

ВВЕДЕНИЕ

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

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, выключатели 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. В 2014 году Gartner назвал облачные вычисления среди 10 главных стратегических направлений развития технологий.

«Облако» – это место, где удобно хранить данные необходимого объема, быстро и доступно пользоваться сервисами и приложениями. Потребителю достаточно иметь подключенное к Интернету устройство, чтобы воспользоваться программой из «облака» в любое время, в любом месте.

Различают три сервисные модели облачных вычислений:

* Infrastructure-as-a-service (IaaS), инфраструктура как услуга: потребитель использует вычислительные ресурсы поставщика (сервер, сетевую инфраструктуру, хранилище данных);
* Platform-as-a-service (PaaS), платформа как услуга: поставщик предоставляет потребителю доступ к использованию программной платформы;
* Software-as-a-service (SaaS), программное обеспечение как услуга: потребитель пользуется готовым приложением поставщика.

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.

Проект предоставляет возможности по управлению освещением, системой микроклимата, управлению электроприводами, внедрению системы безопасности.

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).

ОБЗОР ЛИТЕРАТУРЫ

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.

Ruby – это динамически типизированный ЯП, характеризующийся сложной, но выразительной грамматикой. Он имеет базовую библиотеку классов с мощным и разнообразным API. Ruby включает в себя отличительные особенности языков Lisp, Smalltalk и Perl. Ruby является полностью объектно-ориентированным ЯП, однако в нем также используются процедурный и функциональный стили программирования. Ruby имеет большие потенциальные возможности по поддержке метапрограммирования. Это позволяет использовать Ruby в создании языков, предназначенных для конкретных предметных областей (domain-specific languages - DSL) [1]. Ruby имеет независимую от операционной системы реализацию многопоточности, полностью свободную кроссплатформенную реализацию интерпретатора, отчетливую динамическую типизацию, сборщик мусора и многие другие функции.

Каждое значение в Ruby – это объект, в том числе простые числовые литералы и значения true, false и nil. 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].

Возможности Ruby [3]:

* Позволяет обрабатывать исключения в стиле Java и Python.
* Реализует выражения вместо операторов. Синтаксис Ruby направлен на использование выражений. Управляющие конструкции, такие как if, являющиеся в других ЯП операторами, в Ruby представляют собой выражения. Хотя все «операторы» в Ruby являются выражениями, однако не все возвращают содержательные значения. Например, циклы while и определения методов тоже выражения, которые, как правило, возвращают значение nil. Как и во многих других ЯП, выражения в Ruby обычно выстраиваются из значений и операторов.
* Позволяет переопределять операторы, которые на самом деле являются методами.
* Полностью объектно-ориентированный язык программирования. Все данные в Ruby являются объектами в понимании Smalltalk. Единственное исключение — управляющие конструкции, которые в Ruby, в отличие от Smalltalk, не являются объектами. Например, число «1» — это экземпляр класса Fixnum. Также поддерживается добавление методов в класс и даже в конкретный экземпляр во время выполнения программы.
* Не поддерживает множественное наследование, однако вместо него может использоваться концепция «примесей», основанная в данном языке на механизме модулей.
* Содержит автоматический сборщик мусора. Он работает для всех объектов Ruby, в том числе для внешних библиотек.
* Позволяет легко создавать расширения для Ruby на Си из-за имеющегося сборщика мусора и из-за простого и удобного API.
* Поддерживает замыкания с полной привязкой к переменным.
* Поддерживает блоки кода (код заключается в { … } или do … end) и итераторы. Это дает возможность вызывать методы в отношении целых чисел, что очень часто используется. Например, методы times и upto – это методы целочисленных объектов. Это особая разновидность методов – итераторы и ведут себя как циклы. Код помещается в фигурные скобки (блок) и служит как тело цикла. Несмотря на то, что язык поддерживает обычные циклы for и while, очень распространена реализация циклов с использованием структурных компонентов, которые являются по сути вызовами методов. Блоки могут использоваться в методах или преобразовываться в замыкания.
* Целые переменные в Ruby автоматически конвертируются между типами Fixnum (32-разрядные) и Bignum (больше 32 разрядов) в зависимости от их значения. Это позволяет производить целочисленные математические расчёты с неограниченно большой точностью.
* Не требует предварительного объявления переменных, однако для интерпретатора предпочтительно, чтобы переменным присваивалось пустое значение nil (тогда интерпретатор знает, что идентификатор обозначает переменную, а не имя метода). Язык использует простые соглашения для обозначения области видимости. Пример: просто var — локальная переменная, @var — переменная экземпляра (член или поле объекта класса), @@var — переменная класса, $var — глобальная переменная.
* В Ruby непосредственно в языке реализованы многие шаблоны проектирования, так, например, «одиночка» (singleton) может быть (хотя и не обязательно) реализован добавлением необходимых методов к одному конкретному объекту (см. ниже).
* Может динамически загружать расширения, если это позволяет операционная система.
* Имеет независимую от ОС поддержку невытесняющей многопоточности.
* Перенесён на множество платформ. Он разрабатывался на Linux, но работает на многих версиях Unix, DOS, Microsoft Windows (в частности, Win32), Mac OS, BeOS, OS/2 и т. д.

