МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики  
Кафедра информационных технологий**

**КУРСОВАЯ РАБОТА**

**ИССЛЕДОВАНИЕ И ПРИМЕНЕНИЕ СОВРЕМЕННЫХ WEB-ТЕХНОЛОГИЙ ДЛЯ РЕАЛИЗАЦИИ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ IoT СИСТЕМ**

Работу выполнил\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Е. Грибков

(подпись)

Направление подготовки 02.03.03 “Математическое обеспечение и администрирование информационных систем” курс 3

Направленность (профиль) Технология программирования

Научный руководитель

канд., физ.-мат. наук\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Подколзин

(подпись, дата)

Нормоконтролер

ст. преп.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.В. Харченко

(подпись, дата)

Краснодар  
 2020

**РЕФЕРАТ**

Курсовая работа содержит 36 страниц, 9 рисунков, 12 источников

WEB-ТЕХНОЛОГИЯ, IoT, ВЕБ ПРИЛОЖЕНИЕ, ОДНОСТРАНИЧНОЕ ПРИЛОЖЕНИЕ

В работе рассматривается IoT технологии, пути их развития, а также основные проблемы, рассматриваются и сравниваются различные WEB технологии для серверного и клиентского приложения. Создана система, позволяющая совмещать умные устройства в одном сервере, который сам отслеживает состояние каждого из устройств в виде хранилищ, и при срабатывании определенных событий выполняет действия, описанные пользователем. К серверу написано клиентское приложение, с использованием современных WEB технологий в виде одностраничного приложения, позволяющее визуально настроить различные параметры сервера, а также визуально увидеть данные, хранящиеся в каждом из хранилищ.

СОДЕРЖАНИЕ

[Введение 4](#_Toc39759580)

[1 Интернет вещей 5](#_Toc39759581)

[1.1 Сети интернет вещей 9](#_Toc39759582)

[1.2 Проблемы интернет вещей 11](#_Toc39759583)

[1.2.1 Платформенная фрагментация 11](#_Toc39759584)

[1.2.2 Безопасность 12](#_Toc39759585)

[1.2.3 Конфиденциальность 13](#_Toc39759586)

[1.2.4 Стандартизация 14](#_Toc39759587)

[2 Постановка задачи 15](#_Toc39759588)

[3 Сравнение и выбор инструментов разработки сервера 16](#_Toc39759589)

[3.1 Язык и фреймворк 16](#_Toc39759590)

[3.2 Выбор системы управления базой данных 19](#_Toc39759591)

[3.3 Архитектура приложений 20](#_Toc39759592)

[3.4 Авторизация и аутентификация 23](#_Toc39759593)

[4 Моделирование базы данных, точки входа 25](#_Toc39759594)

[5 Конечные адреса 27](#_Toc39759595)

[6 Клиентское приложение 28](#_Toc39759596)

[6.1 JavaScript библиотеки 28](#_Toc39759597)

[6.2 CSS-библиотеки 30](#_Toc39759598)

[6.3 Пакетный менеджер и сборка проекта 31](#_Toc39759599)

[6.4 Итоговое клиентское приложение 32](#_Toc39759600)

[Заключение 34](#_Toc39759601)

[Список использованных источников 35](#_Toc39759602)

# ВВЕДЕНИЕ

IoT системы повсеместно используются в квартирах людей, на производствах, и даже в рамках целых городов, создавая умные экосистемы. При этом развитие направление не прекращаются и с каждым днем появляются новые идеи и концепции. При этом многие решения созданы на этапе зарождения и появления умных систем, потому используют относительно устаревшие технологии.

Целью данной работы являются изучение современных веб технологий, и создание на их основе системы построения и управления IoT сетями, для этого необходимо решить следующие задачи:

* рассмотрение концепции IoT, исследование её проблем и направлений развития;
* рассмотрение современных веб технологий;
* разработка сервера для управления умными устройствами;
* разработка клиента, для настройки сервера, а также визуального отслеживания работы подключенных устройств.

# Интернет вещей

Интернет вещей [8] (IoT) – концепция вычислительной системы, в которой множества устройств, имеющий уникальный индикационный номер (UUID), способы обмениваться информацией и взаимодействовать между собой или с внешним миром, минуя или ограничивая необходимость человека.

Таким образом на сегодняшний день насчитывается более 5 биллионов различных устройств, подключенных к локальной сети или внешнему Интернету.

Подобные системы используются во многих средах: жилые здания, заводы, транспорт, города, платежные системы, носимые устройства и другие. Такая распространённость обусловлена быстрым ростом систем коммуникаций, разработке новых способов передачи информации, эволюции вычислительные процессоров, а также общее уменьшение затрат на передачу и хранение информации, а также на её обработку.

Огромные количество областей применений IoT устройств обычно разделяется на потребительские, промышленные и инфраструктурные области:

Часто IoT устройства являются частью большой системы домашней автоматизации, включающей в себя управление освещением, климат контроль, медиа систему и систему безопасности.

Умные или автоматизированные дома могут основываться на платформах или специальных концентраторах (хабах), которые управляют подключенными к ним домашними устройствами [8]. Для этого обычно применяется закрытая локальная сеть, распространяющаяся на территории дома. Также умные устройства одного производителя могут коммуницировать между собой без связывающей их интернет сети. В данный момент на рынке существует большое количество коммерческих продуктов в этой области, таких как Google Home, Amazon Echo и другие, а также непатентованные некоммерческий продукты: OpenHAB, Home Assistant, Majordomo и тд.

Одной из ключевых областей применений IoT систем является оказание помощи людям с ограниченными возможностями и пожилым людям. Обычно такие системы используют вспомогательные модули для помощи людям с конкретными проблемами. Системы могут быть напрямую взаимодействовать с различными имплантами. Часто подобные устройства оснащены дополнительными функциями безопасности, включающие в себя датчики состояния, позволяющие следить за состоянием человека, использующий их и предпринимать на их основании какие-либо действия. Таким образом подобные устройства способны улучшить качество жизни их пользователей, а также предоставить им больше свободы.

Умные устройства позволяют улучшить качество обслуживания клиентов, уменьшит затраты в сферах коммерции.

Интернет медицинских вещей (IoMT) [11][10] представляет собой приложения, используемые для целей медицины и здравоохранения, сбора и анализа данных для исследования и мониторинга. Подобные устройства могут использоваться в системах удаленного мониторинга и экстренного оповещения, они могут как отслеживать сердечные ритмы или артериальное давление человека, так и управлять специальными имплантами, такие как кардиостимуляторы и слуховые аппараты, внедряются умные кровати, отслеживающие, когда они заняты и когда пациент пытается встать с кровати, а также способные поддерживать необходимое давление и помогать пациенту. Развитие и распространение IoMT позволило собрать достаточно данных для мониторинга общего уровня здоровья стран и отдельных городов.

