СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данная дипломная работа посвящена проектированию веб-сервиса для удаленного доступа к программно-аппаратной платформе домашней автоматизации.

Развитие домашней автоматизации начиналось от одного компьютера в комнате до сети вычислительных устройств расположенных по всему дому. Многие из них – устройства, непосредственно общающиеся с пользователем, такие как планшеты, ноутбуки и т.д. Однако всё больше и больше домашняя сеть начинает содержать устройств специального назначения – «вещи» – что превращает домашнюю сеть в сеть не только для пользователей, но и для своих устройств – сеть «вещей».

С начала 2010-х годов в результате повсеместного распространения беспроводных сетей, появления облачных вычислений, освоения программно-конфигурируемых сетей и развития технологий межмашинного взаимодействия (Machine-to-Machine) начинается системное внедрение практических идей «интернета вещей» в сфере ИТ.

Новое понятие Internet - Internet of Things (IoT) - это существующая сеть Интернет, расширенная подключенными к ней вычислительными сетями различных устройств, физических предметов или вещей, которые могут самостоятельно организовывать разнообразные модели подключения или общения («Thing – Thing», «Thing – User» и «Thing - Web Object»).

Термином «вещь» в IoT обозначаются интеллектуальные, т.е. «умные» предметы или объекты, к которым относятся датчики или приводы, снабженные микроконтроллером с ОС реального времени со стеком протоколов, памятью и устройством связи, встроенные в различные объекты, например, в электросчетчики, газовые счетчики, счетчики потребления холодной и горячей воды, датчики давления, вибрации или температуры и т.д.

«Умные» объекты могут быть организованы в вычислительную сеть физических устройств, подключенных через шлюзы (хабы или специализированные IoT платформы) к традиционной сети Интернет.

На технологическом уровне IoT – это способ развития инфраструктуры сети (физической основы) Интернет, в которой «умные» вещи самостоятельно, без участия человека, подключаются к сети для удаленного взаимодействия с другими устройствами (Thing - Thing) или взаимодействия с автономными или облачными ЦОДами или DATA-центрами (Thing - Web Objects) с целью передачи данных на хранение, обработку и анализ данных, принятия управленческих решений, направленных на изменение окружающей среды, а также с целью взаимодействия с пользовательскими терминалами (Thing - User) для контроля и управления этими устройствами.

Концепция облачных вычислений возникла в 2006 году. Amazon.com, в то время книжный интернет-магазин, представил Amazon Web Services (AWS), положив начало движению облачных вычислений.

AWS объединяет широкий набор сервисов, таких как вычислительные мощности и хранилища данных. Впоследствии к Amazon.com присоединились Netflix, Microsoft, Google, Apple и IBM, образовав обширный рынок облачных вычислений.

Задача данного программного проекта заключена в реализации контроля всех подключённых устройств с единого, дружественного пользователю web-интерфейса в реальном времени. Поддержке общения между устройствами и решение ими определенных повседневных задач без участия человека. Предоставление пользователю возможности гибкой настройки системы под свои потребности. Подразумевается, что пользователь выбирает тип управляемых устройств среди поддерживаемых системой. Тип устройства определяет конкретные задачи, выполняемые им.

Основные возможности:

– управление различными типами устройств;
– веб-доступ с любого устройства в глобальной сети;
– редактор сценариев работы устройств;
– web-интерфейс с обновлением в реальном времени;
– push-уведомления;
– интеграция со сторонними веб-сервисами (сервис погоды);
– модель безопасности с разграничением доступа между пользователями;
– просмотр статистики работы устройств;
– пользователю доступны данные всех устройств без их сохранения в облаке;
– синхронизация состояния устройств, управляемых непосредственно и через web-интерфейс;
– plug and play (PnP) авто определение новых подключённых устройств.

Web-сервис представляет собой сервер-клиентское приложение, в котором сервер находится на облачной PaaS (Platform as a Service) платформе Heroku. Клиентская часть выполняется в браузере. Клиент выполняет запросы на сторонние сервисы, такие как сервис погоды openWeatherMap, используя его API, а также устанавливает соединение с домашним сервером автоматизации, через который происходит управление устройствами и от которого приходят уведомления о состоянии системы в реальном времени.

Разработка web-приложения производилась с помощью технологий Ruby on Rails, AngularJS, jQuery, Bootstrap, PatternFly, HTML5. Приложение имеет REST (Representation state transfer) архитектурный стиль. Ruby on Rails реализует паттерн MVC (Model-View-Controller), AngularJS – MVW (Model-View-Whatever).

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор аналогов

При проектировании системы были изучены наиболее популярные аналоги.

«MajorDoMo» – это программное обеспечение (ПО), позволяющее компьютеру выполнять функции контроллера домашней автоматики (см. рисунок 1.1).

Рисунок 1.1 – Система «MajorDoMo»

Основные особенности системы – это создание сценариев, GPS-трекинг и реакция на местоположение пользователя, голосовые уведомления и распознание голоса, управление мультимедиа, маркет дополнений, анализ состояния и самодиагностика.

К основным недостаткам системы относятся:

- для полноценной работы требует компьютер с 2 Gb оперативной памяти;
- после установки система требует настройку;

- для программирования скриптов необходимо знание базового синтаксиса PHP, а также списка функций, реализованных в «MajorDoMo» для работы с автоматикой;
- доступ только в локальной сети.

«Home Assistant» – open-source платформа для домашней автоматизации.

Платформа запускается либо на Raspberry Pi3 либо на компьютере (см. рисунок 1.2).

