Семён Аникеев
начальник Центра компьютерных технологий Института нечётких систем
(г. Москва, Российская Федерация)
(e-mail: pavelkohno@mail.ru)

Павел Кохно
доктор экономических наук, профессор,
директор Института нечётких систем (г. Москва, Российская Федерация)
(e-mail: pavelkohno@mail.ru)

ТЕХНОЛОГИИ ДЕЦЕНТРАЛИЗОВАННОЙ СОЦИАЛЬНОЙ СЕТИ

В статье предлагаются технические, математические и другие решения, способные значительно сократить народнохозяйственные затраты России как социального государства на обеспечение функционирования электронных социальных сетей. Авторы анализируют действующие централизованные социальные сети с учётом существующих недостатков. Предложен методический подход к разработке многоплатформенной децентрализованной социальной сети с использованием межпланетной файловой системы. При этом разработана децентрализованная социальная сеть, смарт-контракт Ethereum, модуль взаимодействия с «межпланетной файловой системой» (IPFS)[1], спроектировано решение для клиентской стороны, база данных, а также настроен локальный сервер. Произведён анализ способов повышения скорости загрузки контента. Для разработки клиентской части децентрализованной социальной сети предложено несколько популярных фреймворков. Исследованы адаптированные модели жизненного цикла, т. е. способы описания этапов разработки программного обеспечения с учетом особенностей конкретного проекта. Учитывая специфику разработки многоплатформенной децентрализованной социальной сети с использованием IPFS, наиболее подходящей моделью жизненного цикла может быть Agile. Для расчёта вычислительной и емкостной сложности разработанной системы предложен алгоритм. При расчёте рассмотрены основные функции компонентов системы.

Ключевые слова: межпланетная файловая система, смарт-контракт Ethereum, децентрализованная социальная сеть, конкурентные решения, пользователи, блокчейн, CID, многоплатформенность, провайдер Metamask, загрузка контента.

DOI: 10.31857/S020736760027685-2

Введение. Социальные сети стали неотъемлемой частью жизни современного человека. Для многих пользователей соцсети стали привычным способом коммуникации и социализации, а для некоторых – даже способом основного заработка [1]. Однако, несмотря на постоянное развитие уже привычных централизованных социальных сетей, существующие решения не лишены недостатков: случаются массовые утечки персональных данных пользователей, централизованные социальные сети подвержены цензуре и кибератакам. Чтобы

[1] Так называемая «межпланетная файловая система» (Interplanetary Filing System, IPFS) — это блокчейн-сеть, используемая для хранения всех типов файлов децентрализованным, одноранговым (P2P) способом без доверия (*trustless*). Он призван заменить протокол передачи гипертекста (HTTP), доминирующий протокол запроса-ответа в Интернете.

устранить типичные недостатки централизованных соцсетей, предлагается использование набирающих популярность децентрализованных технологий: блокчейна и межпланетная файловая система [2] (IPFS). Подобные технологии позволяют создать надежную, защищенную децентрализованную систему, доступ к которой возможен с различных платформ. Благодаря децентрализации возможно значительно сократить влияние кибератак на работоспособность системы, предоставить больший контроль пользователей над собственными персональными данными и построить более безопасную систему, не подверженную цензуре и ограничениям со стороны третьих лиц, по сравнению с централизованными аналогами. Также децентрализация системы подразумевает возможность масштабирования и развертывания собственных узлов системы, тем самым распределяя нагрузку при росте количества пользователей и объема данных в системе.

Предложенные в статье технические решения позволяют обеспечить более высокую производительность по сравнению с конкурентными решениями благодаря использованию языка программирования Golang, а также обеспечить более быструю загрузку контента, защищенность и децентрализацию данных благодаря использованию IPFS и Ethereum. Также нами сформированы требования к разрабатываемой децентрализованной социальной сети и составлено техническое задание на проектирование и разработку приложения, а также была выбрана адаптированная модель жизненного цикла Agile, позволяющая вести гибкую разработку с учетом изменения требований к приложению.

Спроектирована эффективная архитектура социальной сети, позволяющая уменьшить связность компонентов приложения и повышающая отказоустойчивость всей системы: разработанная архитектура учитывает возможности и ограничения блокчейна Ethereum и определяет использование сети IPFS для хранения данных в системе и повышения эффективности работы всей системы. Благодаря использованию смарт-контракта Ethereum и сети IPFS, система предоставляет возможность децентрализованного доступа пользователя к системе с любого подходящего клиента, а с помощью провайдера Metamask – удобство взаимодействия с системой.

В совокупности предложенные технические, математические и другие решения значительно сокращают народнохозяйственные затраты России как социального государства на разработку и внедрение в практику предлагаемых решений на всех уровнях управления от федерального до муниципального. Кроме того, децентрализованная социальная сеть способствует стабилизации в обществе и повышает уровень доверия людей государству.

Исследовательский раздел.

Анализ существующих конкурентных решений среди социальных сетей. На рынке уже присутствуют некоторые конкурентные решения в области децентрализованных социальных сетей. Рассмотрим самые популярные и развитые социальные сети среди них.

Mastodon [3], Misskey [4] и diaspora [5] – децентрализованные социальные сети, которые были созданы в ответ на проблемы, связанные с централизованными социальными сетями, такими как Twitter и Facebook. Diaspora* использует собственный протокол diaspora*[2] federation protocol [6] для взаимодействия с другими социальными сетями, такими как Friendica и Hubzilla, тогда как Mastodon и Misskey используют протокол ActivityPub [7] для обмена данными между различными серверами. Рассмотрим каждую из них более подробно.

Mastodon – децентрализованная социальная сеть, предоставляющая функции микроблогинга для обмена короткими сообщениями, известными как «гудки» («toots»). Mastodon работает на принципе распределенной сети, где серверы, называемые «узлами», управляются их администраторами и могут взаимодействовать друг с другом. Примечательно то, что любой пользователь может создать свой узел и администрировать его на свое усмотрение. Пользователи могут создавать аккаунты на любом узле и общаться с другими пользователями, независимо от того, на каком узле они зарегистрированы. Каждый узел устанавливает свои собственные правила, касающиеся поведения пользователей, и может быть настроен под конкретные потребности сообщества.

Основной функционал Mastodon включает в себя: таймлайн – пользователи могут видеть сообщения от других пользователей, на которых они подписаны; локальный таймлайн – пользователи могут просматривать сообщения от пользователей, зарегистрированных на том же узле, что и они; хэштеги – пользователи могут использовать хэштеги для объединения сообщений по теме; подписки – пользователи могут подписываться на других пользователей, чтобы видеть их сообщения в своем таймлайне. Mastodon также поддерживает различные форматы контента, включая текстовые сообщения, изображения и видео; она также интегрируется с другими децентрализованными социальными сетями – такими как Diaspora и Friendica, что позволяет поддерживать общение с пользователями этих сетей.