Жилые и рабочие помещения могут быть оборудованы специальными датчиками для мониторинга здоровья и общего благополучия пожилых людей, а также для обеспечения лечения и необходимых условий выздоровления людей. Подобные системы могут собирать, обрабатывать, передавать и анализировать ценную информацию в средах, таких как домашняя система мониторинга или больничная информационная система.

Применение IoT в сфере здравоохранения играет ключевую роль в лечении хронических заболеваний, а также в профилактике и контроле различных болезней. Удаленный мониторинг, возможный благодаря подключению мощных беспроводных устройств, позволяет врачам собирать данные пациентов и применять сложные алгоритмы в анализе данных о их здоровье.

Также интернет вещей часто применяется в сфере транспорта и логистики. Он может помочь с интеграцией коммуникаций, управления т обработки информации в различных транспортных средах. Такие системы способны обеспечить меж транспортную и внутритранспортную связь, интеллектуальное управление движением, умные парковки, электронные системы оплаты за проезд, развитую логистику, управление автопарком, помощь и безопасность на дороге. Различные датчики способны помечать и отслеживать перемещения грузов, предупреждать о порче или хищении. Также различные IoT устройства в сочетании с машинным обучением могут сократить количество дорожно-транспортных происшествий.

Умные устройства могут использоваться для мониторинга и управления электрическими, электронными и механическими системами, используемыми в различных типах зданий, для домашней автоматизации и автоматизации зданий. Так интеграция интернета вещей с системами энергоснабжений здания может снизить общее энергопотребление, повысить уровень безопасности и другое.

Промышленные IoT устройства (IIoT) [12] пересылают и обрабатывают данные из подключенного оборудования, операционных технологий, своих местоположений и людей. В сочетание с системами мониторинга операционных технологий, IIoT позволяет регулировать и контролировать промышленные системы. Также, такая система может быть использована для автоматического обновления данных по сырью, так как его нехватка может остановить работу всего производства.

В производстве IoT может реализовать бесшовную интеграцию различных производственных устройств [12], способными к коммуникации и связи с другими устройствами в сети. Подобная сетевое пространство способно предоставить полный контроль над производственным процессом, оснащением, увеличить качестве и скорость производства, оптимизировать или усовершенствовать производимую продукцию в реальном времени, увеличить безопасность и надежность производства. При совмещении IIoT систем с системами энергоснабжения можно уменьшить энергопотребление. Промышленный IoT в сфере производства может создать настолько большую ценность, что в конечном итоге приведет к четвертой промышленной революции.

В сельском хозяйстве умные устройства позволяют собирать данные о температуре, количестве осадков, влажности, скорости ветра, зараженности почвы и других параметрах. Эти данные возможно использовать для автоматизации методов ведения хозяйства, анализа ситуации и принятия оптимальных решений, минимизации рисков и потерь.

Мониторинг и контроль деятельности городских и сельских инфраструктур, мостов, железнодорожных путей, ветряные электростанции являются одной из ключевых путей приложения IoT [10]. Различные инфраструктуры с использованием интернет вещей могут использоваться для мониторинга любых событий или условий, способных поставить под угрозу безопасность людей. IoT системы могут принести пользу строительной отрасли, снизить затраты, оптимизировать передвижения, повысить качество рабочего дня, внедрить безбумажный документооборот и увеличить продуктивность, эффективно составлять расписание обслуживания и ремонта, координировать действия команд. Аналитика данных в реальном времени может позволить быстрее принимать решения и сохранять деньги.

Использование таких систем на уровне города способно сильно увеличить качество жизни граждан, путем предоставления различной актуальной информации в реальном времени, умном взаимодействии с городом, таком как умная парковка, точное отслеживание передвижения публичного транспорта. Подключение к единой сети устройств способно снизить потребление электричества, распределить нагрузку на сеть.

Военные IoT системы (IoMT) предполагают применений IoT технологии в военной области в целях разведки, наблюдения и других боевых задач. На них сильно влияют будущие перспективы ведений военных действий в городских условиях, включающие в себя использование датчиков, боеприпасов, техники, роботов, биометрических средств для человека и других умных технологий, используемых на поле боя.

Интернет вещей часто применяется для идентификации продуктов [12] и посылок. Для этого используются специальные QR коды или NFC метки, которые не выполняются какой-либо работы, однако содержат в себе уникальный идентификатор, позволяющий явно определить информацию о продукте или посылке. Сами по себе такие метки не являются частью интернет вещей, однако широко применяются как посредники цифровых взаимодействий. Такие матки защищены от подделки. QR коды содержат специальные водяные знаки, а NFC метки способны зашифровать свое содержимое.

## Сети интернет вещей

Для функционирования IoT сетей необходимы сами сети, по которым происходит передача информации. Существует большое количество технологий, способных обеспечить такую сеть, как проводных, так и беспроводных [8].

Каждое из устройств в сети должно иметь свое уникальное имя для идентификации. В оригинальном концепте IoT сетей в качестве идентификаторов использовались FRID метки, но сейчас чаще всего применяются IP адреса, или mac адреса устройств. Их использование позволяет объединять в сеть любые устройства, имеющие доступ к этой сети. При такой адресации умные устройства не могут взаимодействовать между собой, потому их объединяют агенты – централизованные IoT серверы (хабы).

Помимо интернета в сетях интернет вещей активно используются различные беспроводные модули малого радиуса действия:

* Bluetooth сеть – предполагает построение ячеистой сети с использованием модулей Bluetooth Low Energy (BLE) с увеличенным числом узлов и стандартизированным прикладным уровнем;
* Light-Fidelity (Li-Fi) – предполагает использование беспроводной технологии, аналогичной Wi-Fi, но передающую информацию волнами в видимом свете для увеличения пропускной способности;
* Near-field communication (NFC) – специализированный протокол связи, позволяющий двум устройствам обмениваться данными в диапазоне от 50мм до 6см. Обычно применяется в месте со специализированными NFC метками;
* Радиочастотная идентификация (RFID) – технология, использующая электромагнитное поле для считывания данных со специальных меток-тегов;
* Wi-Fi – технология для локальных сетей, описанная стандартом IEEE 802.11, в котором устройства могут обмениваться данными через общую точку доступа, или напрямую между устройствами;
* ZigBee – протокол связи для локальных сетей, описанный стандартом IEEE 802.15.4, имеющий низкое энергопотребление, низкую скорость передачи, низкую стоимость и высокую пропускную способность;
* Z-Wave – беспроводный протокол связи, используемый в основном для домашней автоматизации;

Среднего радиуса действия:

* LTE-Advanced – высокоскоростной протокол связи для мобильных сетей. Является расширением стандарта LTE, имеющим расширенное покрытие, более высокую пропускную способность и меньшую задержку;
* 5G – современный протокол связи в мобильных сетях, позволяющий достигнуть высоких требований по скорости и пропускной способности, а также позволяющий подключать большое количество IoT устройств;

Сети среднего типа действия относятся к мобильным сетям и в городах они уже развернуты, что позволяет использовать IoT устройства в уже существующей сети.