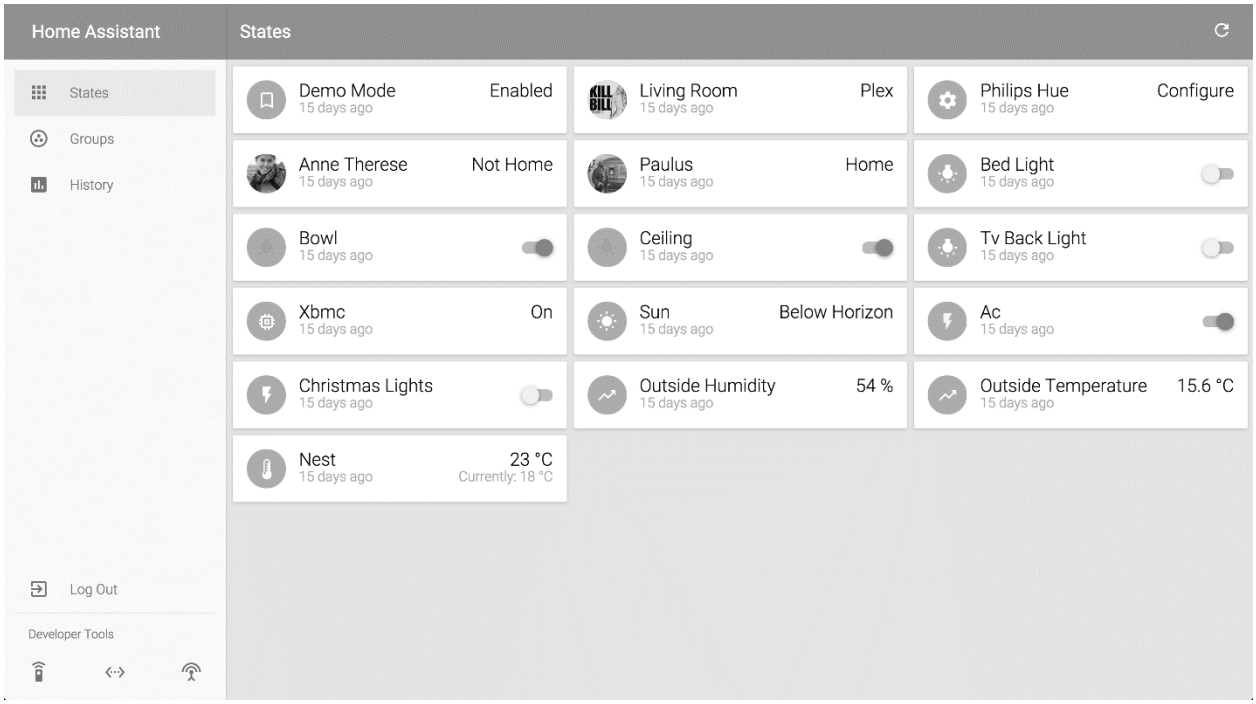

Рисунок 1.2 – Система «Home Assistant»

Основные особенности системы – это работа с устройствами различных производителей, автоматическое обнаружение устройств с последующим отображением в основной группе «Home», frontend построен на библиотеке Polymer, предоставляет WebSocket API и RESTful API.

К основным недостаткам системы относятся:

- для запуска, подключения и настройки компонентов, выключателей, сценариев, групп, триггеров пользователь должен знать YAML;
- для доступа к системе не с глобальной сети необходимо настроить port forwarding на роутере с использованием Dynamic DNS сервисов, также пользователь должен самостоятельно настроить шифрование;

«ThinkingHome» – это платформа домашней автоматизации на .NET (см. рисунок 1.3). Она позволяет реализовывать функционал в виде плагинов, при этом не заботясь о решении базовых задач.

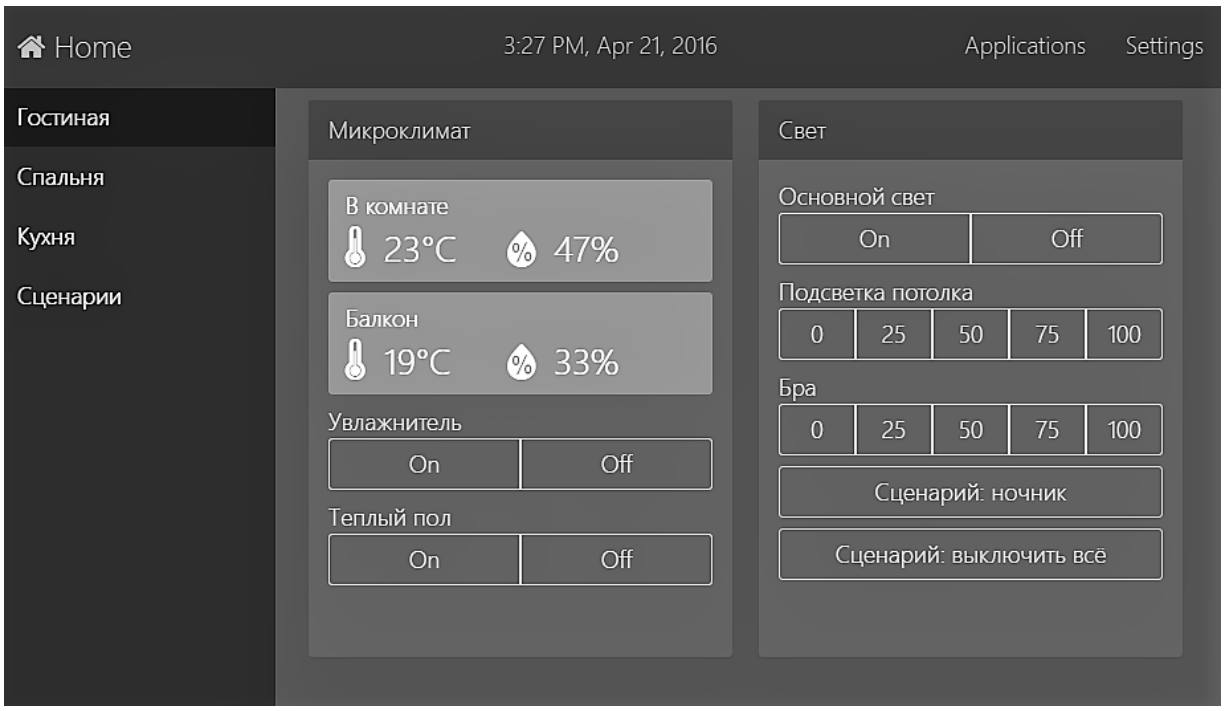

Рисунок 1.3 – Система «ThinkingHome»

При помощи плагинов можно расширять список вероятных событий в системе. В основном вся функциональность системы как раз и обеспечивается системой плагинов. В основном сервис содержит только инфраструктуру для их работы, обеспечивает загрузку и организацию жизненного цикла, предоставляет возможность работы с БД и систему логирования. Веб-интерфейс доступен с любого устройства в домашней сети (компьютера, смартфона, планшета). Он может отображать информацию о прогнозе погоды, расписании автобусов и т.д. Через web-интерфейс доступно как ручное управление домом – управление устройствами со смартфона или планшета, так и управление через сценарии – автоматическое выполнение действий при наступлении определённых событий. Система имеет возможность работать с устройствами от разных производителей. Стандартные плагины осуществляют поддержку MQTT протокола, микроклимата, прогноза погоды, таймера, nooLite. Таким образом пользователю доступны: система плагинов, API логирования, API хранения данных в БД, инфраструктура HTTP API, инфраструктура веб-интерфейса, клиент-серверная шина сообщений, API локализации.

К основным недостаткам системы относятся:

- создание сценариев на ЯП JavaScript;
- система реализована на .NET (доступна только на Windows OS);
- скудный набор готовых плагинов;
- доступ только с устройств в домашней сети;

«OpenHub» – open source система автоматизации на Java (см. рисунок 1.4). Система позволяет интегрировать разные системы и технологии домашней автоматизации в одно единственное решение, которое позволяет

накладывать правила автоматизации и предоставляет единообразный пользовательский интерфейс. «OpenHub» использует Eclipse SmartHome framework. Он написан полностью на Java и использует Apache Karaf вместе с Eclipse Equinox как Open Services Gateway Initiative (OSGi) runtime и связывает с Jetty как HTTP сервер.