Misskey – это децентрализованная социальная сеть с открытым исходным кодом, которая разработана на языке программирования Node.js и использует базу данных MongoDB [8]. Аналогично Mastodon, Misskey является платформой для микроблоггинга и одной из замен Twitter в Fediverse [9]. В целом имеет схожий с Mastodon основной функционал. Одной из ключевых особенностей Misskey является то, что она поддерживает децентрализованную модель данных, что означает, что каждый участник сети может хранить свои данные на своем собственном устройстве. Это повышает уровень безопасности и приватности пользователей, так как они могут контролировать свои данные и решать, кому предоставлять доступ к ним. Функционал поддерживает обмен текстом, изображениями, видео и аудио, а также отображает предупреждения о контенте и проводит опросы. Можно также

[2] Знак "звёздочка" в данном случае является частью логотипа.

использовать хэштеги и находить сообщения на их основе, но подписаться на хэштег нельзя. Нельзя редактировать свои сообщения, но можно их удалить и опубликовать заново. Misskey также предлагает ряд дополнительных функций, которые делают ее уникальной среди других децентрализованных социальных сетей. Например, она поддерживает функцию автоматического перевода текстов на другие языки, что может быть полезно для пользователей, которые общаются на разных языках.

Diaspora* – это социальная сеть, которая позволяет пользователям создавать свои собственные серверы и подключаться к другим серверам. Она имеет функциональность, аналогичную Facebook, включая профили пользователей, группы и возможность обмена сообщениями.

Diaspora* также позволяет пользователям создавать закрытые группы и общаться внутри них. Эта сеть разработана с учетом приватности и безопасности пользователей, а также с целью предоставления им контроля над своими данными. Она отличается от Mastodon и Misskey более широкими возможностями управления приватностью, и более широкой интеграцией с другими социальными сетями:

– в отличие от Mastodon и Misskey, которые ориентированы на собственную экосистему, Diaspora* предоставляет возможность подключаться к другим социальным сетям и сервисам – таким как Facebook и Twitter;

– более ограниченный функционал: например, отсутствие возможности создания тредов или многопользовательских чатов. Однако это делает ее более удобной для пользователей, которые ищут простую и безопасную альтернативу централизованным социальным сетям.

Все три социальные сети имеют свои преимущества и недостатки.

Mastodon имеет большое количество пользователей и сообществ, но может быть сложно найти интересных людей для подписки.

Misskey имеет более удобный интерфейс и меньше ограничений на размер сообщений, но ее пользовательская база меньше, чем у Mastodon.

Diaspora* имеет более простой интерфейс и более удобную систему поиска пользователей и сообществ, но ее пользовательская база также меньше, чем у Mastodon.

Однако хоть вышеперечисленные социальные сети и распределены, они не полностью избавились от централизации – подключение к социальной сети осуществляется через какой-нибудь узел, который может быть недоступен. Узлы, в свою очередь, полностью контролируются администраторами, что может нарушать приватность пользователей, зарегистрированных на них. И наконец, загрузка медиаконтента реализована все так же через CDN [10] или через запрос к узлу, на котором есть эти данные. Проанализируем способы устранения этих недостатков для повышения доступности и скорости загрузки контента.

Анализ способов повышения скорости загрузки контента. Есть несколько способов, которые могут помочь повысить скорость загрузки контента для пользователей в социальных сетях:

1) оптимизация изображений и видео: изображения и видео можно оптимизировать, чтобы уменьшить их размер и ускорить время загрузки. Например, можно использовать форматы изображений с меньшим размером, сокращать продолжительность видео, уменьшать разрешение и т. д.;

2) использование CDN: Content Delivery Network (CDN) – это сеть серверов, расположенных в разных частях мира, которые могут быстро доставлять контент пользователям в их регионах. Использование CDN может существенно ускорить загрузку контента для пользователей;

3) кэширование: кэширование – это процесс сохранения данных на сервере или на устройстве пользователя, чтобы ускорить доступ к этим данным в будущем. Кэширование может существенно ускорить загрузку контента для пользователей;

4) поддержка HTTP/2 [11]: HTTP/2 – это протокол передачи данных, который предоставляет более быстрый и эффективный способ передачи данных в Интернете. Поддержка HTTP/2 может помочь ускорить загрузку контента для пользователей;

5) использование прогрессивной загрузки: прогрессивная загрузка – это технология, которая позволяет загружать контент по частям. Например, сначала загружается небольшое изображение низкого разрешения, а затем более крупное изображение высокого разрешения. Это позволяет пользователю быстро увидеть контент, даже если он не полностью загружен.

С учетом недостатков аналогов разрабатываемое решение должно быть максимально независимым от серверов и доступным для пользователей. Таким образом, нежелательно использовать CDN, а также оптимизировать изображение и видео, допуская потерю качества. Использование HTTP/2 также нецелесообразно, поскольку протокол еще не широко распространен и может не поддерживаться браузером пользователя. Целесообразным решением в таком случае является использование межпланетной файловой системы (IPFS) – протокола и распределенной системы хранения и обмена файлами, которая использует уникальный идентификатор содержимого в качестве адреса для доступа к файлам.

IPFS основывается на технологиях блокчейна и P2P-сетей и позволяет создавать децентрализованные приложения с высокой степенью безопасности и доступности. Она также может использоваться для обмена контентом в сети без централизованного контроля, что делает ее полезным инструментом для создания децентрализованных социальных сетей и других приложений.

Когда пользователь запрашивает файл, IPFS находит наиболее близкий узел с запрошенным контентом и загружает данные с этого узла, что может значительно ускорить процесс загрузки, особенно если контент был загружен ранее и уже есть в сети. IPFS также позволяет кэшировать контент на локальном

устройстве пользователя, что может увеличить скорость загрузки в будущем. Кроме того, IPFS может быть интегрирована с другими технологиями, такими как CDN, чтобы ускорить процесс загрузки и снизить нагрузку на сеть.

Выбор средств разработки приложения. Выбор средств разработки для децентрализованной социальной сети является важным этапом проекта. Для достижения поставленных целей необходимо выбрать оптимальный набор инструментов, который обеспечит эффективную работу над проектом и сократит время его разработки. Исходя из анализа конкурентных решений и возможностей оптимизации разрабатываемого решения по сравнению с аналогами, основополагающим компонентом разрабатываемой социальной сети является IPFS как протокол для распределенного хранения и обмена файлами. Для регистрации и авторизации пользователей и защиты их персональных данных мы будем также использовать блокчейн-платформу Ethereum [12], которая является одной из самых популярных платформ, активно развивается и предоставляет использование смарт-контрактов для разработки децентрализованных приложений. С учетом недостатка блокчейна Ethereum по ограничению размера записываемого объема данных платформа плохо оптимизирована для хранения больших файлов. Но благодаря использованию IPFS файлы можно хранить в децентрализованной файловой системе, а в блокчейн записывать хэш и метаинформацию о сохраненном файле. Таким образом, можно получить эффективную работу этих технологий в паре.

Выбор средств разработки клиентской части. Для разработки клиентской части децентрализованной социальной сети можно выбрать один из нескольких популярных фреймворков: React [13], Angular [14] или Vue [15].