Дальнего радиуса действия:

* Low-power wide-area networking (LPWAN) – прокол беспроводной сети, предназначенный для обеспечения связи на большом расстоянии с низким потреблением энергии и низкой скоростью передачи данных.
* Very small aperture terminal (VSAT) – технология, использующая спутниковую связь и небольшие антенны для узкополосных и широкополосных данных.

Помимо беспроводных сетей часто используется соединение при помощи Ethernet – сетевого стандарта общего назначения.

## Проблемы интернет вещей

### Платформенная фрагментация

В IoT среде не установлено никаких стандартов, что приводит к отсутствию совместимости устройств разных производителей, и фрагментации платформы. Большое количество разнообразий, как в аппаратной части, так и в программном обеспечении, усложняет разработку приложений, способных работать совместимо с несовместимыми технологиями. Это выражается также на закрытости IoT экосистем различных поставщиков, что сильно затрудняет их связь с аналогичными системами.

Помимо сегрегации проблему составляется аморфная вычислительное природа интернет вещей, так как патчи и устранения ошибок и уязвимостей в программном обеспечении устройств часто не достигают устройств пользователей.

### Безопасность

По мере увеличения числа различных интернет устройств возникают новые потенциально уязвимые месте. Устройства со слабой системой защиты могут служить точками доступа для злоумышленников [9], что позволит портить, перепрограммировать устройства. Недостаточная защищённость потоков информации в IoT сети может подвергать личные данные пользователя риску хищения. Само же устройство может быть точкой входа в интернет и представлять проблему уже все локальной сети. При этом технические ограничения делают умные устройства уязвимее компьютеров, что вынуждает производителей тратить больше ресурсов на совершенствование системы защиты.

Многие системы умных устройств состоят из групп почти идентичных устройств, что усиливает каждую потенциальную ошибку безопасности, распространяя её на все подобные устройства, к примеру, использующие один протокол связи или одного уязвимого посредника.

Часто умные устройства устанавливаются в места, где у злоумышленника есть возможность физического контакта с самим устройством, что сильно увеличивает риски взлома.

При этом количество интернет устройств растет, и внедряется во все сферы жизни, во многих городах устанавливают большое количество инфраструктурных умных устройств, что улучшает качество жизни, но и увеличивает проблемы безопасности, так как умный гаджет для дома можно выключить в случае кибератаки, но нельзя просто выключить, к примеру, умный счетчик, умный светофор или кардиостимулятор.

### Конфиденциальность

Умные сети интернет вещей обычно собирают, обрабатывают и передают информацию, и это не создает проблем в случае с домашней автоматизацией, с локальной сетью, однако многие умные устройства устанавливаются в публичные места, и собирают информацию о людях без их согласия. Так на улицах устанавливают умные камеры, отслеживающие информацию о проходящих мимо людях, умные турникеты, двери, датчики и другое способно в той или иной степени отслеживать перемещение человека. В автомобили встраивают GPS модули для улучшения работы карт, однако данные о положении автомобиля также передаются производителям этих машин. Также личная техника может записывать аудио и видео и передавать их на сервера производителя без какого-либо согласия пользователя, маскируя их под условия пользования. Таким образом использование умных устройств может нарушать право на личную жизнь человека, и с начала зарождения IoT сетей данная проблема не находит какого-либо решения.

Безопасность информационных потоков в IoT сетях также может стать проблемой для личных данных пользователя [9]. Часть устройств использует безопасные потоки, с применением, например, HTTPS, однако большая часть устройств в качестве информационной магистрали использует более быстрые, простые или дешевые протоколы, это обусловлено ограниченным ресурсом автономности, а следовательно многие устройства не могут позволить себе дорогую операцию установки HTTPS соединения. Также такие устройства часто не могут позволить себе зашифровывать и расшифровывать данные, так как современные методы шифрования требовательны к вычислительным ресурсам система. Единственным надежным способом в таком случае остается соединение при помощи кабеля, но такой способ могут использовать только некоторые средства домашней автоматизации и производственные сети.

### Стандартизация

Ключевую роль в развитии любой отрасли занимает разработка и применение стандартов, позволяющая развивать уже существующие технологии, а не создавать новые. На поприще умных устройств не существует единого стандарта в их создании разработке, проектировании интерфейсов, используемых технологиях. Отсутствие каких-либо стандартов приводит к тому, что многие производители стремятся создать свой протокол общения в IoT, свою закрытую экосистему устройств. Для конечного пользователя подобные систему позволяют легко интегрировать в свою среди устройства одного производителя, однако устройства разных производителей практически никогда не соединимы, так как их системы являются закрытыми к расширению.

Решением данной проблемы может быть создание специализированного комитета, по стандартизации именно решений в сфере умных устройств, однако большое количество решений, а также разноуровневое взаимодействие устройств делает подобную стандартизацию невозможной или даже не желательной.

# Постановка задачи

Для управления, настройки и автоматизации умных устройств в составе IoT сети необходим специализированный сервер, а также интерфейс-приложение для настройки, и визуального отслеживания работы умных устройств, для этого необходимо спроектировать и разработать трехзвенное приложения для отслеживания и управления умными устройствами и построения IoT сетей. Оно должно включать в себя:

* сервер, выступающий центральным звеном в сети;
* базу данных, для хранения информации клиентов и рабочей информации сервера;
* клиент, для настройки сервера, отслеживания состояния различных устройств, управления ими.

От сервера необходима скорость, безопасность, возможность тонкой настройки управляемых им устройств, возможность быстрого развертывания на персональных компьютерах, а также мини одноплатных компьютеров, к примеру Raspberry Pi 4. Также, так как данный сервер должен координировать работу умных устройств и хранить их информацию необходимо минимизировать задержки.