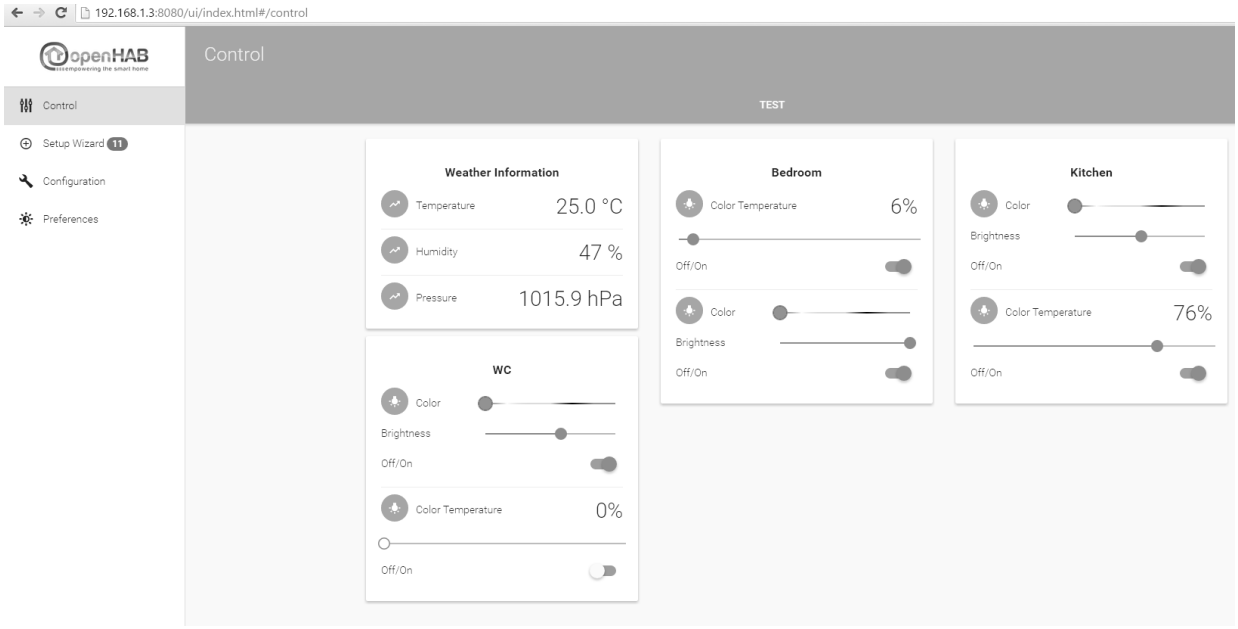

Рисунок 1.4 – Система «OpenHub»

Основные особенности системы: архитектура не зависит от производителя устройств, независима от аппаратного обеспечения и протоколов, единый пользовательский интерфейс, удалённое управление и уведомления, триггеры, запускаемые событиями или таймером, механизм плагинов, сценарии.

Основные недостатки:

- нет механизма plug and play, требует технических навыков в программировании, электрике и пайке.
- упрощённый интерфейс

«The Thing System» – это комплекс программных компонентов и сетевых протоколов для домашней автоматизации. «Домашний» сервер в нем называется steward. Steward написан на node.js и запускается как на обычном компьютере, так и на одноплатном, таком как Raspberry Pi. Дизайн и архитектура системы подчиняется правилу, что IoT должен вращаться вокруг правил, по которым «вещи» следят за событиями, в ответ на которые могут выполнять задачи. Не каждый выбор, опцию нужно предоставлять пользователю. IoT должен быть контекстно зависим, должен действовать на упреждение, а не реагировать в ответ.

Основные недостатки:

- не имеет визуализации (предоставляет только API).

1.2 Обоснование выбора используемого инструментария

Данный проект архитектурно состоит из двух частей: backend и frontend. На стороне backend-а используется Ruby on Rails framework. На стороне, frontend-а используется AngularJS framework. Рассмотрим особенности данных технологий.

1.2.1 Ruby

Ruby является динамически типизированным ЯП. Он характеризуется достаточно сложной, но выразительной грамматикой. Базовая библиотека классов с мощным и разнообразным API является одним из преимуществ этого ЯП. Он включает в себе отличительные особенности языков Lisp, Smalltalk и Perl. Ruby является полностью объектно-ориентированным ЯП, однако в нем также используются процедурный и функциональный стили программирования. Потенциальные возможности по поддержке метапрограммирования позволяют использовать Ruby в создании языков, предназначенных для конкретных предметных областей (domain-specific languages - DSL) [1]. Ruby имеет множество функций: независимую от операционной системы реализацию многопоточности, полностью свободную кроссплатформенную реализацию интерпретатора, отчетливую динамическую типизацию и др.

Простые числовые литералы, значения true, false, nil и другие значения в Ruby – это объекты. Любая функция является методом. В большинстве ЯП, за исключением Ruby, в вызовах функций и методов необходимы круглые скобки. Как правило, для Ruby круглые скобки необязательны и часто отсутствуют, особенно при вызовах методов, не требующих аргументов, что делает эти вызовы похожими на ссылки на поименованные поля или поименованные переменные объекта. Это сделано с определенной целью, так как Ruby строго следит за инкапсуляцией своих объектов – отсутствует доступ к внутреннему состоянию объекта за его пределами. Любой доступ происходит через посредника с помощью метода доступа [1, 2].

Ruby не поддерживает множественное наследование. Взамен он использует мощный механизм примесей. Все классы (напрямую или через другие классы) выведены из класса Object, вследствие этого любой объект может использовать определённые в нём методы (например, class, to_s, nil?). Процедурный стиль тоже поддерживается, однако все глобальные процедуры неявно являются закрытыми методами класса Object [1, 2].

Ruby — мультипарадигменный язык: он поддерживает процедурный стиль (определение функций и переменных вне классов), объектно-ориентированный (всё — объект), функциональный (анонимные функции, замыкания, возврат значения всеми инструкциями, возврат функцией последнего вычисленного значения). Он поддерживает отражение, метапрограммирование, информацию о типах переменных на стадии выполнения [1, 2, 3].

1.2.2 Ruby on Rails

Ruby on Rails (или коротко Rails) – это framework для веб разработки, написанный на ЯП Ruby. С появления в 2004 году, Ruby on Rails стремительно набрал популярность и стал одним из мощнейших инструментов для построения динамических веб-приложений. Ruby on Rails используется такими знаменитыми компаниями как: Airbnb, Basecamp, Disney, GitHub, Hulu, Kickstarter, Shopify, Twitter, и The Yellow Pages [4].

Преимущества Ruby on Rails:

1 Это полностью open-source проект, доступный по MIT License, в результате чего является бесплатным для скачивания и использования.

2 Реализация Model-View-Controller (MVC) паттерн для веб-приложений.

3 Обеспечение их интеграции с веб-сервером и сервером баз данных.

4 Использование REST-стиля построения веб-приложений.

5 Применение в разработке приложений следующих принципов: максимальное повторное использование кода (принцип Don’t repeat yourself); использование соглашений по умолчанию по конфигурации (принцип Convention over configuration), при котором явная спецификация конфигураций требуется только в нестандартных случаях [5].