React является одним из самых популярных фреймворков для разработки клиентской части, который обладает широкой функциональностью и большой экосистемой. Однако его использование может быть связано с большим количеством boilerplate-кода и сложностью внедрения.

Angular также является популярным фреймворком для разработки клиентской части, который обладает многими возможностями «из коробки», такими как поддержка маршрутизации, форм и валидации. Но его использование может быть связано с трудностями внедрения и более сложным процессом обучения.

Vue – относительно новый фреймворк, который также обладает многими возможностями, но при этом имеет более простую структуру и легче внедряется в проект. Vue также имеет удобную документацию, что упрощает процесс изучения фреймворка.

В нашем случае выбор Vue будет наиболее подходящим, так как его простота и удобная документация ускорят процесс разработки клиентской части и облегчат процесс обучения. Кроме того, Vue также обладает гибкой архитектурой, что позволит легко интегрировать его с IPFS и Ethereum, используя для этого подходящие библиотеки и плагины.

Выбор средств разработки серверной части. IPFS реализована на языке программирования Go (Golang), и разработчики предоставляют пакет go-ipfs-api [16] для взаимодействия с узлом IPFS. Кроме того, Go можно использовать для написания смарт-контрактов на платформе Ethereum с помощью библиотеки go-ethereum [17], которая является официальной реализацией Ethereum на Go. Эта библиотека предоставляет множество функций для работы с Ethereum – таких, как создание и отправка транзакций, чтение и запись данных в блокчейн и многое другое. Кроме того, Go имеет высокую скорость выполнения и эффективность, что может быть важным при работе с блокчейн-технологиями, которые зачастую имеют высокие требования к производительности. Таким образом, Go является самым подходящим языком программирования для разработки заявленной социальной сети. Рассмотрим несколько популярных фреймворков для Go.

1. Fiber [18] – это быстрый и эффективный веб-фреймворк, который основан на fasthttp. Он имеет простой и понятный интерфейс API, который удобен для разработчиков. Fiber поддерживает многопоточность и встроенную поддержку WebSocket, а также предоставляет инструменты для обработки запросов и ответов, маршрутизации, сжатия, кэширования и многое другое. Fiber имеет дружественную документацию и активную поддержку сообщества.
2. Gin [19] – это еще один быстрый и эффективный веб-фреймворк, который также основан на fasthttp. Он имеет более широкий набор функций, чем Fiber, включая встроенную поддержку мидлваров, простую маршрутизацию и шаблонизацию. Gin также имеет поддержку многопоточности, WebSocket и поддержку RESTful API. Gin также имеет дружественную документацию и активное сообщество, которое предоставляет множество плагинов и дополнений.
3. Echo [20] – это еще один быстрый и легкий веб-фреймворк для языка Go. Он имеет простой и понятный интерфейс API, который позволяет быстро создавать веб-приложения. Echo также поддерживает встроенные механизмы маршрутизации, мидлвары, сжатие и кэширование. Однако он не имеет встроенной поддержки WebSocket и многопоточности.

Для разработки серверной части решено использовать фреймворк Fiber для Go, который является легковесным и быстрым фреймворком для разработки веб-приложений. Он достаточно популярен, имеет понятную документацию на нескольких языках и, по сравнению с другими рассмотренными фреймворками, активно развивается в последние годы.

Проектный раздел.

Выбор адаптированной модели жизненного цикла разработки приложения. Адаптированные модели жизненного цикла [21] – это способы описания этапов разработки программного обеспечения с учетом особенностей конкретного проекта. Наиболее распространенными моделями являются Waterfall,

Agile, Spiral и V-образная. Waterfall – линейная модель, где каждый этап проходится последовательно, и к предыдущему не возвращаются. Это подход, который обычно используется для крупных проектов с ясно определенными требованиями и четкими сроками. Agile – гибкая модель, которая позволяет разработчикам быстро реагировать на изменения требований.

Она подходит для более гибких проектов, где требования могут меняться на протяжении всего процесса разработки. Spiral – итерационная модель, где процесс разработки проходит через несколько циклов. Каждый цикл представляет собой фазу планирования, разработки и тестирования. Эта модель подходит для проектов с высоким уровнем риска или неопределенности. V-образная модель сочетает преимущества Waterfall и Spiral, предполагая более подробную разработку требований и тестирования в начале процесса разработки, а затем включая итеративные циклы для дальнейшей разработки и тестирования.

Учитывая специфику разработки многоплатформенной децентрализованной социальной сети с использованием IPFS, наиболее подходящей моделью жизненного цикла может быть Agile. Это позволит быстро реагировать на изменения требований и учитывать возможности технологий, которые еще находятся в разработке.

Разработка архитектуры приложения. Архитектура разрабатываемой социальной сети будет состоять из следующих компонентов:

1) клиентская часть (Frontend) – пользовательский интерфейс, который будет предоставлять функциональность пользователям социальной сети. Может быть реализована как браузерное веб-приложение или мобильный клиент;

2) серверная часть (Backend) – сервер, который будет предоставлять API для клиентской части. Необходим для взаимодействия пользователя с конкретным узлом IPFS для сохранения и получения контента;

3) IPFS (InterPlanetary File System) – распределенная файловая система, которая будет использоваться для хранения контента пользователей. IPFS предоставляет децентрализованный и устойчивый способ хранения файлов, используя хеширование контента. Контент будет храниться на IPFS-узлах, а не на централизованном сервере.

4) Ethereum – платформа для смарт-контрактов, которая будет использоваться для регистрации и управления данными пользователей. Ethereum предоставляет распределенную базу данных, которая позволяет создавать и исполнять смарт-контракты. Данные пользователей будут храниться в смарт-контрактах Ethereum, которые будут доступны для чтения и записи из любого узла сети.

Разработанная функциональная схема на основе перечисленных компонентов представлена на рис. 1.

Клиентская часть будет взаимодействовать с серверной частью через разработанный API, который будет предоставлять доступ к функционалу социаль-

ной сети – такой как регистрация, вход в систему, создание и просмотр контента. Серверная часть будет использовать IPFS для хранения контента, а Ethereum – для регистрации и управления данными пользователей и, таким образом, обеспечивать децентрализацию и защиту данных. Благодаря такой архитектуре социальной сети, пользователь может в любой момент сменить клиентское приложение, узел серверной части или узел IPFS, сохранив доступ к социальной сети, своим данным и контенту других пользователей. Более того, пользователь может разработать и использовать собственную реализацию клиентской части, развернуть локальный узел серверной части или локальный узел IPFS, например, для кэширования и оффлайн-доступа к сохраненному контенту.