Формальная спецификация ЯП Ruby отсутствует, поэтому интерпретатор Ruby с веб-сайта http://www.ruby-lang.org считается эталонной реализацией, которая определяет сам язык. Эта реализация создана на Си и имеет название MRI (Matz Ruby Implementation). В Ruby 1.9 исходный MRI-интерпретатор объединили с YARV («Yet Another Ruby Virtual machine») - новой виртуальной машиной Ruby), чтобы создать новую эталонную реализацию, выполняющую внутреннюю компиляцию в байт-код, а затем выполняющую этот байт-код на виртуальной машине.

* JRuby – реализация Ruby, созданная на Java, доступна на веб-сайте http://hruby.org. JRuby – это программное обеспечение с открытым кодом разработанное в Sun Microsystems.
* IronRuby – это реализация Ruby? Созданная компанией Microsoft для .NET framework и DLR (Dynamic Language Runtime). Домашняя веб-страница проекта – http://www.ironruby.net.
* Rubinius – проект с открытым кодом, является альтернативной реализацией Ruby, написанной преимущественно на самом Ruby. Виртуальная машина Rubinius, называется shotgun, в общих чертах основана на архитектуре Smalltalk-80 VM.
* Cardinal – это реализация Ruby, предназначенная для запуска на виртуальной машине Parrot VM [3].

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].

* Ruby on Rails является полностью open-source проектом, доступным по MIT License, что в результате является бесплатным для скачивания и использования.
* Реализует Model-View-Controller (MVC) паттерн для веб-приложений.
* Обеспечивает их интеграцию с веб-сервером и сервером баз данных.
* Ruby on Rails использует REST-стиль построения веб-приложений.
* Применяет следующие принципы в разработке приложений: 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].
* Rails также получил большую популярность благодаря своему элегантному и компактному дизайну. 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 и быстро решаемы с Rails, а результирующий код программ получается кратким и читаемым.
* Rails также быстро адаптируется к новым веяниям в веб-технологиях. Например, в Rails одним из первых был полностью реализовал архитектурный стиль REST для веб-приложений. Также при создании другими фреймворками новых техник, создатель Rails, David Heinemeier Hansson и рабочая группа Rails не стесняются использовать эти новые идеи. Пожалуй, 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.
* Наконец, Rails выигрывает от необычайно увлечённого и разнообразного сообщества. Результаты: сотни open-source разработчиков, многолюдных конференции, огромное количество гемов, богатый набор информативных блогов, и рог изобилия форумов и каналов IRC (Internet Relay Chat). 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 - это объект приложения, а вид - экранное представление. Контроллер описывает, как интерфейс реагирует на управляющие воздействия пользователя. 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. М VC отделяет вид от модели, устанавливая между ними протокол взаимодействия «подписка/оповещение». Вид должен гарантировать, 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 МVС 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 МVС описываются паттернами наблюдатель, компоновщик и стратегия [6].

Существенной особенностью Rails является то, что она накладывает значительные ограничения на структурирование веб-приложений, которые, надо отметить, заметно упрощают создание приложений. Rails навязывает структуру для приложения — вы разрабатываете модели, представления и контроллеры как отдельные функциональные блоки, a Rails при выполнении вашей программы связывает их вместе. Отличительной особенностью Rails является то, что процесс увязки базируется на использовании разумных умолчаний, которые, как правило, избавляют вас 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 1.1 – 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).

Active Record выполняет следующее:

* 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.
* Соединение с БД. Можно подключиться к базе данных, используя API, предоставляемый Active Record, который создает необходимый 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.
* 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. Для создания новой строки таблицы создается новый объект класса, а его переменные экземпляра заполняются значениями. Стоит заметить, что все это Active Record делает за вас.
* 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:

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

* 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.
* Фильтрация. Есть ситуации, когда необходимо вызвать определенный код, перед тем как исполнять логику Контроллера или после него, например, аутентификация пользователей, логирование событий, предоставление персонального ответа. В таких случаях применяются фильтры, предоставляемые Action Controller. (Существуют три основных фильтра: before, after и around).
* 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.

* 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.
* test. При тестировании мы обычно каждый раз наполняем БД любым 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.
* production. В итоге 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].

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 1.2 – Обобщённый cтек 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 1.3 – 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.

JavaScript – это интерпретируемый ЯП с объектно-ориентированными возможностями. Ядро языка JavaScript поддерживает работу с такими простыми типами данных, как числа, строки и булевы значения. Кроме 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, появляющееся в пределах окна веб-браузера. Эта встроенная версия JavaScript запускает сценарии, внедренные в HTML-код веб-страниц. Чтобы подчеркнуть, что сценарий исполняется на клиентском компьютере, а не на веб-сервере, эту версию называют клиентским языком JavaScript.

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, а те, в свою очередь, могут использовать модель DOM для модификации документа или управления способом его отображения, т. е. 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].

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 1.3 – MVC в AngularJS

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

Angular с самого начала поддерживал unit-тестирование JavaScript кода.

Чистые, декларативные вьюхи. Angular вьюхи – 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 вьюхами и кодом.

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

PatternFly – это bootstrap-based CSS framework, созданный компанией RedHat. Bootstrap содержит компоненты (CSS классы), используемые в HTML для создания форм, панелей, предупреждающих сообщений и т.д. Преимущество Bootstrap-подобных фреймворков в том, что можно создать полнофункциональный пользовательский интерфейс без написания какого-либо CSS. Bootstrap содержит множество классов CSS для разных ситуаций.

Bootstrap состоит из следующих частей:

* типография;
* сеточная система, упрощает размещение мульти колоночных компонентов;
* стили форм;
* компоненты, используются для решения простых дизайнерских вопросов;

СПИСОК ЛИТЕРАТУРЫ

[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/.