6 Элегантный и компактный дизайн Rails, способствующий большой популярности. Используя податливый нижележащий ЯП Ruby, Rails фактически создает предметно-ориентированный язык (domain-specific language) для написания веб-приложений. В результате много общих задач веб-программирования – таких как генерирование HTML, создание моделей данных и маршрутизация URI – легко решаемы с Rails, а итоговый код программ получается кратким и выразительным.

7 Быстрая адаптация к новым веяниям в веб-технологиях. Например, в Rails одним из первых был полностью реализовал архитектурный стиль REST для веб-приложений. Создатель Rails, David Heinemeier Hansson и рабочая группа Rails используют эти новые идеи также при создании другими фреймворками новых техник. Наиболее ярким примером является слияние Rails и Merb (конкурирующая веб-платформа), так что Rails теперь получает преимущества от модульной конструкции Merb, стабильного API, а также повышенной производительности.

8 Увлечённое и разнообразное сообщество пользователей Rails, сотни open-source разработчиков, многолюдных конференций, форумов и каналов IRC (Internet Relay Chat), огромное количество гемов, богатый набор информативных блогов. Большое количество активных программистов Rails также облегчает обработку неизбежных ошибок приложений: алгоритм – “Ищи в Google сообщение об ошибке” – почти всегда добывает соответствующее сообщение в блоге или ветке форума [5].

MVC состоит из объектов трех видов:

- модель - объект приложения;

- вид - экранное представление;
- контроллер - описывает, как интерфейс реагирует на управляющие действия пользователя.

1.2.3 MVC в Ruby on Rails

До появления схемы MVC эти объекты в пользовательских интерфейсах смешивались. MVC отделяет их друг от друга. Вследствие этого повышается гибкость и улучшаются возможности повторного использования. MVC отделяет вид от модели, устанавливая между ними протокол взаимодействия «подписка/оповещение». Вид гарантирует, что внешнее представление отражает состояние модели. При каждом изменении внутренних данных модель оповещает все зависящие от нее виды. В результате этого вид обновляет себя. Такой подход позволяет присоединить к одной модели несколько видов, обеспечив тем самым различные представления. Можно создать новый вид, не переписывая модель. MVC позволяет также изменять реакцию вида на действия пользователя. При этом визуальное представление остается прежним. Например, можно изменить реакцию на нажатие клавиши или использовать всплывающие меню вместо командных клавиш. MVC инкапсулирует механизм определения реакции в объекте Controller. Отношение вид-контроллер - это пример паттерна проектирования стратегия. Стратегия - это объект для представления алгоритма. Он используется с целью статической или динамической подмены одного алгоритма другим, если существует много вариантов одного алгоритма или, когда с алгоритмом связаны сложные структуры данных, которые хотелось бы инкапсулировать. В MVC используются и другие паттерны проектирования, например, фабричный метод, который позволяет задать для вида класс контроллера по умолчанию, и декоратор для добавления к виду возможности прокрутки. Но основные отношения в схеме MVC описываются паттернами наблюдатель, компоновщик и стратегия [6].

Rails накладывает значительные ограничения на структурирование веб-приложений, которые заметно упрощают создание приложений. Rails навязывает структуру для приложения — модели, представления и контроллеры разрабатываются как отдельные функциональные блоки, а Rails при выполнении заданной программы связывает их вместе. Отличительной особенностью Rails является то, что процесс увязки базируется на использовании разумных умолчаний, которые, как правило, избавляют от написания каких-либо внешних конфигурационных метаданных, обеспечивающих взаимную работу. Приоритет соглашения над конфигурацией является примером концепции Rails [7].

Модель в Ruby on Rails предоставляет остальным компонентам приложения объектно-ориентированное отображение данных. Объекты модели могут осуществлять загрузку и сохранение данных в реляционной базе данных и реализуют бизнес-логику.

Для хранения объектов модели в реляционной СУБД по умолчанию в Rails используется библиотека ActiveRecord.

Представление создаёт пользовательский интерфейс с использованием полученных от контроллера данных. Представление также передает запросы пользователя на манипуляцию данными в контроллер.

Контроллер в Rails — это набор логики, который запускается после получения HTTP-запроса сервером. Контроллер отвечает за вызов методов модели и запускает формирование представления.

В Rails-приложении входящий запрос сначала посылается маршрутизатору, который решает, в какое место приложения должен быть отправлен запрос и как должен быть произведен синтаксический разбор этого запроса. В результате на данном этапе где-то в коде контроллера идентифицируется конкретный метод (называемый в Rails действием). Действие может искать запрошенные данные, может взаимодействовать с моделью и может вызвать другое действие. В результате выполнения действие подготавливает информацию для представления, которое создает изображение для пользователя [7].

Схема MVC в Rails на рисунке:

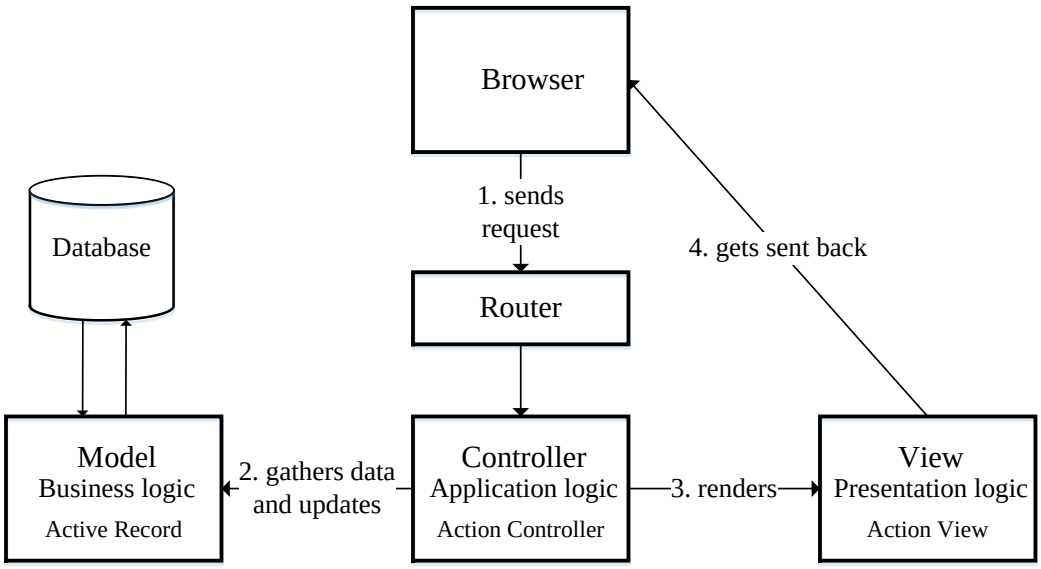

Рисунок 1.5 – Паттерн MVC

Исходя из архитектуры, построенной на MVC, RoR использует три компонента:

- Active Record;
- Action View;
- Action Controller.