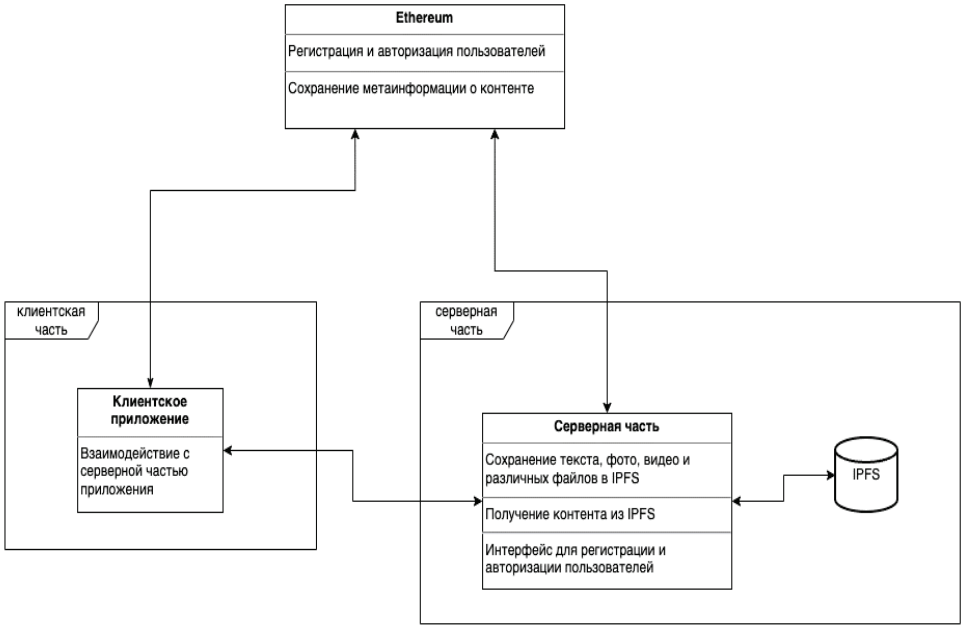

Рис. 1. Функциональная схема приложения.
Источник: разработано авторами.

Разработка архитектуры клиентской части приложения. Для обеспечения децентрализации разрабатываемой системы было решено использовать смарт-контракты блокчейна Ethereum: с помощью смарт-контракта можно реализовать регистрацию и аутентификацию пользователя в социальной сети, а также хранить информацию о пользователе в блокчейне и список опубликованных им постов. Однако такой подход имеет свои ограничения: любое изменение блокчейна, например, обновление информации о пользователе или регистрация нового пользователя, требует создания новой транзакции, подписанной пользовательским приватным ключом от криптокошелька.

Для того чтобы изменения, содержащиеся в транзакции, были применены, необходимо, чтобы транзакция была отправлена в сеть блокчейна и обработана майнером, что влечет за собой оплату пользователем комиссии за работу майнера. Из-за того что приватный ключ пользователя нельзя публиковать в сети Интернет и можно хранить лишь в зашифрованном хранилище на физическом устройстве, все транзакции по изменению информации в блокчейне будут осуществляться с клиентской части приложения, а серверная часть будет обращаться к функционалу смарт-контракта только для чтения информации, сохраненной в блокчейне. Для ускорения разработки и удобства использования системы пользователем будет использоваться провайдер Metamask [22].

Metamask – это популярный кошелек для Ethereum и расширение для браузера, которое позволяет пользователям взаимодействовать с децентрализованными приложениями Ethereum непосредственно из веб-браузера. Провайдер Metamask обеспечивает удобный и безопасный способ для клиентской части децентрализованного приложения получать доступ к кошельку пользователя и инициировать транзакции от его имени, не требуя от приложения непосредственного доступа к приватным ключам пользователя. Для работы с Metamask пользователю необходимо использовать любой браузер с поддержкой расширения Metamask на компьютере или мобильном устройстве.

Разработанная архитектура клиентской части представлена на рис. 2. В разработанной архитектуре пользователь с помощью веб-приложения клиентской части и подключенного провайдера Metamask может изменять информацию для хранения в блокчейне или записывать новую информацию в нем.

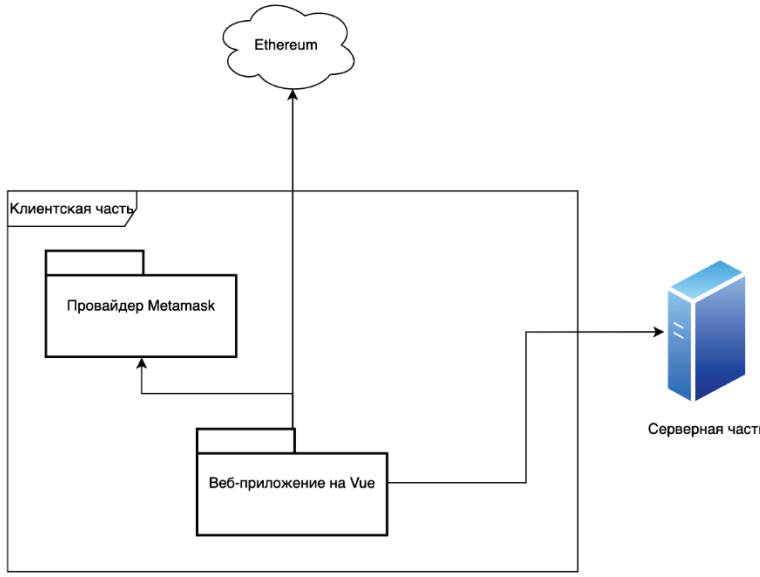

Рис. 2. Архитектура клиентской части.

Источник: разработано авторами.

Для оптимизации работы системы и уменьшения размера комиссии за выполнение транзакции в блокчейне в него записывается лишь CID файла, сохраненного в IPFS с помощью API серверной части приложения.

Разработка архитектуры серверной части приложения. Для разработки прототипа серверной части приложения архитектуру серверной части можно разбить на следующие модули: модуль маршрутизации и обработки запросов отвечает за принятие запросов от клиента, маршрутизацию их к соответствующим обработчикам, а также за обработку запросов и отправку ответов клиенту; модуль обработки Ethereum-транзакций отвечает за взаимодействие с блокчейн-сетью Ethereum: запрашивает информацию из смарт-контрактов и обрабатывает полученные результаты; модуль обработки IPFS-запросов обеспечивает взаимодействие с IPFS-сетью, загрузку и хранение файлов в децентрализованной файловой системе; модуль уведомлений отвечает за отправку пользователям уведомлений о различных событиях в приложении – таких как новые сообщения, комментарии, лайки и другие. Схема архитектуры серверной части социальной сети представлена на рис. 3.

В рамках разработанной архитектуры данные пользователя, а также метаданные о данных в IPFS будут храниться на узле блокчейна, а на узле IPFS будут храниться файлы, в том числе фото и видео, а также данные в формате JSON. Такая модульная архитектура позволяет обеспечить высокую отказоустойчивость и масштабируемость приложения, упростить его тестирование и поддержку, а также обеспечить гибкость и возможность быстрой модификации функционала.

