* **App\_Start**: хранит ряд статических файлов, которые содержат логику инициализации приложения при запуске;
* **Content**: содержит вспомогательные файлы, которые не включают код на c# или javascript, и которые развертываются вместе с приложением, например, файлы стилей css;
* **Controllers**: содежит файлы классов контроллеров. По умолчанию в эту папку добавляются два контроллера - HomeController и AccountController;
* **fonts**: хранит дополнительные файлы шрифтов, используемых приложением;
* **Models**: содержит файлы моделей. По умолчанию Visual Studio добавляет пару моделей, описывающих учетную запись и служащих для аутентификации пользователя;
* **Scripts**: каталог со скриптами и библиотеками на языке javascript;
* **Views**: здесь хранятся представления. Все представления группируются по папкам, каждая из которых соответствует одному контроллеру. После обработки запроса контроллер отправляет одно из этих представлений клиенту. Также здесь имеется каталог Shared, который содержит общие для всех представления;
* **Global.asax**: файл, запускающийся при старте приложения и выполняющий начальную инициализацию. Как правило, здесь срабатывают методы классов, определенных в папке App\_Start;
* **Startup.cs**: поскольку в приложении MVC 5 используются библиотеки, применяющие спецификацию OWIN, то данный файл организует связь между OWIN и приложением. (OWIN представляет спецификацию, описывающую взаимодействие между компонентами приложения);
* **Web.config**: файл конфигурации приложения;

Конкретная структура каждого отдельного приложения, естественно, будет отличаться, а гибкость MVC позволяет изменять структуру, приспосабливая, ее к своим потребностям. Но описанные выше моменты будут общими для большинства проектов.

#### Механизмы аутентификации

Нам доступен выбор механизма аутентификации при создании проекта (кнопка Change Authentication). По умолчанию установлен тип Individual User Accounts. Если мы нажмем на кнопку Change Authentication, то нам будет доступен выбор из следующих типов аутентификации (Рис. 2.3):

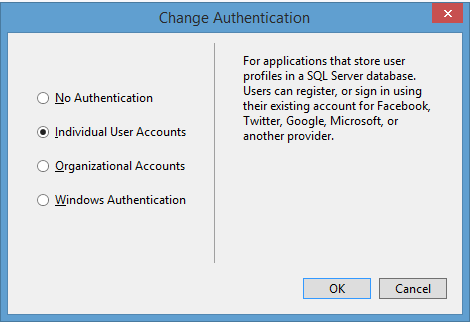


Рис. 2.3 – Выбор механизма аутентификации

* **No Authentication**: приложение не требует аутентификации пользователя;
* **Individual User Accounts**: требуется индивидуальная аутентификация, учетные записи пользователей хранятся в базе данных, кроме того, доступна аутентификация с помощью социальных сетей;
* **Organizational Accounts**: в основном предназначено для корпоративных приложений, которые используют Active Directory или Office 365;
* **Windows Authentication**: для управления учетными записями используется аутентификация Windows, так называемые intranet-приложения;

### Язык программирования С#

C# — объектно-ориентированный язык программирования общего назначения. В языке гармонично сочетаются простота, выразительность и производительность промежуточного кода, получаемого после компиляции. Главным архитектором и идеологом языка с первой версии является Андрес Хейлсберг (создатель Turbo Pascal и архитектор Delphi). Язык C# является платформенно нейтральным, но создавался для хорошей работы с Microsoft .NET [18]. Этот язык соче- тает простой синтаксис, похожий на синтаксис языков C++ и Java, и пол- ную поддержку всех современных объектно-ориентированных концепций и подходов. В качестве ориентира при разработке языка было выбрано безопасное программирование, нацеленное на создание надежного и простого в сопровождении кода [19]. Язык имеет богатую поддержку парадигмы объекто-ориентированного программирования, включающую поддержку инкапсуляции, наследования, абстракции, обработку исключений и полиморфизма. Отличительными чертами C# с точки зрения ОО парадигмы являются:

– Унифицированная система типов. В C# сущность, содержащая данные и методы их обработки, называется типом. В C# все типы, являются пользовательскими типами, или примитивами, такими как число, производны от одного базового класса.