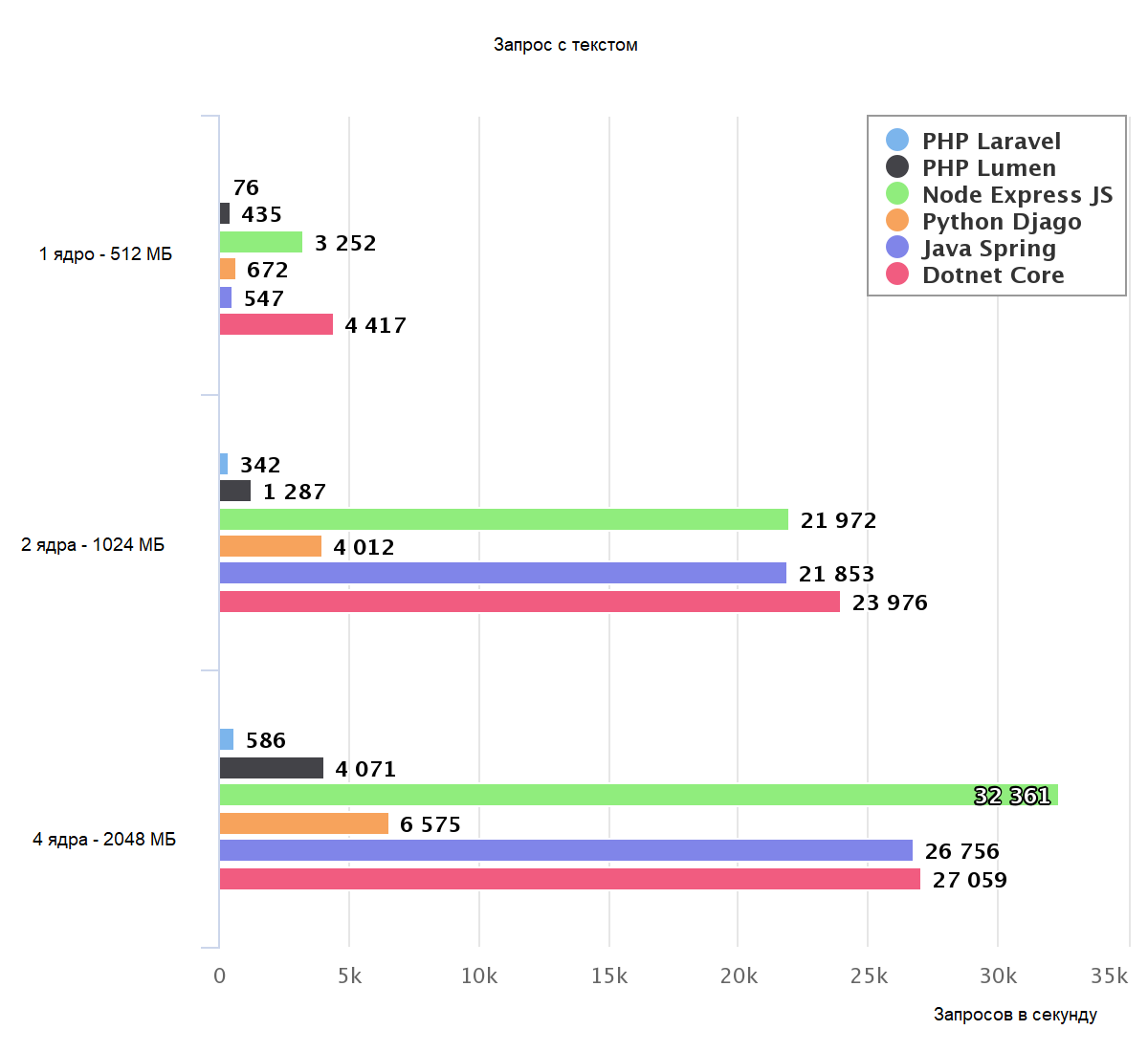
От клиентского приложения требуется визуализация данных сервера, отслеживаемых и управляемых устройств, визуальные интерфейсы для настройки и управления устройствами, желателен современный внешний вид и дизайн.

# Сравнение и выбор инструментов разработки сервера

## Язык и фреймворк

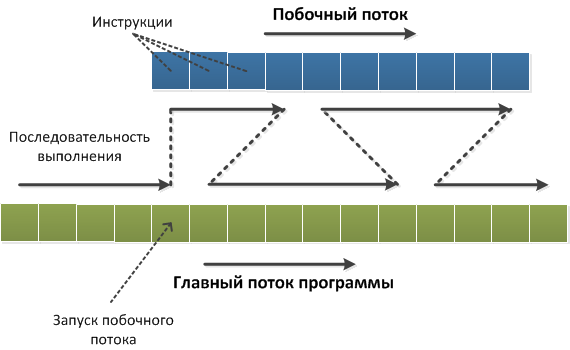
Исходя из задач, стоящих перед сервером, следует, что главными критериями выбора языка, а также, если необходимо, фреймворка, являются скорость обработки входящих запросов, простота внедрения и легковесность.

На данный момент существует огромное количество языков программирования, а также фреймворков для них, однако между ними стоит большая разница в быстродействии, масштабируемости, инструментах разработки.

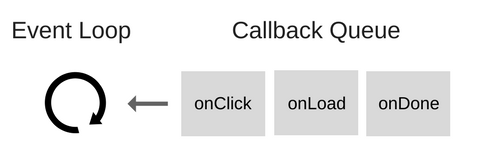
  
Рисунок 1 – Количество выполненных запросов в секунду для различных фреймворков

Как видно из рисунка 1 [1], популярные фреймворки, а именно PHP и Python сильно ограничены по быстродействию. Так получается из-за того, что языке не имеют встроенной возможности выполнять запросы асинхронно, что очень важно для сервера, а также имеют очень плохую общую производительность, однако данные языки имеют низкий порог вхождения из-за своей простоты.

В то же время Express JS, Spring и Dotnet Core имеют сравнительно высокую производительность, благодаря возможности исполнения запросов асинхронно. При этом языки имеют различный механизм асинхронности. Так C# и Java используют систему потоков, которые, в случае наличия у процессора более одного ядра, позволяют выполнять программные коды параллельно главному потоку программы, однако и при наличии одного ядра выполнение программы в несколько потоков возможно. В этом случае процессор по определённым алгоритмам переключается с одного потока на другой, записывая контекст выполнения текущего, и заменяя на сохраненный контекст другого потока, как это показано на рисунке 2 [2].

  
Рисунок 2 – Асинхронные потоки на одном ядре

Асинхронность JavaScript основывается на некотором бесконечном цикле, называемом циклом событий, в котором находится программа. Далее в некоторых точках программы происходит добавление запроса на выполнение некоторого кода в стек запросов (рис 3) [3], чаще всего для этого используют систему колбеков – функций, исполняемых в случае происхождения некоторого действия. При этом длинные цепочки команд не блокируют программу, а приостанавливаются и передают управление другим.

  
Рисунок 3 – Асинхронный механизм JavaScript

Такая реализация позволяет выполнять полностью асинхронный код по запросу, и в случае наличия только одного ядра процессора такая реализация асинхронного хода программы справляется лучше, нежели потоки.

Также, можно размножить процессы программы на JavaScript, в этом случае быстродействие будет линейно масштабироваться с ростом количества ядер процессора, что видно на рисунке 1, однако такой поход требует особую архитектуру приложения, так как процессы, являясь изолированными, практически не могут передать информацию между собой.

Исходя из вышесказанного и учитывая, что целевая платформа – одноплатные компьютеры, на которых в среднем имеется несколько ядер, но существуют ограничения по количеству оперативной памяти, хорошим выбором будут Spring и Dotnet Core, при этом последний немного выигрывает по скорости.

## Выбор системы управления базой данных

Существует огромное количество систем управления базами данных (СУБД), они все выполняют одну задачу, однако сильно отличаются по количеству затрачиваемых ресурсов, модели распространения, быстродействия, а также функционалу.

Каждая из СУБД рассчитана на свой формат хранения данных, которые на данный момент делятся на две группы: SQL и NoSQL.

