ВВЕДЕНИЕ

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

Внедрение практических решений для концепции «интернета вещей» начиная с 2010-х годов считается одной из устойчивых тенденций в сфере ИТ. Это напрямую связанно с появлением облачных вычислений, повсеместным распространением беспроводных сетей, освоением программно-конфигурируемых сетей и развитием технологий межмашинного взаимодействия (Machine-to-Machine).

IoT (Internet of Things) - это новая парадигма Internet. Под термином "вещь" в IoT подразумеваются интеллектуальные, т.е. "умные" предметы или объекты. IoT - это традиционная или существующая сеть Интернет, расширенная подключенными к ней вычислительными сетями физических предметов или вещей, которые могут самостоятельно организовывать различные модели подключения или общения ("Thing - Thing", "Thing - User" и "Thing - Web Object").

Следует отметить, что «умные» вещи– это датчики или приводы, снабженные микроконтроллером с ОС реального времени со стеком протоколов, памятью и устройством связи, встроенные в различные объекты, например, в электросчетчики или газовые счетчики, датчики давления, вибрации или температуры, выключатели и т.д. "Умные" объекты могут быть организованны в вычислительную сеть физических объектов, которые могут быть подключены через шлюзы (хабы или специализированные IoT платформы) к традиционной сети Интернет.

На технологическом уровне IoT – это концепция развития инфраструктуры сети (физической основы) Интернет, в которой "умные" вещи без участия человека способны подключиться к сети для удаленного взаимодействия с другими устройствами (Thing - Thing) или взаимодействия с автономными или облачными ЦОДами или DATA-центрами (Thing - Web Objects) для передачи данных на хранение, их обработку, аналитику и принятия управленческих решений, направленных на изменение окружающей среды, или для взаимодействия с пользовательскими терминалами (Thing - User) для контроля и управления этими устройствами.

Концепция облачных вычислений, появилась в 2006 году, когда Amazon.com, в то время книжный интернет-магазин, представил Amazon Web Services (AWS), положив начало движению облачных вычислений. AWS предоставляет широкий набор сервисов, таких как вычислительные мощности и хранилища данных. Скоро к Amazon.com присоединились Netflix, Microsoft, Google, Apple и IBM, и рынок облачных вычислений разросся. В 2014 году Gartner назвал облачные вычисления среди 10 главных стратегических технологических трендов. «Облако» – это место, где можно хранить данные нужного вам объема, быстро и легко пользоваться сервисами и приложениями. Подключенное к Интернету устройство – единственное, что вам нужно, чтобы воспользоваться программой из «облака» в любое время, в любом месте.

Выделяют три сервисные модели облачных вычислений:

* Infrastructure-as-a-service (IaaS), инфраструктура как услуга, когда потребитель использует вычислительные ресурсы поставщика (сервер, сетевую инфраструктуру, хранилище данных);
* Platform-as-a-service (PaaS), платформа как услуга, когда поставщик предоставляет потребителю доступ к использованию программной платформы;
* Software-as-a-service (SaaS), программное обеспечение как услуга, когда потребитель может пользоваться 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 вид управляемых устройств среди поддерживаемых системой в соответствии с задачами, исполняемые ими.

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, имеющий сложную, но выразительную грамматику. Он имеет базовую библиотеку классов с мощным и богатым API. Ruby получил черты таких языков, как Lisp, Smalltalk и Perl. Ruby полностью объектно-ориентированный язык, хотя в нем используются также и процедурный и функциональный стили программирования. Ruby имеет большие потенциальные возможности по поддержке метапрограммирования, что позволяет использовать Ruby в создании языков, предназначенных для конкретных предметных областей (domain-specific languages - DSL) [1]. Ruby имеет независимую от операционной системы реализацию многопоточности, строгую динамическую типизацию, сборщик мусора и многие другие возможности. Ruby имеет полностью свободную кроссплатформенную реализацию интерпретатора.

Каждое значение в Ruby – это объект, даже простые числовые литералы и значения true, false и nil. Любая функция является методом. В большинстве ЯП в вызовах функций и методов требуются круглые скобки, но не в Ruby. Обычно для Ruby круглые скобки необязательны и часто опущены, особенно при вызовах методов, не требующих аргументов. Отсутствие круглых скобок в вызовах методов делает эти вызовы похожими на ссылки на поименованные поля или поименованные переменные объекта. Это сделано специально, так как Ruby острого следит за инкапсуляцией своих объектов – нету доступа к внутреннему состоянию объекта за его пределами. Любой доступ осуществляется через посредника в виде метода доступа [1].

Ruby не поддерживает множественное наследование, он использует мощный механизм примесей вместо него. Все классы (напрямую или через другие классы) выведены из класса Object, поэтому, любой объект может использовать определённые в нём методы (например, class, to\_s, nil?). Процедурный стиль также поддерживается, но все глобальные процедуры неявно являются закрытыми методами класса Object [11].

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 [11].

Возможности Ruby [1,11]:

* Позволяет обрабатывать исключения в стиле Java и Python.
* Реализует выражения вместо операторов. Синтаксис Ruby направлен на использование выражений. Управляющие конструкции, такие как if, являются в других ЯП операторами, в Ruby представляют собой выражения. Хотя все «операторы» в Ruby являются выражениями, но не все возвращают содержательные значения. К примеру, циклы while и определения методов тоже выражения, которые обычно 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, но 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.
* Создавать расширения для Ruby на Си очень просто частично из-за сборщика мусора, частично из-за несложного 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, что позволяет производить целочисленные математические расчёты со сколь угодно большой точностью.
* Не требует предварительного объявления переменных, но для интерпретатора желательно, 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 обязан) 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.

СПИСОК ЛИТЕРАТУРЫ

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

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

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

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

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

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

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

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

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

[10] Thomas D. The Pragmatic Programmers’ Guide – Dallas, Texas, 2013.

[11] Википедия [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Ruby.

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