– Классы и интерфейсы. В классической объекто-ориентированной парадигме существуют только классы. В C# дополнительно существуют и другие типы, например, интерфейсы. Интерфейс — это сущность напоминающая классы, но содержащая только определения членов. Конкретная реализация указанных членов интерфейса происходит в типах, реализующих данный интерфейс. В частности, интерфейсы могут быть использованы при необходимости проведения множественного наследования (в отличие от языков C++ и Eiffel, C# не поддерживает множественное наследование классов).

– Свойства, методы и события. В чистой объекто-ориентированной парадигме все функции являются методами. В C# методы являются лишь одной из возможных разновидностей членов типа, в C# типы также могут содержать свойства, события и другие члены. Свойство — это такая разновидность функций, которая инкапсулирует часть состояния объекта. Событие — это разновидность функций, которые реагируют на изменение состояния объекта [18]. В большинстве случаев C# обеспечивает безопасность типов в том смысле, что компилятор контролирует чтобы взаимодействие с экземпляром типа происходило согласно контракту, который он определяют. Например, компилятор C# не скомпилирует код, который обращается со строками, как если бы они были целыми числами. Говоря более точно, C# поддерживает статическую типизацию, в том смысле что большинство ошибок типов обнаруживаются на стадии компиляции. За соблюдение более строгих правил безопасности типов следит исполняющая среда. Статическая типизация позволяет избавиться от широкого круга ошибок, возникающих из-за ошибок типов. Она делает написание и изменение программ более предсказуемыми и надежными, кроме того, статическая типизация позволяет существовать таким средствам как автоматическое дополнение кода и его предсказуемый статический анализ. Еще одним аспектом типизации в C# является её строгость. Строгая типизация означает, что правила типизации в языке очень «сильные». Например, язык не позволяет совершать вызов метода, принимающего целые числа, передавая в него вещественное число [18]. Такие требования спасают от некоторых ошибок. C# полагается на автоматическое управление памятью со стороны исполняющей среды, предоставляя совсем немного средств для управления жизненным циклом объектов. Несмотря на это, в языке все же присутствует поддержка работы с указателями. Данная возможность предусмотрена для случаев, когда критически важна производительность приложения или необходимо обеспечить взаимодействие с неуправляемым кодом [18]. Как уже упоминалось C# не является платформенно зависимым языком. Благодаря усилиям компании Xamarin возможно писать программы на языке C# не только для операционных систем Microsoft, но и ряда других ОС. Существуют инструменты создания приложений на C# для серверных и мобильных платформ: iOS, Android, Linux и других. Создатели языка C# не являются противниками привнесения в язык новых идей и возможностей, в отличии от создателей одного из конкурирующих языков. Каждая новая версия компилятора языка привносит различные полезные возможности, которые отчаются требованиям индустрии. Далее приводится краткий обзор развития языка. Первая версия C# была похожа по своим возможностям на Java 1.4, несколько их расширяя: так, в C# имелись свойства (выглядящие в коде как поля объекта, но на деле вызывающие при обращении к ним методы класса), индексаторы (подобные свойствам, но принимающие параметр как индекс массива), события, делегаты, циклы foreach, структуры, передаваемые по значению, автоматическое преобразование встроенных типов в объекты при необходимости (boxing), атрибуты, встроенные средства взаимодействия с неуправляемым кодом (DLL, COM) и прочее [20]. Версия Microsoft .NET 2.0 привнесла много новых возможностей в сравнении с предыдущей версией, что отразилось и на языках под эту платформу. Проект спецификации C# 2.0 впервые был опубликован Microsoft в октябре 2003 года; в 2004 году выходили бета-версии (проект с кодовым названием Whidbey), C# 2.0 окончательно вышел 7 ноября 2005 года вместе с Visual Studio 2005 и Microsoft .NET 2.0. Ниже перечислены новые возможности в версии 2.0

– Частичные типы (разделение реализации класса более чем на один файл).

– Обобщённые, или параметризованные типы (generics). В отличие от шаблонов C++, они поддерживают некоторые дополнительные возможности и работают на уровне виртуальной машины. Вместе с тем, параметрами обобщённого типа не могут бы

– Анонимные методы, обеспечивающие функциональность замыканий.

– Оператор ??: return obj1 ?? obj2; означает (в нотации C# 1.0) return obj1!=null ? obj1 : obj2;.

– Обнуляемые (nullable) типы-значения (обозначаемые вопросительным знаком, например, int? i = null;), представляющие собой те же самые типы-значения, способные принимать также значение null. Такие типы позволяют улучшить взаимодействие с базами данных через язык SQL.