SQL формат основывается на использовании плоских таблиц – двумерных массивов данных, состоящих из столбцов и строк. Множестве столбцов формирует атрибуты таблицы, а множество строк – тело таблицы, где каждая строка — это отдельная запись, состоящая из соответствующих полей. Каждая строка таблицы уникальна и имеет свой уникальный номер, именуемый ключом. Таблицы между собой могут иметь внешние связи, реализующиеся через внешний ключ.

NoSQL формат включает в себя все остальные варианты хранения данных. Это могут быть JSON документы, страницы с ссылками, объединённые связями, формирующие граф или дерево. Такой формат проигрывает SQL в производительности и применяется только для некоторых узкоспециализированных задач.

На сегодняшний день все SQL СУБД используют стандартизированные алгоритмы работы с информацией, поэтому имеют сравнительно равную производительность и быстродействие, NoSQL же, из-за структуры данных, сильно проигрывают SQL СУБД в скорости, а потому не применимы для поставленной задачи.

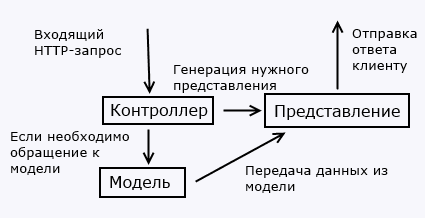
СУБД чаще всего формируются в виде сервера, общение с которым происходит через сокеты. Это позволяет выносить сервера с данными на отдельную физическую машину, однако для поставленной задачи это не нужно, потому хорошим выбором будет SQLite, которая не работает в виде отдельного процесса, а существует в виде набора библиотек с открытым исходным кодом, то есть является встраиваемой в приложения. Подобный формат накладывает ограничения на возможные операции, однако абсолютно не нагружает процессор и оперативную память отдельным процессом.

## Архитектура приложений

Выбор архитектуры сервера важен, так как от этого полностью зависит возможность его развития в том, или ином направлении.

На данный момент для серверных приложений обычно применяются MVC и REST архитектуры.

MVC (Model-View-Controller) – архитектура, при которой приложение разделяется (рис 4) [4] на компоненты с данными, пользовательским интерфейсом и управляющей логикой, именуемые моделью, представлением и контроллером. При такой архитектуре части приложения развиваются отдельно, и часто слабо связаны между собой при помощи интерфейсов или инверсии зависимости, что позволяет модифицировать отдельные части приложения без опасности создать проблемы в других местах.

  
Рисунок 4 – Схема архитектуры MVC

REST (передача состояния представления) – архитектурный стиль, для создания распределённых веб сервисов. В некоторых случаях приводит к повышению производительности и упрощению архитектуры. Сама архитектура выражается в шести ограничениях [5]:

* Клиент-серверная модель взаимодействия

Основное ограничение, позволяющее упросить серверную часть и улучшить масштабируемость, также такое разделение позволяет обособленно разрабатывать клиентские приложения

* Отсутствие состояния

Серверное приложение не должно хранить информацию о текущем состоянии клиентов, то есть все запросы должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса. Состояния сессии при этом могут храниться на стороне клиента и передаваться вместе с запросами, например, для аутентификации.

* Кэширование

Клиенты и промежуточные узлы могут кэшировать ответы сервера для увеличения производительности. При этом сервер должен помечать ответы как кэшируемые или некэшируемые с целью предотвращения сохранения пользователями устаревших или неверных данных.

* Единый интерфейс

Фундаментальное правило REST архитектуры. Включает в себя: идентификацию ресурсов с использованием URI. При этом сами ресурсы могут отличаться от представлений, возвращенных клиентам; Возможностью манипуляции ресурсами через представление – метаданных ресурса должно быть достаточно для его модификации; “Самоописываемость” сообщений – данных в ответе должно хватать для работы с ресурсами.

* Возможность слоев

Серверное приложение может быть разделено на слои, и клиент не должен знать, общается он с сервером напрямую или с его промежуточным узлом.

Еще одним подходом к архитектуре и разработке WEB сервисов является, удаленный вызов процедур (RPC) – позволяющий программам вызвать функции и процедуры на сервере. Такой способ позволяет более гибко управлять сервером, но создает большие проблемы с нагрузкой на сервер, а также при передаче данных, так как нет единого протокола или стандарта. Особенно данная проблема проявляется в случае когда происходит связь между разными операционными системами, в которых одни и те же данные могут представляться по разному, или между разными языками, где ответ, корректный на одном языке программирования является некорректным на другом. Решением данных проблем могут выступать расширения протоколов, устанавливающие свою спецификацию и стандарт, к примеру JSON-RPC, ограничивающий передачу всех данных в формате JSON, или SOAP, требующий передавать массивную структуру вспомогательных элементов, превышающий стандартный заголовок HTTP или HTTPS запроса, в добавок к исходным данным. Все это делает RPC плохим выбором для IoT системы.

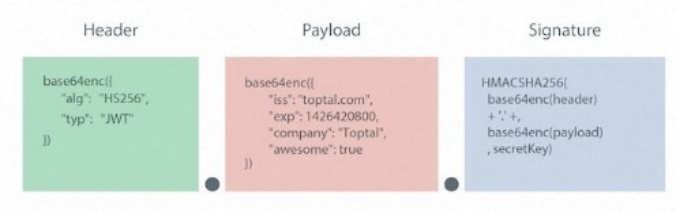
MVC архитектура подразумевает монолитный вид приложения, где сервер самостоятельно генерирует своё визуальное представление, и сам же его обрабатывает. Подобный вариант исключает возможность работы сервера без представления, а также не дает использовать сторонние клиенты, что не подходит под условия поставленной задачи.

REST позволяет серверу работать самостоятельно, а потому является хорошим вариантом для поставленной задачи. Также это позволит использовать не только HTTP запросы, но и другие транспортные протоколы. В качестве формата представление ресурсов выбран JSON, так как с ним легко работать при известной структуре объектов.

## Авторизация и аутентификация

Так как система должна быть защищенной, а также работать с пользовательскими данными, в ней должна быть предусмотрено система аутентификации и авторизации. Для авторизации пользователя можно использовать классическую авторизацию по логину и паролю, а также OAuth для интеграции учетной записи с другими сервисами. После этого необходимо подписать сессию пользователя, для этого часто применяется система cookies, но она не применима, так как сервер использует REST архитектуру, а следовательно, должен позволять интеграцию не только в браузер.

Для REST сервисов обычно применяются JSON Web Token (JWT), к тому же это позволит взаимодействовать с сервером умным устройствам. JWT это JSON файл, содержащие (рис 5) [6] данные о сессии пользователя, издателе и возможных пользователях токена, время его жизни, а также дополнительные данные. Данный JSON документ шифрует при помощи SHA-256 и, следовательно, больше не может быть прочитан никем, кроме владельца ключа.