Сочетание последних двух известно, как Action Pack. Рассмотрим эти компоненты.

Active Record – это Модель в RoR. Модель хранит данные и предоставляет базу для работы с данными. Кроме этого Active Record также

является ORM фрэймворком. ORM значит Object-relational mapping (Объектно-реляционная проекция). Собственно, Active Record делает следующие вещи:

1 Проекция таблицы на класс. Каждая таблица проецируется на один или несколько классов по принципу convention over configuration (соглашение выше конфигурации). Одно из таких соглашений – имя таблицы должно быть во множественном числе, а название класса – в единственном. Атрибуты таблицы налету проецируются в атрибуты экземпляра Руби. После того, как все проекции сделаны, каждый объект ORM класса представляет определенную строку таблицы, с которой класс был спроецирован.

2 Соединение с БД. Вы можете подключиться к базе данных, используя API, предоставляемый Active Record, который создает необходимый вам запрос непосредственно в движок БД при помощи адаптеров. У Active Record есть адаптеры для MySQL, Postgres, MS SQLServer, DB2, и SQLite. Необходимо лишь записать параметры доступа к БД в файле database.yml.

3 Операции CRUD. Это операции create (создание), retrieve (получение), update (обновление) и delete (удаление) над таблицей. Так как Active Record – это ORM фрэймворк, вы всегда работаете с объектами. Чтобы создать новую строку таблицы, вы создаете новый объект класса и заполняете его переменные экземпляра значениями. Стоит заметить, что все это Active Record делает за вас.

4 Проверка данных. Проверка данных перед помещением их в таблицу – это первый шаг в безопасности вашего проекта. Active Record предоставляет проверку Модели. Данные могут быть проверены автоматически с помощью множества готовых методов, которые, в случае необходимости, можно переписать под собственные нужды.

ActionView – это вид. Он включает в себя логику, необходимую для вывода данных Модели. Представление в Rails отвечает за создание полного или частичного ответа, отображаемого в браузере, обработанного приложением или посланного в виде электронной почты. В простейшем виде представление является фрагментом HTML-кода, отображающего какой-нибудь неизменный текст. Но чаще всего вам потребуется включить динамическое содержимое, созданное методом действия в контроллере [7]. Наиболее часто используемые функции Action View:

5 Шаблоны (Templates). Шаблоны – это файлы, содержащие заполнители (placeholders), которые буду заменены на контент. Шаблоны могут содержать HTML-код и код Ruby, встраиваемый в HTML с использованием синтакса встроенного (embedded) Ruby (ERb).

6 Помощники (helper, далее хелпер) форм и форматирования. Хелперы форм позволяют создавать такие элементы страниц, как чекбоксы, списки, используя готовые методы. В свою очередь хелперы форматирования позволяют форматировать данные необходимым нам способом, методы существуют для дат, валют и строк.

7 Макет. Макеты (layouts) определяют, как контент будет расположен на странице. Динамически создаваемая страница может содержать вложение из нескольких страниц, даже без использования таблиц и фрэймов, используя API Макета.

Action Controller. В веб-приложении Контроллер регулирует поток логики приложения. Он находится на границе программы, перехватывая все запросы, на основе которых он изменяет какой-то объект Модели и вызывает Вид, чтобы отобразить обновленные данные. В RoR Action Controller является Контроллером, вот его основные функции:

8 Поддержка сессий. Сессия – это период времени, проведенный пользователем на сайте. Его можно отследить с помощью cookie или объекта сессии. Cookie – небольшой файл, он не может содержать объекты, в отличие от объекта сессии.

9 Фильтрация. Бывают ситуации, когда необходимо вызвать определенный код, перед тем как исполнять логику Контроллера или после него, например, аутентификация пользователей, логирование событий, предоставление персонального ответа. Помогают в таких случаях фильтры, предоставляемые Action Controller. Существуют три основных фильтра: before, after и around. О них – позже.

10 Кэширование. Кэширование – это процесс, при котором наиболее запрашиваемый контент сохраняется в кэше, чтобы не было необходимости запрашивать его вновь и вновь.

Среды. RoR поощряет использование отдельных сред для каждого из этапов цикла жизни приложения: разработка (development), тестирование (testing) и эксплуатация (production), для каждого из которых создается отдельная БД. Рассмотрим каждую среду.

11 development. В среде разработки ставка делается на немедленное отображение нового варианта при изменении кода – достаточно обновить страницу в браузере. Скорость в этой среде не важна. Когда случается ошибка, она выводится на экран.

12 test. При тестировании мы обычно каждый раз наполняем БД каким-нибудь глупым текстом, чтобы убедиться, что нормальное поведение не зависит от содержания БД. Процедуры юнит-тестинга и теста функциональности в RoR автоматизированы и производятся через консоль. Тестовая среда предоставляет отдельное пространство, в которых оперируют эти процедуры.

13 production. В конце концов ваше приложение выходит к финальной черте, пройдя тесты и избавившись от багов. Теперь обновления кода будут происходить редко и можно сконцентрироваться на производительности, включить кэширование. Нет необходимости писать огромные логи ошибок и пугать пользователей сообщениями об этих ошибках в браузере. Для вас – среда production.

Вокруг Rails сложилась большая экосистема плагинов, которые также называются «джемы» (gem с англ. — «самоцвет»). Для управлений плагинами

существует специальная система RubyGems. Некоторые из них со временем были включены в базовую поставку Rails, например, Sass и CoffeeScript; другие же, хотя и не были включены в базовую поставку, являются стандартом де-факто для большинства разработчиков, например, средство модульного тестирования RSpec [8, 9].

1.2.4 Стек технологий

Существует много веб-приложений (особенно те, что написаны на Ruby on Rails) построенных при помощи слоёной архитектуре, которая часто называется *стек*, потому, что диаграммы обычно отображают слои как сложенные блоки (см. рисунок 1.2).

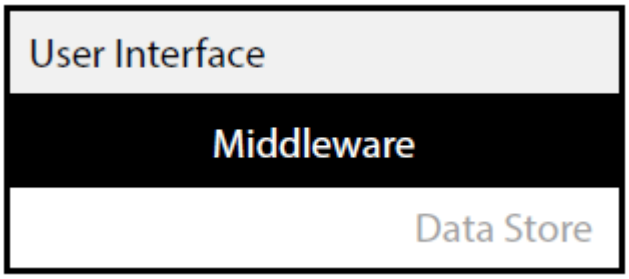

Рисунок 1.6 – Обобщённый стек технологий

Rails представляет середину стека и является middleware. Rails – это место, где находится основная логика приложения. Дно стека – хранилище данных – место, где сохраняется значимая информация приложения. Это обычно реляционная система управления Relational Database Management System (RDBMS). Вершиной стека является пользовательский интерфейс. В веб-приложении он реализуется HTML, CSS и JavaScript выполняемый в браузере [8, 9].