– Поддержка 64-разрядных вычислений позволяет увеличить адресное пространство и использовать 64-разрядные примитивные типы данных [20]. Третья версия языка имела одно большое нововведение

— Language Integrated Query (LINQ), для реализации которого в языке дополнительно появилось множество дополнительных возможностей. Ниже приведены некоторые из них:

– Ключевые слова select, from, where, позволяющие делать запросы из SQL, XML, коллекций и т. п.

– Инициализацию объекта вместе с его свойствами: Customer c = new Customer(); c.Name = "James"; c.Age=30; можно записать как Customer c = new Customer { Name = "James", Age = 30 };

– Лямбда-выражения: listOfFoo.Where(delegate(Foo x) { return x.size > 10; }); теперь можно записать как listOfFoo.Where(x => x.size > 10);

– Деревья выражений — лямбда-выражения теперь могут быть представлены в виде структуры данных, доступной для обхода во время выполнения, тем самым позволяя транслировать строго типизированные C#- выражения в другие домены (например, выражения SQL).

– Вывод типов локальной переменной: var x = "hello"; вместо string x = "hello"; – Безымянные типы: var x = new { Name = "James"};

– Методы-расширения — добавление метода в существующий класс с помощью ключевого слова this при первом параметре статической функции.

– Автоматические свойства: компилятор сгенерирует закрытое поле для кода вида public string Name { get; private set; } C# 3.0 совместим с C# 2.0 по генерируемому MSIL-коду; улучшения в языке — чисто синтаксические и реализуются на этапе компиляции [20]. Visual Basic .NET 10.0 и C# 4.0 были выпущены в апреле 2010 года, одновременно с выпуском Visual Studio 2010. Новые возможности в версии 4.0:

– Возможность использования позднего связывания.

– Именованные и опциональные параметры.

– Новые возможности COM interop.

– Ковариантность и контрвариантность интерфейсов и делегатов.

– Контракты в коде (Code Contracts) [20].

В C# 5.0 было немного нововведений, но они носят большую практическую ценность. В новой версии появилась упрощенная поддержка выполнения асинхронных функций с помощью двух новых слов — async и await. Ключевым словом async помечаются методы и лямбда-выражения, которые внутри содержат ожидание выполнения асинхронных операций с помощью оператора await, который отвечает за преобразования кода метода во время компиляции.

### Язык программирования JavaScript

Когда создавался язык JavaScript, у него изначально было другое название: «LiveScript». Но тогда был очень популярен язык Java, и маркетологи решили, что схожее название сделает новый язык более популярным. Планировалось, что JavaScript будет эдаким «младшим братом» Java. Однако, история распорядилась по-своему, JavaScript сильно вырос, и сейчас это совершенно независимый язык, со своей спецификацией, которая называется ECMAScript, и к Java не имеет никакого отношения.

JavaScript может выполняться не только в браузере, а где угодно, нужна лишь специальная программа – интерпретатор. Процесс выполнения скрипта называют «интерпретацией».

Для выполнения программ, не важно на каком языке, существуют два способа: «компиляция» и «интерпретация»:

* **Компиляция** – это когда исходный код программы, при помощи специального инструмента, другой программы, которая называется «компилятор», преобразуется в другой язык, как правило – в машинный код. Этот машинный код затем распространяется и запускается. При этом исходный код программы остаётся у разработчика.
* **Интерпретация** – это когда исходный код программы получает другой инструмент, который называют «интерпретатор», и выполняет его «как есть». При этом распространяется именно сам исходный код (скрипт). Этот подход применяется в браузерах для JavaScript.

Во все основные браузеры встроен интерпретатор JavaScript, именно поэтому они могут выполнять скрипты на странице. Но, разумеется, JavaScript можно использовать не только в браузере. Это полноценный язык, программы на котором можно запускать и на сервере, и даже в стиральной машинке, если в ней установлен соответствующий интерпретатор.

Современный JavaScript – это «безопасный» язык программирования общего назначения. Он не предоставляет низкоуровневых средств работы с памятью, процессором, так как изначально был ориентирован на браузеры, в которых это не требуется.