  
Рисунок 5 – Структура JWT

Обычно для пользователя используется сразу два токена в целях безопасности. Один токен для доступа к ресурсам, имеет малое время жизни, в зависимости от нужд приложения, от пяти минут до нескольких часов. Второй токен необходим для генерации новой пары токенов по уже существующей, имеющий долгий период жизни, от пары дней до месяцев. При этом несколько нарушается идея JWT и REST, согласно которой на сервере не должна храниться информация о сессии пользователя, однако такой способ позволяет сбрасывать не действительные или скомпрометированные токены, что уменьшает риски, связанные с их хищением.

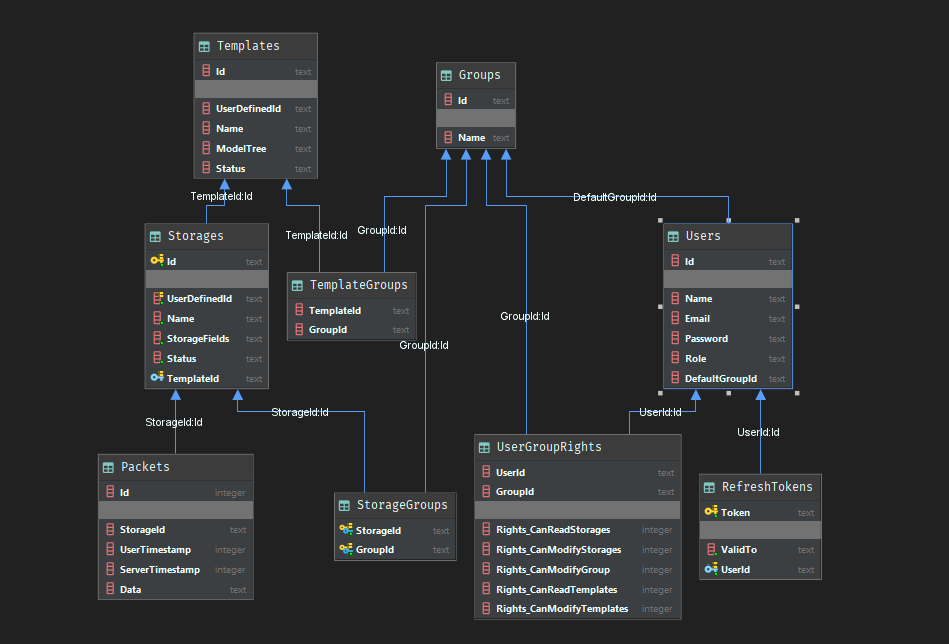
Система из двух токенов удобна для авторизации пользователей, но умные устройства могут долгое время не иметь связи с серверов, а также могут не иметь ресурсов на дополнительные запросы с обновлением токенов, потому для них лучше использовать классическую схему с одним токеном, а также, возможно, бесконечным, или очень долгим временем жизни.

# Моделирование базы данных, точки входа

Из задач, поставленных в пункте 2, следует необходимость в наличии сущностей для авторизации, хранения данных, хранения моделей, пользователей. Сформированы следующие сущности:

1. Users (пользователи) – пользователи системы, зарегистрированные в системе. Пользователь может быть как разработчиком системы, так и просто пользователем. Каждый пользователь также является точкой входа в приложения, посредством аутентификации по имени и паролю пользователя, и генерированием двух JWT токенов.
2. RefreshTokens – специализированные токены для обновления пары токенов для пользователя.
3. Templates – шаблоны хранилищ данных для умных устройств. Шаблоны хранятся в виде JSON объекта, имеющего древовидную структуру. Такой способ хранения позволяет тратить меньше времени на сбор данных по базе данных, накладывая единственное ограничение на невозможность индексирования и внедрения в запросы полей объекта стандартными средствами SQLite.
4. Storages – типизированные хранилища данных. Каждое хранилище может представлять собой одно умное устройства, или сразу несколько. Оно отслеживает текущее состояние объектов, проверяет входящие данные на соответствие типам, указанным в JSON объекте StorageFields. Хранилища могут быть анонимными, а могут быть сгенерированными из шаблонов.
5. Packets – представляет собой историю приходящих на хранилище данных, также представленных в виде JSON объектов. Важно, что на сервер не обязательно отправлять полный пакет данных, а достаточно только выбранных его полей. В этом случае создастся дубликат последнего пакета и в нем заменятся данные на те, что пришли на сервер.
6. Groups – цетральная часть всей системы. Группы совмещают пользователей, хранилища и шаблоны связями многим-ко-многим. При этом сами группы являются точками входа в приложения умных устройств. Такой способ связи сущностей позволяет тонко настраивать групповые политики, правила пользования и доступа, однако оказывает сильное влияние на производительность базы данных, так как для получения, к примеру, списка хранилищ пользователя, необходимо объединить 5 таблиц. При правильном использовании JWT токенов, и продуманным контроллерам этой проблемы можно избежать.
7. TemplateGroups, StorageGroups, UserGroupRights – служебные таблицы для связи много-ко-многим. Последняя таблица помимо служебной информации включает данные о возможностях пользователя в пределах группы.

Внешние связи и ключевые атрибуты показаны на рисунке 6.

  
Рисунок 6 – Структура базы данных сервера

# Конечные адреса

Одним из плюсов REST сервисов является понятная спецификация программного интерфейса приложения – API. Он представлен в виде веб адресов конечной точки – endpoint. Это позволяет удобно и быстро совершать запросы на сервер удобными URI адресами, при этом данные запроса могут быть составной частью URI адреса, а могут быть переданы в теле запроса в удобном для клиента виде, обычно JSON или XML.

Полученный сервер включает в себя большое количество таких адресов, предназначенных для аутентификации, работы с группами, шаблонами, пользователями, пакетами и хранилищами.

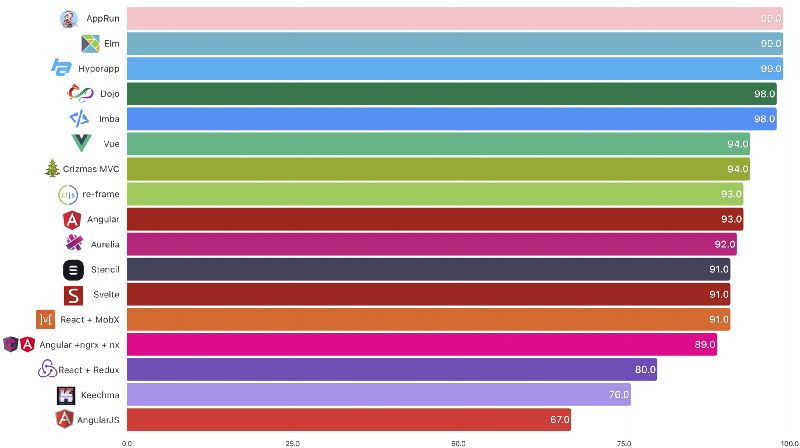
Также для удобства использования сгенерирована документация (рис 7) на основе Swagger API – специализированная документация для REST сервисов.