Стек, построенный на выбранных технологиях выглядит так (см. рисунок 1.3):

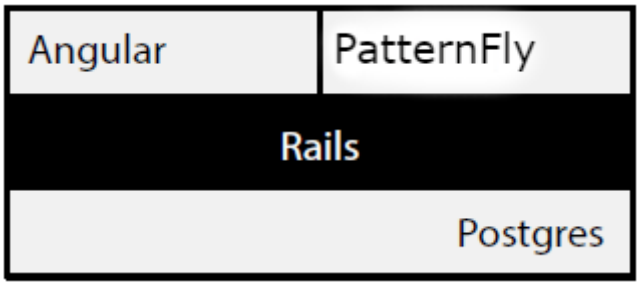

Рисунок 1.7 – Стек технологий

PostgreSQL – это open-source SQL база данных, выпущенная в 1997 году. Она поддерживает множество продвинутых опций, которых нет в других

популярных open-source базах данных таких как MySQL или коммерчески базах, таких как Microsoft SQL Server.

PostgreSQL позволяет создавать очень сложные ограничения. Например, можно потребовать, чтобы email пользователя был из определенного домена, чтобы штат в U.S. адресе был написан точно, как две буквы в верхнем регистре, или даже чтобы штат в адресе уже был в списке разрешенных государственных кодов. Это же можно сделать и при помощи Rails, но выполнение на уровне базы данных означает, что ни баг в коде, ни существующий скрипт, ни разработчик в консоли, ни программа не саможет поместить невалидную информацию в базу данных.

PostgreSQL поддерживает перечисляемые типы, массивы и словари (называются HSTOREs). Во многих базах данных необходимо иметь раздельные таблицы для таких структур данных.

Postgres поддерживает JSON тип данных, позволяя сохранять произвольную информацию в столбце. Это означает, что можно использовать Postgres в качестве хранилища для документов или сохранять данные, которые не соответствуют схеме. Используя JSONB тип данных, вы убеждаетесь, что JSON поля могут быть индексируемы также, как и структурированные поля таблицы.

1.2.5 JavaScript

JavaScript – это интерпретируемый ЯП с объектно-ориентированными возможностями. Ядро языка JavaScript поддерживает работу с такими простыми типами данных, как числа, строки и булевы значения. Помимо этого, он обладает встроенной поддержкой массивов, дат и объектов регулярных выражений. Обычно JavaScript применяется в веб-браузерах, а расширение его возможностей за счет введения объектов позволяет организовать взаимодействие с пользователем, управлять веб-браузером и изменять содержимое документа, отображаемое в пределах окна веб-браузера. Эта встроенная версия JavaScript запускает сценарии, внедренные в HTML-код веб-страниц. Как правило, эта версия называется клиентским языком JavaScript, чтобы подчеркнуть, что сценарий исполняется на клиентском компьютере, а не на веб-сервере.

Когда интерпретатор JavaScript встраивается в веб-браузер, результатом является клиентский JavaScript. Клиентский JavaScript включает в себя интерпретатор JavaScript и объектную модель документа (Document Object Model, DOM), определяемую веб-браузером.

Документы могут содержать JavaScript-сценарии, которые в свою очередь могут использовать модель DOM для модификации документа или управления способом его отображения. Другими словами, можно сказать, что клиентский JavaScript позволяет определить поведение статического содержимого веб-страниц. Клиентский JavaScript является основой таких технологий разработки веб-приложений, как DHTML (глава 16), и таких архитектур, как Asynchronous Javascript and XML (Ajax) [10].

1.2.6 AngularJS

AngularJS – это JavaScript MVC framework, созданный и поддерживаемый Google. Angular позиционирует себя как Model-View-Whatever framework, в нашем случае Whatever - это контроллер (см. рисунок 1.4). Angular воспринимает view не как статический кусок HTML, а как полномасштабное приложение. Angular предоставляет мощные средства по организации кода и позволяет структурировать разметку для создания выразительного, тестируемого, управляемого frontend кода [11].

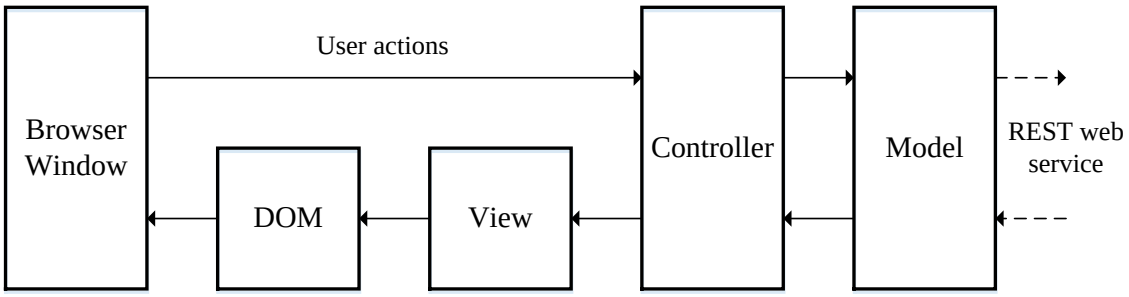

Рисунок 1.8 – MVC в AngularJS

Angular помогает чисто разделить код и представление. Angular организует frontend как приложение со своими собственными путями, контроллерами и представлениями. Это упрощает frontend и позволяет легко организовать JavaScript код.

Angular с самого начала поддерживал unit-тестирование JavaScript кода.

Чистое, декларативное представление. Angular представление – это просто HTML. Angular добавляет специальные атрибуты, называемые директивами, которые позволяют чисто соединить данные и функции с разметкой. Нет необходимости встраивать код или скрипты, существует чистое разделение между представлением и кодом.

Angular имеет большую экосистему компонентов и модулей благодаря своей популярности. Множество типичных вопросов имеют решение в экосистеме Angular [11, 12].

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Системы домашней автоматизации, интернета вещей очень популярны на данный момент. Данный проект является интерфейсом пользователя к программно-аппаратной платформе удаленного управления устройствами.

Пользователь, заходит на домашнюю страницу, откуда может посмотреть краткую информацию о проекте, имеет возможность перейти на страницу регистрации.

В процессе регистрации помимо почты и пароля пользователь может опционально ввести идентификационный номер домашнего сервера автоматизации. Проводятся валидации полей формы, пользователь оповещается о неправильно заполненных полях и ему предоставляется возможность исправить эти поля и отправить форму заново.

Если пользователь зарегистрирован, то он видит dashboard проекта, на котором находятся карточки со статистикой:

- кол-во подключенных датчиков в системе;
- кол-во логических областей, которым принадлежат датчики;
- кол-во пользовательских скриптов;
- кол-во скриптов находящихся на выполнении;
- кол-во новоподключённых устройств.

Постоянными элементами интерфейса являются Navbar и Vertical Navigation.

Navbar содержит следующие элементы слева направо:

1 Hamburger Menu. Иконка меню служит для открытия и скрытия вертикальной навигационной панели.

2 Logo. SVG изображение.

3 Application Title. Содержит имя продукта

4 Notification Icon. Через нее доступна Notification Drawer панель критических событий сервера. Это самодостаточная система, которая может быть просмотрена без необходимости перехода на другие страницы приложения. На самой иконке находится badge, отображающий кол-во новых уведомлений с домашнего сервера автоматизации.