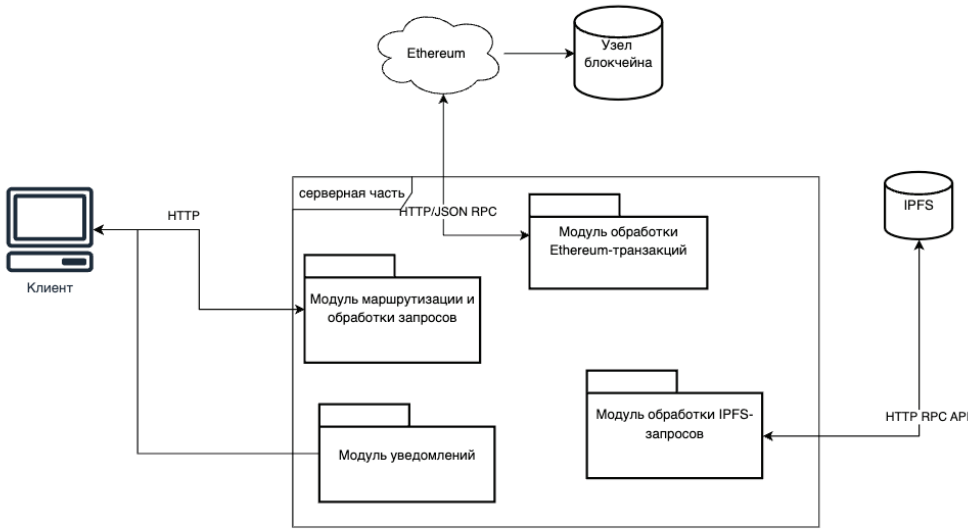

Рис. 3. Архитектура серверной части.

Источник: разработано авторами.

Проектирование схемы базы данных. Для обеспечения децентрализации данных, хранящихся в системе, нецелесообразно использование привычных реляционных и нереляционных СУБД для таких централизованных клиент-серверных систем, как PostgreSQL [23] или MongoDB. С учетом уже используемых технологий IPFS и смарт-контрактов блокчейна Ethereum возможно организовать хранение данных в разрабатываемой системе: на узлах IPFS будут храниться данные, используемые системой, например, изображения в байтовом представлении или информация о постах и их содержимом в формате JSON [24], а в блокчейне Ethereum будут храниться указатели, CID [25], на необходимое содержимое, записанное в IPFS – например список CID-постов, опубликованных пользователем. Для записи данных в сеть IPFS используются модели данных, описанные в виде структур языка Golang, которые конвертируются в формат JSON перед записью. Рассмотрим поля используемых моделей для хранения данных в системе.

Модель «Post»: CID записи в сети IPFS. Заголовок поста. Содержимое поста (модель «PostContent»). ID автора поста. Время создания поста.

Модель «PostContent»: текстовое содержимое поста. Изображения, прикрепленные к посту (модель «Image»). Видео, прикрепленные к посту (модель «Video»). Файлы, прикрепленные к посту (модель «File»).

Модели «Image», «Video» и «File» состоят из двух полей: CID и объекта метаданных содержимого, записанного в IPFS.

Для хранения информации о пользователе используется модель «User» с полями адреса пользователя в сети блокчейна и дополнительной информацией профиля пользователя.

Технологический раздел.

Разработка интерфейса приложения. Для функционирования рассматриваемой социальной сети разработан смарт-контракт Ethereum [26] на языке Solidity [27] и размещен в сети блокчейна. В разработанном контракте реализован функционал регистрации и аутентификации пользователя, получение и изменение информации пользователя, добавление и получение постов пользователя, установка нового пароля пользователя.

Смарт-контракт определяет пользователя по адресу кошелька отправителя транзакции и, таким образом, может идентифицировать текущего пользователя или проверить, существует ли пользователь в социальной сети. Пароль пользователя поступает на вход и хранится в зашифрованном виде с помощью алгоритма SHA256, что обеспечивает безопасность от дешифрования пароля пользователя злоумышленником.

Функции на чтение информации из блокчейна, например, authenticate и getUserInfo, не изменяют информации в блокчейне и будут использоваться на серверной части приложения для аутентификации пользователя и получения необходимой информации о пользователе и его постах из блокчейна.

После создания и развертывания контракта в сети блокчейна разработчик может взаимодействовать с контрактом и использовать его функционал. Взаимодействие с контрактом осуществляется с помощью ABI и вызова функций блокчейна через JSON RPC [28]. Форма регистрации пользователя представлена на рис. 4.

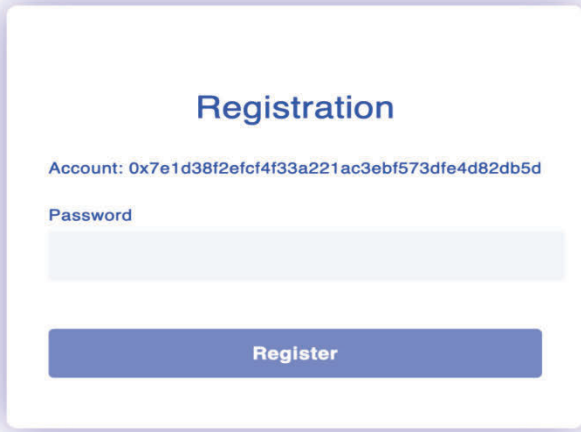

Рис. 4. Форма регистрации.

Источник: разработано авторами.

В поле Account отображен адрес подключенного кошелька пользователя, полученного от провайдера Metamask. Для аутентификации и авторизации пользователя в системе пользователь должен ввести пароль в форму аутентификации, после чего клиентская часть инициирует запрос на аутентификацию к серверной части и, в случае успешной аутентификации, получает JWT-токен [29] для просмотра страницы с постами других пользователей. Форма аутентификации представлена на рис. 5.

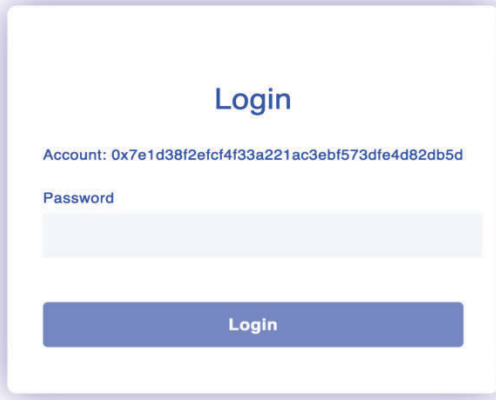

Рис. 5. Форма аутентификации.

Источник: разработано авторами.

После успешной аутентификации пользователь переходит на страницу с постами других пользователей. Клиент получает список постов из ответа на запрос получения всех постов от серверной части приложения при загрузке страницы. Страница со списком постов представлена на рис. 6.

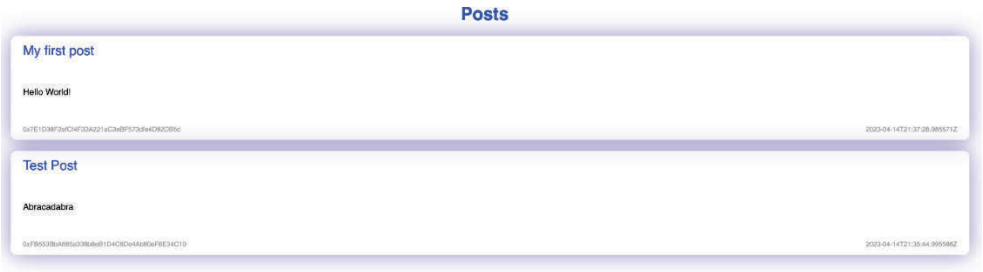

Рис. 6. Список постов пользователей.
Источник: разработано авторами.

