Урок 1

Введение

MVC. Стек технологий. Настройка рабочего окружения. Первый проект.

Язык Ruby и фреймворк Ruby on Rails

Язык Ruby был создан японским программистом Юкихиро Мацумото в 1995 году. Одной из его целей была разработка языка, программирование на котором доставляло бы удовольствие. На сегодня это один из самых удобных и лаконичных языков программирования в мире.

Язык довольно долго «варился» в замкнутой японской среде и стал доступен мировой общественности лишь ближе к 2000-м годам, когда его документации была перевода на английский язык. Поэтому как язык программирования для создания сайтов Ruby стал использоваться относительно поздно − к 2005 году. В это время для создания сайтов уже применялись следующие языки программирования:

- Java.
- Perl.
- PHP.
- Python.
- C#.

В Интернете постоянно происходят изменения, и время жизни проектов не так велико. Постоянно требуется разработка новых и модификация существующих сайтов. Для того, чтобы снизить время и стоимость разработки, создаются фреймворки, которые ускоряют создание новых проектов.

Фреймворк − это заготовка приложения плюс набор дополнительных библиотек и утилит для его развития, тестирования и сопровождения.

В качестве языка программирования для Web-разработки Ruby приобрел популярность в 2005 году. Именно в это время был создан Web-фреймворк Ruby on Rails (RoR).

Одно из преимуществ Ruby on Rails − скорость разработки, что очень востребовано в стартапах. Ведь им важно выпустить свою версию на рынок как можно быстрее. Первая версия Twitter была разработана как раз на Ruby on Rails. С использованием Ruby on Rails написаны многие известные сайты: GitHub.com, Lenta.ru, GeekBrains.ru.

Высокая скорость разработки достигается благодаря тому, что Ruby on Rails, вслед за Ruby, предполагает большое количество соглашений. Как результат, требуется меньше времени и кода. Это снижает количество ошибок, а сам проект может разрабатывать небольшая команда. Заработная плата разработчиков − одна из основных статей расходов, поэтому сильная RoR-команда позволяет быстро и дешево разрабатывать проекты.

Еще одним преимуществом Ruby on Rails является его целостность и сосредоточенность сообщества разработчиков на одном фреймворке. Более того, Ruby on Rails — это не один фреймворк, а целая их группа. Так как Ruby on Rails относительно поздно дебютировал в Web, ему удалось избежать архитектурных и организационных ошибок, которые были допущены в других экосистемах, например, JavaScript или PHP.

В последних экосистемах силы разработчиков были распылены по нескольким альтернативным проектам. Существует несколько фреймворков, компоненты которых несовместимы друг с другом. В результате относительно большие сообщества вынуждены создавать, поддерживать и переписывать несколько версий одной и той же библиотеки для разных фреймворков. В настоящий момент в сообществе PHP-разработчиков зародилось движение PSR-стандартов, призванных объединить требования и интерфейсы компонентов так, чтобы их можно было использовать в разных фреймворках без дополнительной адаптации. Процесс находится только в начале пути, и на его решение уйдет несколько лет.

В Ruby-сообществе, благодаря гемам rack, rake и bundle, которые поддерживают все современные фреймворки и Ruby-сервера, эта проблема решена несколько лет назад. Все фреймворки и их компоненты уже совместимы друг с другом. Именно поэтому удачно созданный компонент − гем − быстро становится достоянием Ruby-сообщества.

Это позволило Ruby on Rails быстро развиваться и включать в себя другие Ruby-фреймворки: MiniTest и RSpec для тестирования приложений, Slim и Haml для шаблонизации HTML-кода, Capistrano для доставки приложений на сервер и т.д.

Ruby on Rails является своеобразным конструктором, в котором одни части могут быть заменены на другие. Не нравится шаблонизатор ERB из коробки? Одной строкой его можно заменить на альтернативные Slim и Haml. Не подходит тестирование на базе MiniTest? Без труда его можно заменить на RSpec.

Все это вместе позволило освободить силы сообщества и направить их на дальнейшее развитие фреймворка, который на сегодня считается одним из лучших в мире. Более того, в конкурирующих сообществах стали создаваться фреймворки по образу и подобию Ruby on Rails: в PHP — Laravel, в Python — Django.