  
Рисунок 7 – Документация API приложения

# Клиентское приложение

## JavaScript библиотеки

На данный момент существует огромное множество JS библиотек, использующихся для одной и той же цели – отрисовки клиентского приложения или вебсайта (их сравнительные оценки можно увидеть на рисунке 8 [7])

  
Рисунок 8 – Сравнительная оценка JS библиотек (больше – лучше)

При таком количестве, самыми используемыми и лучшими считаются три из них: React, Vue и Angular, при этом последний сильно проигрывает в производительности, так как использует двустороннюю привязку компонентов с сервером, в то же время Vue и React используют виртуальное DOM дерево, что позволяет не перерисовывать страницы целиком, а только перерисовать нужные части, при этом без потерь на операции доступа к элементам реального DOM дерева, являющихся очень долгими.

Благодаря этому, возможно создавать Singe Page Application (SPA) популярные сегодня. Такие приложения ничем не отличаются от стандартных приложений на компьютере, но не требуют прямого скачивания, так как инструменты для отрисовки и выполнения в виде браузера уже есть у пользователя, и ему необходим только сравнительно небольшой размет исполняемых файлов. Эти файлы больше классических веб страниц, однако единоразово скачав их при входе в приложение (на сайт) у клиента уже будут все возможные для отображения части приложения, и последующие операции с сервером в основном состоят из передачи данных, вес которых редко превышает 5 КБ, что сильно уменьшает время загрузки и время отклика, улучшает отзывчивость интерфейса, ускоряется переход между элементами.

Виртуальное DOM дерево состоит из компонент, каждая из которых имеет свое состояние и может легко передавать данные сверху вниз. Для передачи же данных снизу вверх предусмотрена система ссылок, но она используется не часто. Обычно применяются глобальные хранилища состоящий приложения, которые при помощи паттерна внедрения зависимостей, внедряются в компоненты, предоставляя доступ к отслеживаемым данным и API. В Vue такая система существует как часть библиотеки, в React для этого используются сторонние библиотеки: Redux или MobX. Они выполняют одну и ту же цель, имеют схожий интерфейс, однако достаточно сильно различаются по производительности, что видно на рисунке 8.

Также для создания SPA приложения необходим роутер, который будет сопоставлять URI с окнами приложения. И Vue, и React для этого применяют сторонни библиотеки роутеры, с одинаковым механизмом действия.

Также такие библиотеки используют средства версии стандарта EcmaScript 6, однако во многих браузерах он до сих пор не поддерживается. Для этого необходимо заменять неподдерживаемые части специальными конструкциями, реализующими такое же поведение, однако использующие только средства EcmaScript 5, для этого применяется специальный JS компилятор Babel.

В последствие получается огромное количество зависимостей (в среднем 15 тысяч) между файлами кода, которые необходимо правильно и в нужном порядке импортировать в файлы программы. Для такой сборки библиотек в один файл, а также очистки от лишних зависимостей, необходима специальная система сборки. Чаще всего для этого используется webpack [7].

## CSS-библиотеки

CSS является очень старой технологией и почти не развился за время своего существования. В нем нет функций, методов, циклов, привычных для других языков, однако только с помощью него можно стилизовать компоненты, обрисовывающимися браузерами. В больших проектах сложно поддерживать CSS код, потому используются разные методики для улучшения кода, например, разделение кода на маленькие файлы, каждый из которых стилизовал определённо часть страницы или компонент, это упрощало поддержку и редактирование кода. Также сейчас активно применяется BEM методология, которая рассматривает классы стиля как объекты, что позволяет эффективно пере использовать код.

Для дальнейшего улучшения CSS кода используются препроцессоры – специальные языки и компиляторы к ним, расширяющие возможности языка, и транслирующие свой код в исходный CSS код. Препроцессоры включают в себя расширенные операторы, интерполяции, функции и методы, а также примеси. Самыми часто используемыми являются препроцессоры SASS, LESS и Stylus [8].

Другой стороной развития CSS, появившейся с распространением React и Vue стал подход CSS-in-JS, который представляет CSS код в виде классов с параметрами, дублирующие параметры в исходном CSS коде, а затем данные классы добавляются в виртуальное DOM дерево и автоматически генерируют класс, добавляющийся в итоговый CSS файл. Такой подход позволяет инкапсулировать код стиля в файла его компонента, что упрощает поддержку и развитие проекта.

В ходе развития Web программирования появлялись новые возможности и CSS параметры. Браузеры вводили эти параметры до их появления в стандарте и добавляли к меняла их названия, откуда появились вендор префиксы свойств, свойственные для каждого браузера. Для использование некоторых параметров CSS необходимо дублировать их для каждого браузера с использованием его вендор-префикса, для этого необходим пост обработчик, который за разработчика будет дублировать эти префиксы. Самым часто используемым является autoprefixer. Он позволяет писать стили без префиксов каждого браузера, и добавляет их на этапе компиляции.