По итогам разработки клиентской части децентрализованной социальной сети опишем поведение пользователя при работе с системой. Для начала работы с системой пользователь должен установить расширение провайдера Metamask в браузере, если оно еще не установлено, и перейти по веб-адресу разработанной социальной сети. Если пользователь не зарегистрирован – необходимо нажать на кнопку регистрации и заполнить форму регистрации. После подтверждения формы регистрации нужно подтвердить отправку транзакции для добавления информации о пользователе в блокчейн. Если пользователь зарегистрирован и хочет войти в систему – необходимо нажать кнопку входа и заполнить форму аутентификации. После успешной аутентификации и авторизации пользователь будет направлен на ленту постов пользователей.

На странице с лентой постов пользователей социальной сети пользователь может просматривать посты других пользователей или создать собственный, нажав на кнопку создания поста. После нажатия откроется форма для нового поста. После заполнения необходимых полей и подтверждения публикации поста нужно подтвердить транзакцию с помощью провайдера, после чего пост будет виден как автору, так и другим авторизованным пользователям социальной сети.

Разработка бизнес-логики приложения. Серверная часть децентрализованной социальной сети разработана в соответствии с техническим заданием и обеспечивает децентрализацию и масштабируемость компонентов системы. Смарт-контракт, размещенный в блокчейне Ethereum, позволяет осуществлять регистрацию нового пользователя или идентификацию и аутентификацию уже зарегистрированного пользователя, а также хранить идентификаторы на опубликованные посты и описание профиля пользователя в хранилище IPFS. Для осуществления транзакций и изменения данных в блокчейне пользователь

использует криптокошелек и пару ключей: приватный и публичный. Приватный ключ пользователя хранится только на физическом устройстве пользователя и не отправляется ни в один компонент системы через сеть Интернет. Пользователь должен использовать приватный ключ для подписи транзакций с помощью специального провайдера, например Metamask, и не должен где-либо публиковать приватный ключ или набор ключевых слов для доступа к кошельку через провайдера в сети Интернет.

Для аутентификации пользователя на сервере пароль пользователя предварительно шифруется алгоритмом SHA256, что исключает для злоумышленников возможность расшифровать пароль и повышает безопасность аккаунта пользователя. Для повышения безопасности системы также возможна реализация двухфакторной аутентификации: сервер генерирует случайное сообщение; отправляет на клиентское приложение пользователя, где пользователь подписывает сообщение с помощью своего приватного ключа и отправляет обратно серверу; сервер расшифровывает сообщение публичным ключом пользователя и проверяет принятое сообщение на соответствие сгенерированному.

Для доступа к постам, опубликованным в социальной сети, реализована авторизация пользователя по JWT-токену: после успешной аутентификации пользователь получает персональный токен, который используется в заголовках запроса для просмотра опубликованных постов или создания нового поста. Масштабируемость системы обеспечивается возможностью пользователя развернуть собственный сервер приложения и указать в переменных окружения любой адрес узла IPFS или размещенного в сети Ethereum смарт-контракта. Благодаря принципам работы IPFS значительно сокращается возможность утраты данных в системе за счет хранения блоков данных на нескольких узлах сети, а благодаря сети блокчейна обеспечивается безопасность и подлинность транзакций пользователей.

Рассмотрим более детально функционал взаимодействия серверной части приложения с развернутым смарт-контрактом Ethereum и узлом IPFS.

Взаимодействие с развернутым смарт-контрактом Ethereum в сети блокчейн на серверной части осуществляется с помощью установленного пакета geth. Этот пакет предоставляет API для взаимодействия с сетью блокчейна на языке Go и позволяет автоматически сгенерировать необходимые функции на языке Go для работы с контрактом на основе ABI разработанного смарт-контракта [30]. В случае успешной аутентификации пользователя сервер генерирует JWT-токен с помощью алгоритма HS256 и возвращает клиенту для последующих запросов на защищенные эндпоинты.

Взаимодействие между сервером и узлом IPFS обеспечивается посредством библиотеки go-ipfs-api с помощью вызова процедур через специальный API узла. Для подключения и вызова команд нужно создать специальный объект, «Shell», указав в аргументах адрес узла IPFS, к которому нужно подключиться.

Тестирование приложения. В рамках MVP клиентская часть лишь отправляет HTTP запросы для взаимодействия с сервером через REST API и использует расширение Metamask для взаимодействия со смарт-контрактом Ethereum, поэтому в данном случае целесообразно ограничиться ручным тестированием клиентской части. Тестирование API серверной части и взаимодействие сервера с узлом IPFS и смарт-контрактом Ethereum, напротив, заслуживает большего внимания для проверки работоспособности и надежности ключевых узлов системы. Тестирование создания нового поста на клиентской части представлено на рис. 7 и 8.

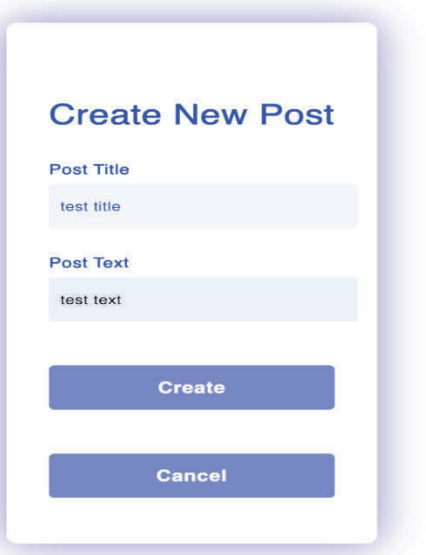

Рис. 7. Создание нового тестового поста.

Источник: разработано авторами.

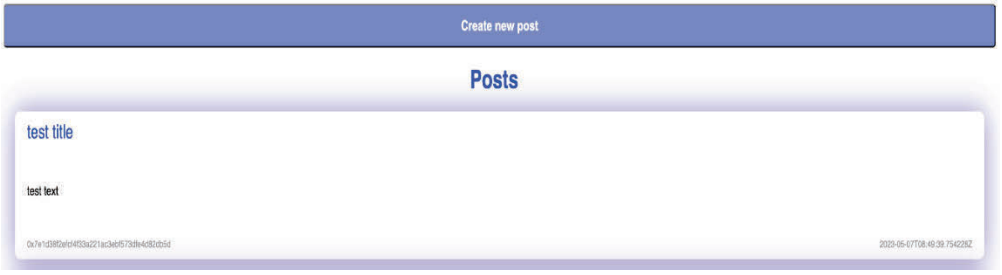

Рис. 8. Отображение тестового поста в ленте.

Источник: разработано авторами.