5 Help Icon. При его нажатии появляется выпадающее меню с обязательной опцией «About», которая запускает модальное окно с информацией о продукте.

6 User Icon. Показывает имя зарегистрированного пользователя. По нажатии на нее появляется выпадающее меню с обязательной опцией «Logout».

Vertical Navigation – это глобальная навигационная панель, отображаемая по левой стороне страницы. Вертикальная навигационная панель имеет до трех уровней вложенности. На ней расположены ссылки на страницу скриптов, страницу логических областей, или выбрать страницу конкретной логической области, содержащей датчики, также присутствует ссылка на dashboard.

Страница логических областей (areas). Датчики, подсоединённые к системе, делятся на логические области, т.е. они принадлежат конкретной области. Логические области служат только лишь для удобной группировки устройств и не отражают физического строения распределённой системы устройств. Логическая область – это удобная абстракция, позволяющая абстрагироваться от деления устройств, например, по местонахождению (комната, дом, и т.д.). Вместо этого название и, возможно, описание area позволяет применять систему в более общих ситуациях автоматизации, не привязываясь к определённым понятиям. Таким образом, area имеет уникальное имя, задаваемое пользователем при создании. Можно редактировать имя и описание логической области. Также доступны все CRUD операции над ней. Датчик, подсоединенный к системе принадлежит к area по умолчанию. Пользователь может менять принадлежность датчика к area в любое время. При удалении area, датчики, принадлежащие к нему, переносятся в area по умолчанию.

Страница логической области (area). Эта страница содержит список датчиков, принадлежащих конкретной логической области. Над датчиками можно производить действие по перенесению их в другие логические области. Для каждого датчика помимо имени и типа отображается некоторая уточняющая информация. Для каждого датчика в списке есть ссылка на индивидуальную страницу датчика.

Страница датчика. Шаблоны страниц датчика зависят от типа датчика. Со страницы датчика производится управление им. Имеется возможность просмотреть полную информацию о нем. Посмотреть, в каких скриптах он задействован. Добавление новых устройств происходит при подсоединении их к системе, вся информация о них передается через домашний сервер автоматизации. Новое устройство помещается изначально в логическую область по умолчанию, откуда может быть перенесено в любую логическую область. Устройство удаляется при отсоединения его от системы.

Notification Drawer для критических событий домашнего сервера. При первом запросе к домашнему серверу автоматизации клиенту передается список критических событий, таких как появление нового устройства в системе, ошибка в устройстве, ошибка выполнения скрипта. Вместе с информацией в критическом сообщении показывается метка времени – когда это событие произошло.

Страница скриптов. Показывает список всех скриптов. Можно создать скрипт. Создание скрипта основано на wizard-е. Каждый шаг в процессе создания скрипта влияет на действия, возможные в дальнейшем. После прохода всех шагов, wizard-а генерируется скрипт, который сохраняется в базе данных и передается на домашний сервер. Имеется возможность запустить отдельный скрипт. При запуске скрипта выполняется попытка его выполнения на домашнем сервере, а состояние выполнения доступно для отслеживания пользователю.

Страница Действий (actions). Содержит список действий, инициализированных пользователем, таких как управление отдельным датчиком или запуск скрипта и показывает результат или состояние действий. У действия имеется метка времени его запуска и окончания.

Страница аккаунта. Содержит данные пользователя, а также идентификатор домашнего сервера. Имеется возможность редактировать эти данные. Основная информация может быть заполнена через форму регистрации.

Toast Notifications. Тост-уведомления показываются в верхнем правом углу приложения. Они служат для показа происходящих событий в реальном времени. Эти уведомления пропадают с течением времени. Они не блокируют информацию, находящуюся за ними и отображаются достаточное время, чтобы пользователь успел прочитать сообщение. Это уведомление не пропадает, если пользователь «завис» над ним.

Блок пользовательского интерфейса. Пользовательский интерфейс организован как Single page application (SPA) – единственный HTML-документ используется в качестве оболочки всех веб-страниц и организует взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript посредством AJAX и WebSocket. SPA приложение передаёт весь необходимый код JavaScript (модули, виджеты, контроллеры) вместе с загрузкой самой страницы. SPA-приложение типичный представитель HTML5. SPA-приложения работают на большом количестве устройств (компьютеры, планшеты, смартфоны). SPA-приложения имеют богатый и насыщенный пользовательский интерфейс, так как веб-страница одна. Намного проще хранить информацию о сеансе, управлять состояниями представлений и управлять анимацией. Некоторые функции, такие как routing переносятся со стороны backend-а на клиентскую, что позволяет не обращаться с запросом по каждой странице, а запрашивать только необходимые данные. Также при заполнении форм, валидации выполняются как на backend-е, так и на frontend-е. AngularJS адоптирован для поддержки SPA принципов. AngularJS фреймворк для клиентской стороны. Angular использует двустороннее связывание данных в пользовательском интерфейсе (UI), связывая UI-элементы м моделью. Для двустороннего связывания Angular применяет паттерн Наблюдатель. Двустороннее связывание позволяет автоматически обновлять представления, как только изменяется модель и наоборот. В традиционном подходе - генерировании HTML на стороне сервера контроллер и модель взаимодействуют внутри процесса на сервере для генерации HTML представлений. В приложении, использующем AngularJS контроллер и модель находятся у клиента в браузере, поэтому новые страницы могут быть сгенерированы без какого-либо взаимодействия с сервером. Angular использует технологию AJAX. Преимущественно используется XMLHttpRequest объект в JavaScript, который предоставляет возможность делать HTTP-запросы из JavaScript на сервер без перезагрузки данных. Результатом запросы к серверу является сырые данные в формате JSON или

XML или же новая HTML страница. В случае возвращения HTML как ответа сервером JavaScript на стороне клиента обновляет частичный участок Document Object Model (DOM). В случае прихода сырых данных JavaScript на стороне клиента обычно генерирует из сырых данных HTML, который затем используется для обновления частичного участка DOM.

Блок связи с домашним сервером расположен на клиентской стороне (frontend). Он выполняет связь с домашним сервером автоматизации по полнодуплексному протоколу WebSocket. Для получения url адреса домашнего сервера пользователю необходимо зарегистрировать домашний сервер в приложении. Он способен это сделать непосредственно во время регистрации или позже, зайдя в настройки аккаунта. Именно этот блок инициализирует REST операции по добавлению нового устройства и другие команды пришедшие с домашнего сервера.

Блок обновления состояний устройств. Данный блок реализует поведенческий паттерн «Посредник». Он обеспечивает взаимодействие множества объектов, находящихся на стороне домашнего сервера со множеством соответствующих объектов на стороне данного веб-приложения. При этом получается слабая связанность и устройства избавляются от необходимости явно ссылаться друг на друга. Команда о подключении нового устройства подаётся через блок связи с домашним сервером. Информация о новом устройстве заносится в базу данных. Домашний сервер шлёт информацию о состоянии каждого устройства при подключении клиента к домашнему серверу и при каждом изменении состояния устройства. При управлении устройствами из пользовательского интерфейса в блок обновления состояния устройств приходит реакция с домашнего сервера. Также страница действий содержит текущую информацию о состоянии устройств.