## Пакетный менеджер и сборка проекта

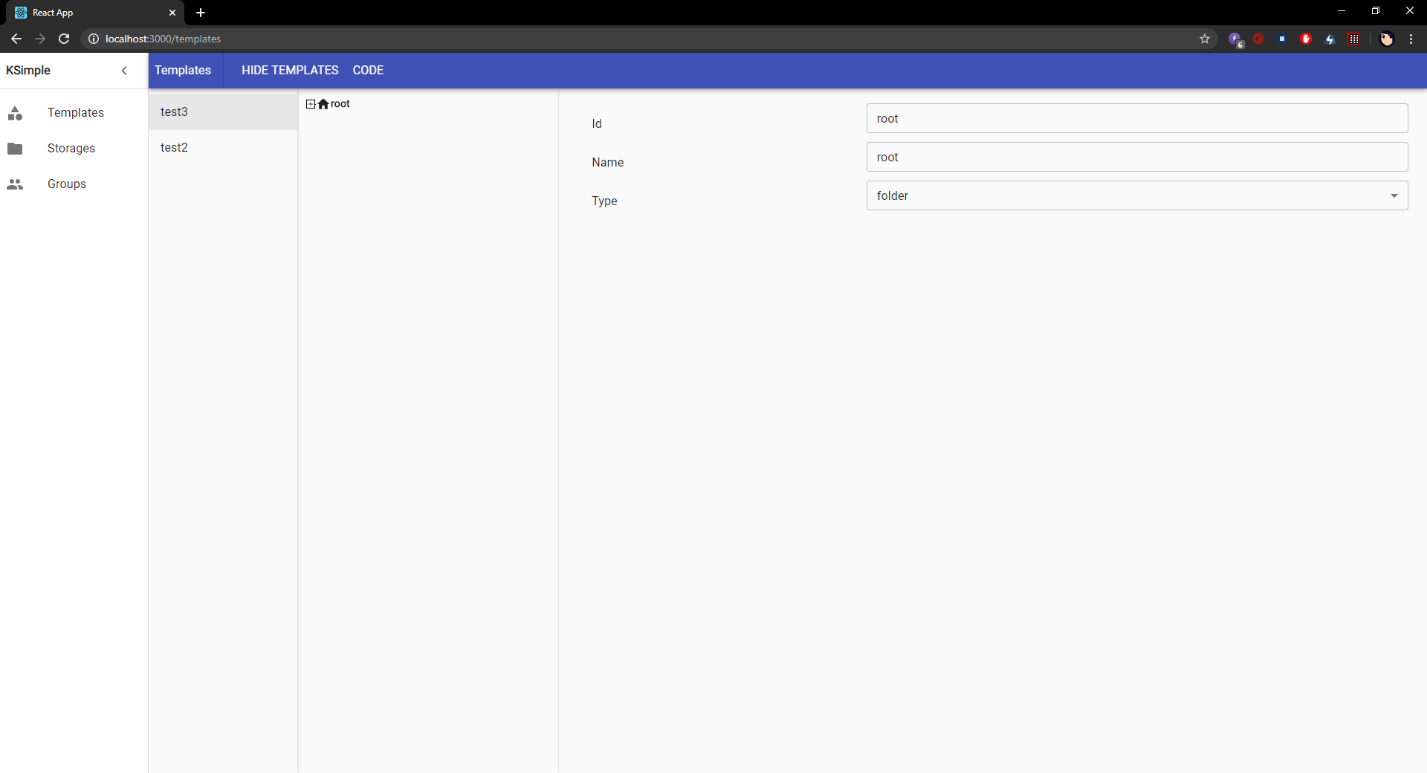
Для отслеживания всех используемых библиотек, плагинов и настроек необходимо приложение, которое будет это делать, а также загружать их и создавать связи. Без него загружать зависимости придётся собственноручно, что негативно сказывается на качестве кода и размере проекта. Самым известным и используемым пакетным менеджером является npm, и надстройка над ним yarn, он же является самым большим реестром пакетов. Он позволяет описать используемые зависимости и параметры в специальном файле, находящемся в форме проекта, указав названия и версии библиотек, и загрузить и связать их все одной командой. Также он включает специальный файл, в котором указаны версии библиотек, используемые в момент разработки, что позволяет не допускать проблем с конфликтом версий при переносе проекта на разные рабочие станции. Это также позволяет не хранить в репозитории проекта исходные коды библиотек, а только их описание.

При подготовке проекта к использованию необходимо сжать, обрезать неиспользуемый и собрать все файлы исходного кода в один. Это позволяет сильно экономить на размере передаваемых файлов, а также уменьшить нагрузку на оперативную память клиента. Для автоматизации этого процесса можно использовать плагины webpack, а именно UglifyJsPlugin, который проводит обфускацию и минификацию JS кода, а также DedupePlugin, который находит одинаковые файлы в проекте и исключает их из итоговой сборки. Неиспользуемые файлы и библиотеки webpack исключает по умолчанию.

Для автоматизации использования CSS препроцессоров, пост обработчиков и Babel, необходимо добавить в webpack загрузчики.

## Итоговое клиентское приложение

Итоговое приложение построено на основе библиотеки React, с использованием CSS-in-JS подхода в виде SPA приложения. Приложение позволяет пользователю зарегистрироваться или авторизоваться, и создавать и изменять шаблоны, а также управлять, просматривать, создавать и изменять хранилища. Пример интерфейса представлен на рисунке 9.

  
Рисунок 9 – Пример интерфейса приложения

После настройки хранилищ пользователь может использовать сгенерированный JWT токен группы, в которой состоит необходимое ему хранилище и отправлять данные в виде JSON документов с данными зарегистрированных полей. Приложение само отслеживать типы полей и предупреждает в случае несоответствия типов. Также пользователю не нужно отправлять весь пакет данных приложения целиком. Он может ограничиться только необходимыми ему полями, приложение отследит изменяемые поля и создаст новый пакет данных, сохранив неизмененные данные. Также в приложении предусмотрены средства автоматизации в виде событий, привычных многим программистам. Данные события позволяет описать в виде кода на JavaScript порядок действий, которые необходимо произвести после активации того или иного события. На данный момент поддерживается событие вставки и считывания данных, а также события по таймеру и по вызову специального URI.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были достигнуты поставленные цели, а именно: проведено исследование современных направлений развития и проблем IoT сетей, проведено изучение современных WEB технологий, разработана и создана система для отслеживания, разработки и управления умными устройствами в составе сети.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Web REST API Benchmark on a Real Life Application [Электронный ресурс] – Режим доступа: <https://medium.com/@mihaigeorge.c/web-rest-api-benchmark-on-a-real-life-application-ebb743a5d7a3> (Дата обращение 02.04.2020)
2. Лекция 08. Многопоточность и графика Application [Электронный ресурс] – Режим доступа: <http://shtanyuk.tk/edu/nniit/java-new/html/08.html> <https://habr.com/ru/company/ruvds/blog/340508/> (Дата обращение 08.04.2020)
3. Введение в ASP.NET MVC 5 [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/mvc5/1.1.php> (Дата обращение 08.04.2020)
4. Representational State Transfer (REST) [Электронный ресурс] – Режим доступа: <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm> (Дата обращение 08.04.2020)
5. JSON Web Токен Учебник: Пример с Laravel и AngularJS [Электронный ресурс] – Режим доступа: <https://github.com/Flibberty-GEA/play-silhouette-slick-seed/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B2%D0%BE%D0%B4-%22JSON-Web-Token-Tutorial:-An-Example-in-Laravel-and-AngularJS%22> (Дата обращение 08.04.2020)
6. Webpack [Электронный ресурс] – Режим доступа: <https://webpack.js.org/> (Дата обращение 08.04.2020)
7. CSS-препроцессоры [Электронный ресурс] – Режим доступа: <https://mrmlnc.gitbooks.io/less-guidebook-for-beginners/content/chapter_1/css-reprocessors.html> (Дата обращение 08.04.2020)
8. Charlier M. Light A. Lui A. Designing Connected Products: UX for the Consumer Internet of Things – Издатель: O'Reilly Media; 1 edition (18 мая, 2015) – ISBN: 1449372569
9. Gilchrist A. IoT Security Issues – Издатель: De Gruyter (Январь, 2017) – ISBN: 9781501505621
10. Al-Turjman F. Intelligence in IoT-enabled Smart Cities – Издатель: CRC Press (9 января, 2018) – ISBN: 9781138316843
11. Khan J. Y., Yuce M. R. Internet of Things (IoT): Systems and Applications – Издатель: Jenny Stanford Publishing (1 октября, 2019) – ISBN: 9814800295
12. Zhang Y., Tao F. Optimization of Manufacturing Systems Using the Internet of Things – Издатель: Academic Press (21 октября, 2016) – ISBN: 9780128099100