Для тестирования серверной части используется стандартная библиотека «testing» языка Go и формат для написания тестов фреймворка Fiber. Для тестирования функционала по взаимодействию с IPFS и со смарт-контрактом Ethereum были использованы тестовый узел IPFS и счет в тестовой сети Sepolia блокчейна Ethereum. Рассмотрим тестирование методов контроллера для управления пользователями «Login» и «Register» для аутентификации и регистрации, соответственно. Внутри тестовых функций описаны тест-кейсы для аутентификации и регистрации: «Успешная аутентификация пользователя»; «Неудачная аутентификация пользователя»; «Регистрация нового пользователя»; «Попытка регистрации существующего пользователя».

При тестировании регистрации пользователя тестовый пользователь удаляется в конце теста. Результаты тестирования представлены на рис. 9.

```
❯ go test -v ./...
=== RUN   TestLogin
=== RUN   TestLogin/Successful_login
Authenticated: true
=== RUN   TestLogin/Failed_login
Authenticated: false
--- PASS: TestLogin (0.63s)
    --- PASS: TestLogin/Successful_login (0.47s)
    --- PASS: TestLogin/Failed_login (0.15s)
=== RUN   TestRegister
=== RUN   TestRegister/Successful_Register
=== RUN   TestRegister/Already_registered_user
--- PASS: TestRegister (11.75s)
    --- PASS: TestRegister/Successful_Register (7.95s)
    --- PASS: TestRegister/Already_registered_user (0.01s)
PASS
ok      github.com/RudeZwloki7/project-d/controllers    12.810s
```

Рис. 9. Результаты тестирования серверной части.
Источник: разработано авторами.

Рассмотрим и оценим технические характеристики серверной и клиентских частей. Серверная часть написана на Golang на фреймворке Fiber и использует протокол HTTP для взаимодействия с узлом IPFS и узлом блокчейна Ethereum. Работа приложения потребляет в среднем 20 Мб ОЗУ и обеспечивает время отклика 10–20 миллисекунд (рис. 10).

Рис. 10. Результаты мониторинга серверной части.
Источник: разработано авторами.

Развернутый локальный узел IPFS потребляет в среднем 250 Мб ОЗУ. Для взаимодействия со смарт-контрактом используется платформа Infura, которая предоставляет возможность работы с выделенным узлом блокчейна Ethereum, следовательно, физические ресурсы машины не используются. Клиентская часть написана на Vue и потребляет в среднем 35 Мб ОЗУ. На рис. 11 показано время выполнения запросов к серверной части при аутентификации, получении всех постов в социальной сети и создании нового поста пользователя.

Name	Status	Type	Initiator	Size	Time	Waterfall
login	200	fetch		384 B	144 ms	
all	200	fetch		864 B	3.96 s	
create	201	fetch		368 B	52 ms	
197592839af84887b84a950dd...	200	fetch		194 B	148 ms	
197592839af84887b84a950dd...	200	fetch		197 B	137 ms	

Рис. 11. Результаты мониторинга клиентской части.
Источник: разработано авторами.

Полученный результат потребления ОЗУ компонентами системы и время выполнения операций внутри системы оценены как удовлетворительные.

Расчет вычислительной и емкостной сложности. Для определения ресурсозатратных алгоритмов разработанной системы необходимо рассчитать их временную и емкостную сложность в худшем случае. При расчёте рассматриваются основные функции компонентов системы. Объем входных данных, с которыми работают алгоритмы, принимается за n. Расчёты вычислительной и ёмкостных сложностей основных функций для компонентов системы осуществляются в следующей последовательности:

$$T(GetUsers) = O(1), \quad (1)$$
$$V(GetUsers) = O(n), \quad (2)$$

где T – вычислительная сложность; *GetUsers* – функция получения списка пользователей; O – трудоёмкость алгоритма; V – ёмкостная сложность; n – размер входных данных.

$$T(GetUsersInfo) = O(n), \quad (3)$$
$$V(GetUsersInfo) = O(n), \quad (4)$$

где T – вычислительная сложность; *GetUsersInfo* – функция получения информации о профилях списка пользователей; O – трудоёмкость алгоритма; n – размер входных данных; V – ёмкостная сложность.

$$T(GetUserPosts) = O(1), \quad (5)$$
$$V(GetUserPosts) = O(n), \quad (6)$$

где T – вычислительная сложность; *GetUserPosts* – функция получения постов пользователя; O – трудоёмкость алгоритма; V – ёмкостная сложность; n – размер входных данных.

$$T(GetUsersPosts) = O(n \times T(GetUserPosts)) = O(n^2), \quad (7)$$
$$V(GetUsersPosts) = V(GetUserPosts) = O(n), \quad (8)$$

где T – вычислительная сложность; *GetUsersPosts* – функция получения информации постов по списку пользователей; O – трудоёмкость алгоритма; n – размер входных данных; V – ёмкостная сложность.

Результаты расчета сложности алгоритмов затем сводятся в таблицу.

Таблица

Сложность основных функций компонентов системы

Функция	Описание	*T*	*V*
GetUsers	Функция получения списка пользователей	*O*(1)	*O(n)*
GetUsersInfo	Функция получения информации о профилях списка пользователей	*O(n)*	*O(n)*
GetUserPosts	Функция получения постов пользователя	O(1)	*O(n)*
GetUsersPosts	Функция получения постов по списку пользователей	$O(n^2)$	*O(n)*

Источник: разработано авторами.

Результаты расчетов показали, что наиболее ресурсозатратной функцией серверной части является *GetUsersPosts*, используемая алгоритмом получения списком постов по списку пользователей. Для клиентской части функции оптимизированы для активного использования хранилища приложения и в наихудшем случае (отсутствие данных в хранилище) показывают линейную сложность. Объем обрабатываемых данных является небольшим, в связи с чем обеспечивается скорость выполнения описанных выше запросов (функций).

Заключение. В результате исследований проведена разработка многоплатформенной децентрализованной социальной сети с использованием межпланетной файловой системы. Проведен анализ конкурентных решений для выявления их недостатков и формирования требований к разработанной социальной сети. В результате анализа приведены подходящие технологии и средства разработки для обеспечения децентрализации системы и повышения скорости загрузки хранящегося в ней контента. Делается обоснованный вывод, что использование межпланетной файловой системы и блокчейна снизит централизацию компонентов системы и окажет положительное влияние на скорость загрузки данных, хранящихся в социальной сети на стороне клиента пользователя. Использование технологий блокчейна и IPFS в связке с алгоритмами шифрования и авторизацией пользователя по JWT-токену обеспечивает необходимую безопасность пользовательских данных при эксплуатации разработанной децентрализованной социальной сети. Применение смарт-контрактов Ethereum и узлов сети IPFS с возможностью развертывания и использования собственных состояний при конфигурации компонентов системы обеспечивают масштабирование и больший контроль пользователя над своими персональными данными в системе. За счет этого достигается стабильность работы при увеличении числа пользователей без снижения скорости загрузки данных, хранящихся в системе.