Блок удаленного управления устройствами. Этот блок непосредственно связан с блоком пользовательского интерфейса. Его логика связана с пользовательскими действиями над устройствами. Он формирует новое состояние устройства на основе пользовательских действий в интерфейсе. Существует необходимость двусторонней связи между веб-сервисом и домашним сервером автоматизации, так как пользователю необходимо понять, выполнена ли его команда с удалённым устройством или нет. Для этого, после передачи команды домашнему серверу веб-сервис ждёт ответ с результатом выполнения команды. Ответ принимается асинхронно, чтобы создать иллюзию немедленного выполнения пользовательской команды. В случае неудачи пользователь оповещается о нештатной ситуации. Передаёт новое состояние устройства в блок связи с домашним сервером.

Блок аутентификации пользователя и регистрации домашнего сервера. Так как веб-сервис предоставляет доступ к конфиденциальной информации (показания датчиков и устройств), а также к управлению данными устройствами, то строгая аутентификация является актуальным вопросом. Базовая аутентификация – включение имени пользователя и пароля в состав

HTTP POST запроса не подходит для данного приложения, так как любой, кто перехватит пакет узнает секретную информацию, поэтому использована дайджест-аутентификация. Методика заключается запросе у пользователя пароля, возможно с подтверждением и сохранения зашифрованной версии пароля в базе. Сравнение зашифрованных паролей вместо непосредственного сравнения даёт дополнительное преимущество – есть возможность аутентифицировать пользователей без хранения в базе данных самих паролей, тем самым избегается проблема системы безопасности приложения. Производит действия с формой при регистрации пользователя. Блок выполняет валидации полей формы.

Блок формирования и управления скриптами. Скрипты позволяют пользователю определить последовательность выполняемых действий, которые будут выполнены при запуске скрипта. Таким образом, скрипты – удобный механизм автоматизации управления. Аналоги представляют скрипты в разной форме, например, описание скрипта в виде YAML формате. Этот вариант является довольно сложным и запутанным для использования пользователем. Вариант с написанием пользователем кода на существующих скриптовых языках, таких как javascript, php и т.д. также не является оптимальным, так как требует хотя бы какого-то знакомства с программированием. Поэтому был выбран вариант создания скрипта через выполнение пользователем пошаговых действий в wizard-е. Этот подход достаточно прост для пользователя. Он скрывает внутреннюю реализацию сценария и тем самым сокращает время, необходимое на его создание. Данный подход также достаточно гибок, так как wizard предоставляет возможность разбить определение сложного сценария на более простые компоненты и может предоставлять пользователю разные шаги в зависимости от решений пользователя на предыдущих шагах wizard-а. Сформированный скрипт отправляется в базу данных и передаётся на домашний сервер автоматизации.
Блок обработки критических сообщений с домашнего сервера. Критические сообщения могут влиять на любую часть пользовательского интерфейса. Они помещаются в Notification Drawer. Также с ними связана логика, при таких асинхронных событиях, приходящих с домашнего сервера как удаление устройства из системы, сбои в работе устройств, сбои в работе скриптов и т.д.

Блок обработки данных с сервиса прогноза погоды. Данные с сервиса могут отображаться на widget-е, а также пользователь может просматривать погоду в конкретном городе. Блок использует API сервиса погоды.

База данных. В базу данных сохраняются данные о пользователях, данные о домашних серверах. В БД также хранятся пользовательские скрипты. Данные о логических областях, данные о датчиках.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.250501.003 С1.

ЗАКЛЮЧЕНИЕ

Результатом разработки системы является web-приложение, выполняющее функцию клиента для системы умного дома.

Отличительной особенностью системы от многих аналогов является реализация механизма PnP, что существенно упрощает эксплуатацию системы. Пользователю необходимо только подключить новое устройство к контроллеру и устройство «подхватится» системой и появится в area по умолчанию в интерфейсе. Пользователь сразу может начать работу с этим устройством, так как все его возможности известны и для каждого типа устройства реализована отдельная веб-страница управления им.

Проект рассчитан на применение пользователями, не имеющими навыков программирования. Поэтому был сделан упор на простоту использования. Пользовательские сценарии реализованы в виде пошаговых wizard-ов, это ускоряет процесс освоения системы. Пользователю нет необходимости изучать синтаксис разных конфигурационных файлов, а также список функций, реализованных системой и язык программирования, для вызова этих функций. Сценарии, генерируемые при помощи wizard-ов полностью абстрагируют пользователя от реализации системы.

Так как общение с домашним сервером ведётся по протоколу WebSocket, взаимодействие получается интерактивным, полнодуплексным, UI реагирует на изменения в системе в реальном времени.

В качестве усовершенствования проекта в дальнейшем планируется:

1 Запустить сервер UI в локальной сети, в которой находится домашний сервер автоматизации. Сделать возможным автоматическое переключение с облачного UI сервера на домашний и наоборот в зависимости от таких факторов, находится ли пользователь в локальной сети и есть ли доступ в Internet.

2 Добавить возможность делать backup данных и настроек с домашнего сервера автоматизации на облако.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Флэнаган Д. Язык программирования Ruby. / Д. Флэнаган, Ю. Мацумото. – СПб.: Питер, 2011. – 496с.

[2] Thomas D. The Pragmatic Programmers' Guide / D. Thomas. – Dallas, Texas, 2013.

[3] Официальный веб-сайт Ruby [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.ruby-lang.org/.

[4] Руби С. Гибкая разработка веб-приложений в среде Rails. 4-е изд./ С. Руби, Д. Томас, Д. Хэнссон. – СПб.: Питер,2012. – 464 с.: ил.

[5] The Ruby On Rails Tutorial Learn web development with Rails [Электронный ресурс]. – Электронные данные. – Режим доступа: http://railstutorial.ru/.

[6] Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования. / Э. Гамма, Р. Хелм, Р. Джонсон – СПб: Питер, 2001. –368 с.: ил.

[7] Ruby on Rails Guides [Электронный ресурс]. – Электронные данные. – Режим доступа: http://guides.rubyonrails.org.

[8] Coperland D. Powerful, Effective, and Efficient Full-Stack Web Development / D. Coperland. – Dallas, Texas, 2016.

[9] Coperland D. Powerful, Effective, and Efficient Full-Stack Web Development Second Edition / D. Coperland. – Raleigh, North Carolina, 2016.

[10] Флэнаган Д. JavaScript. Подробное руководство. / Д. Флэнаган. – СПб: Символ-Плюс,2008. –992 с., ил.

[11] AngularJS API Reference [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.angularjs.org/api.

[12] AngularJS with Ruby on Rails [Электронный ресурс]. – Режим доступа: http://angular-rails.com/.