Что же касается остальных возможностей – они зависят от окружения, в котором запущен JavaScript. В браузере JavaScript умеет делать всё, что относится к манипуляции со страницей, взаимодействию с посетителем и, в какой-то мере, с сервером:

* Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.;
* Реагировать на действия посетителя, обрабатывать клики мыши, перемещения курсора, нажатия на клавиатуру и т.п.;
* Посылать запросы на сервер и загружать данные без перезагрузки страницы (эта технология называется "AJAX");
* Получать и устанавливать cookie, запрашивать данные, выводить сообщения;

JavaScript – быстрый и мощный язык, но браузер накладывает на его исполнение некоторые ограничения. Это сделано для безопасности пользователей, чтобы злоумышленник не мог с помощью JavaScript получить личные данные или как-то навредить компьютеру пользователя.

Этих ограничений нет там, где JavaScript используется вне браузера, например, на сервере. Кроме того, современные браузеры предоставляют свои механизмы по установке плагинов и расширений, которые обладают расширенными возможностями, но требуют специальных действий по установке от пользователя. JavaScript не может читать/записывать произвольные файлы на жесткий диск, копировать их или вызывать программы. Он не имеет прямого доступа к операционной системе.

JavaScript, работающий в одной вкладке, не может общаться с другими вкладками и окнами, за исключением случая, когда он сам открыл это окно или несколько вкладок из одного источника (одинаковый домен, порт, протокол).

Есть способы решить эту проблему, но они требуют специального кода на оба документа, которые находятся в разных вкладках или окнах. Без него, из соображений безопасности, залезть из одной вкладки в другую при помощи JavaScript нельзя. Из JavaScript можно легко посылать запросы на сервер, с которого пришла страница. Запрос на другой домен тоже возможен, но менее удобен, т. к. и здесь есть ограничения безопасности.

JavaScript и является самым распространённым средством создания браузерных интерфейсов.

#### HTML 5

HTML 5 – эволюция стандарта HTML, добавляющая новые теги и, что более важно, ряд новых возможностей браузерам.

Вот несколько примеров:

* Чтение/запись файлов на диск (в специальной «песочнице», то есть не любые);
* Встроенная в браузер база данных, которая позволяет хранить данные на компьютере пользователя;
* Многозадачность с одновременным использованием нескольких ядер процессора;
* Проигрывание видео/аудио, без Flash;
* 2D и 3D-рисование с аппаратной поддержкой, как в современных играх;

Многие возможности HTML5 всё ещё в разработке, но браузеры постепенно начинают их поддерживать.

#### ECMAScript 6

Сам язык JavaScript улучшается. Современный стандарт ECMAScript 5 включает в себя новые возможности для разработки, ECMAScript 6 будет шагом вперёд в улучшении синтаксиса языка.

Современные браузеры улучшают свое ядро, чтобы увеличить скорость исполнения JavaScript, исправляют баги и стараются следовать стандартам. Очень важно то, что новые стандарты HTML5 и ECMAScript сохраняют максимальную совместимость с предыдущими версиями. Это позволяет избежать неприятностей с уже существующими приложениями.

#### Языки поверх JavaScript

Синтаксис JavaScript устраивает не всех: одним он кажется слишком свободным, другим – наоборот, слишком ограниченным, третьи хотят добавить в язык дополнительные возможности, которых нет в стандарте. Это нормально, ведь требования и проекты у всех разные.

В последние годы появилось много языков, которые добавляют различные возможности «поверх» JavaScript, а для запуска в браузере – при помощи специальных инструментов «трансляторов» превращаются в обычный JavaScript-код.

Это преобразование происходит автоматически и совершенно прозрачно, при этом неудобств в разработке и отладке практически нет.

При этом разные языки выглядят по-разному и добавляют совершенно разные вещи:

* Язык CoffeeScript – это «синтаксический сахар» поверх JavaScript. Он сосредоточен на большей ясности и краткости кода. Как правило, его особенно любят программисты на Ruby;
* Язык TypeScript сосредоточен на добавлении строгой типизации данных. Он предназначен для упрощения разработки и поддержки больших систем. Его разрабатывает Microsoft;
* Язык Dart интересен тем, что он не только транслируется в JavaScript, как и другие языки, но и имеет свою независимую среду выполнения, которая даёт ему ряд возможностей и доступна для встраивания в приложения (вне браузера). Он разрабатывается компанией Google;

Существуют также трансляторы, которые берут код, использующий возможности будущих стандартов JavaScript, и преобразуют его в более старый вариант, который понимают все браузеры. Например, babeljs. Благодаря этому, мы можем использовать многие возможности будущего уже сегодня.

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 |