Важно отметить, что в процессе разработки был реализован следующий функционал: регистрация и авторизация пользователя, публикация нового поста и отображение всех постов социальной сети. Для ограничения доступа к защищенным ресурсам системы использовано взаимодействие клиента и сервера с помощью JWT-токена. В результате тестирования проверена работоспособность разработанного функционала компонентов социальной сети. В ходе оценки технических характеристик разработки было выявлено потребление объема ОЗУ серверной частью социальной сети в среднем 270 Мб (вместе с локальным узлом IPFS) и клиентской частью – в среднем 35 Мб. Таким образом, в результате разработки получено решение, обеспечивающее эффективное взаимодействие с узлом IPFS и смарт-контрактом Ethereum. Разработанное решение обеспечивает более высокую скорость доставки контента, хранящегося в IPFS, и более низкую централизованность данных по сравнению с конкурентными решениями. В результате расчета вычислительной и емкостной сложностей алгоритмов определено, что наиболее ресурсоемким алгоритмом является получение опубликованных постов по списку пользователей.

Экономическая целесообразность разработки многоплатформенной децентрализованной социальной сети с использованием межпланетной файловой системы заключается в том, что данный программный продукт позволит сократить время загрузки и отображения информации в социальной сети, значительно повысить безопасность пользовательских данных и обеспечить больший

контроль пользователя над их распространением и отображением. При использовании разрабатываемого веб-приложения снижается риск утечки персональных данных пользователей по сравнению с конкурентными централизованными решениями. Также возможно размещение решения в сети Интернет в соответствии с политикой открытого исходного кода для уменьшения возможных уязвимостей и расширения функционала разрабатываемого продукта с помощью мирового сообщества. Выбор программных и организационно-технологических проектных решений обеспечил минимизацию финансовых, материальных и трудовых затрат. Разработка является экономически целесообразной.

Литература

1. *Аникеев С.А.* Параграф 4.5. Парадигмы программирования в монографии: Мобилизационно-военная индустриализация / автор Кохно П.А. // Москва: Институт нечётких систем, 2023. – 217 с. С. 132–135. URL: http://innclub.info / (дата обращения: 28.06.2023).
2. IPFS Powers the Distributed Web. URL: https://ipfs.tech/ (дата обращения: 22.05.2023).
3. Mastodon – Decentralized social media. URL: https://joinmastodon.org/(дата обращения: 27.06.2023).
4. About Misskey | Misskey Hub. URL: https://misskey-hub.net/en/docs/misskey.html (дата обращения: 27.06.2023).
5. The diaspora* Project. URL: https://diasporafoundation.org/ (дата обращения: 27.06.2023).
6. diaspora* federation protocol. URL: https://diaspora.github.io/diaspora_federation/ (дата обращения: 28.06.2023).
7. ActivityPub. URL: https://www.w3.org/TR/activitypub/ (дата обращения: 28.06.2023).
8. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/ (дата обращения: 02.06.2023).
9. Fediverse. URL: https://ru.wikipedia.org/wiki/Fediverse (дата обращения: 03.06.2023).
10. Что такое CDN и как это работает? URL: https://habr.com/ru/companies/selectel/articles/463915/ (дата обращения: 06.06.2023).
11. Hypertext Transfer Protocol Version 2 (HTTP/2). URL: https://httpwg.org/specs/rfc7540.html (дата обращения: 06.06.2023).
12. Home | ethereum.org. URL: https://ethereum.org/en/(дата обращения: 07.06.2023).
13. React. URL: https://react.dev/ (дата обращения: 10.06.2023) – Текст: электронный..
14. Angular. URL: https://angular.io/ (дата обращения: 10.06.2023).
15. Vue.js – The Progressive JavaScript Framework | Vue.js. URL: https://vuejs.org/ (дата обращения: 10.06.2023).
16. go-ipfs-api – Go Packages. URL: https://pkg.go.dev/github.com/ipfs/go-ipfs-api (дата обращения: 13.06.2023).
17. Home | go-ethereum. URL: https://geth.ethereum.org/ (дата обращения: 13.06.2023).
18. Fiber. URL: https://gofiber.io/ (дата обращения: 15.06.2023).
19. Gin Web Framework. URL: https://gin-gonic.com/ (дата обращения: 15.06.2023).
20. Echo – High performance, minimalist Go web framework. URL: https://echo.labstack.com/ (дата обращения: 15.06.2023).
21. Обзор моделей жизненного цикла разработки программного обеспечения. URL: https://pandia.ru/text/77/217/2229.php (дата обращения: 24.06.2023).
22. MetaMask: The crypto wallet for Defi, Web3 Dapps and NFTs. URL: https://metamask.io/ (дата обращения: 27.06.2023).

23. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/ (дата обращения: 03.06.2023).
24. Работа с JSON. URL: https://developer.mozilla.org/ru/docs/Learn/JavaScript/Objects/JSON (дата обращения: 05.06.2023).
25. Content Identifiers (CIDs) – IPFS Docs. URL: https://docs.ipfs.tech/concepts/content-addressing/ (дата обращения: 05.06.2023).
26. Introduction to smart contracts. URL: https://ethereum.org/en/smart-contracts/ (дата обращения: 07.06.2023).
27. Solidity. URL: https://docs.soliditylang.org/en/v0.8.20/ (дата обращения: 07.06.2023).
28. Реализация алгоритма SHA-256. URL: https://habr.com/ru/articles/729260/ (дата обращения: 10.06.2023).
29. JSON-RPC 2.0 Specification. URL: https://www.jsonrpc.org/specification (дата обращения: 12.06.2023).
30. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction (дата обращения: 14.06.2023).

Semyon Anikeev (e-mail: pavelkohno@mail.ru)
Head of the Center for Computer Technologies of the Institute of Fuzzy Systems
(Moscow, Russian Federation)

Pavel Kokhno (e-mail: pavelkohno@mail.ru)
Grand Ph.D. in Economics, Professor,
Director of the Institute of the Fuzzy Systems
(Moscow, Russian Federation)

ON THE TECHNOLOGIES OF DECENTRALIZED SOCIAL NETWORKS

The article analyzes modern centralized social networks, taking into account the existing shortcomings. A methodical approach to the development of a multiplatform decentralized social network using an interplanetary file system is proposed. Besides, a decentralized social network, an Ethereum smart contract, and a module for interacting with the interplanetary file system have been developed, a solution for the client side had been designed, a database and a local server have been configured. The ways to increase the speed of loading content are analyzed. To develop the client part of a decentralized social network, several popular frameworks have been proposed. Adapted life cycle models have been investigated, that is, ways to describe the stages of software development, depending on the characteristics of a particular project. Given the specifics of developing a multi-platform decentralized social network using IPFS, Agile is supposed to be the most appropriate life cycle model. An algorithm is proposed to calculate the computational and capacitive complexity of the system being developed. The calculation takes into account the main functions of the system components.

Keywords: interplanetary file system, Ethereum smart contract, decentralized social network, competitive solutions, users, blockchain, CID, multiplatform, Metamask provider, content download.

DOI: 10.31857/S020736760027685-2