Ruby — полностью объектно-ориентированный язык с богатыми возможностями мета-программирования и расширения языка. Ruby on Rails интенсивно использует эти возможности, открывая и модифицируя практически все стандартные классы Ruby. Очень часто говорят, что Ruby on Rails — это другой язык программирования по сравнению с Ruby. Изучать Ruby on Rails можно, не зная Ruby. Многие изучают Ruby уже после изучения Ruby on Rails.

Ruby on Rails — это система, интенсивно использующая объектно-ориентированные возможности языка Ruby и паттерны проектирования. Поэтому прежде, чем мы начнем его использовать, стоит познакомиться с тем как сформировалось объектно-ориентированное программирование и узнать, почему оно важно.

Исторический экскурс

Программирование — относительно молодая инженерная область, в которой не сложились традиция и последовательное развитие предметной области. Связано это с бурным развитием микроэлектроники и элементной базы, которые более 50 лет подчиняются закону Мура. Согласно последнему, количество транзисторов, размещаемых на одном кристалле интегральной схемы, удваивается каждые два года. Таким образом, последние 50 лет микроэлектроника, а вслед за ней и компьютеры, развивались по экспоненциальному закону.

Транзисторы и интегральные схемы предназначены для компьютеров, а языки программирования — для людей. При помощи языков программирования разработчики «объясняют» компьютерам, какие задачи необходимо выполнить. Чтобы успеть за бурно развивающейся элементной базой, программисты вынуждены были не только создавать новые языки, но и менять концепции, лежащие в их основе.

Первоначально приходилось иметь дело с номерами команд интегральных схем. Например, в случае центрального процессора есть какие-то команды: сложить два числа, переместить число из оперативной памяти в регистр процессора или наоборот и т.д. Всем этим командам назначаются номера, после которых следуют аргументы. Такие номера называются кодами, а кодирование при помощи их называется программирование в кодах. Результат такого кодирования — исполняемый файл, содержимое которого можно напрямую передавать процессору.

Программировать в кодах крайне неудобно, так как люди, как правило, плохо запоминают числа, поэтому команды стали обозначать мнемониками: MOVE — переместить, ADD — сложить числа и т.д.

Перед использованием программы мнемоники транслировались в коды процессора, в результате чего получался все тот же исполняемый файл, как и в случае программирования в кодах. Такую систему мнемоник, а также первых приемов программирования с их использованием, стали называть ассемблерами.

Так как закон Мура приводил к быстрой смене поколений и архитектуры процессоров, программировать при помощи ассемблеров было по-прежнему неудобно. Каждый ассемблер ориентировался на конкретный процессор и программы, созданные с использованием ассемблера, не работали под альтернативные процессоры. Их приходилось каждый раз переписывать и адаптировать. В силу ориентации на архитектуру процессора разработать универсальный ассемблер не представлялось возможным. Кроме того, ассемблеры были довольно низкоуровневыми, в них были слабо развиты средства для повторного использования кода.

Следующим этапом эволюции было создание универсальных языков программирования, которые бы не зависели от архитектуры процессора. Так появились языки программирования высокого уровня. Так как они были более долговечны, чем ассемблеры, в них начали развиваться средства для повторного использования кода. В первую очередь, это процедурный и функциональный подход. Первые процедурные языки программирования — Fortran и C — сформировали современный облик языков программирования. Особенно глубокий след оставил C, на котором была написана первая операционная система-долгожитель — UNIX.

Универсальные языки программирования привели к развитию операционных систем, баз данных и сложного программного обеспечения: игр, офисных пакетов. Программное обеспечение становилось все более и более сложным. Наличие нескольких операционных систем привело к необходимости создания переносимых между ОС программ. С одной стороны, это повлияло на стандартизацию языков программирования. С другой — привело к еще большему усложнению ПО. Средств, заложенных в процедурные языки программирования, стало не хватать.

В ответ на все возрастающую сложность стали создаваться новые специализированные языки программирования. Например, бухгалтерский учет оперирует такими понятиями, как сотрудник, организация, налог, валюта, расчетный счет, подзаконный акт. Языки программирования оперируют либо собственной предметной областью (числа, строки, ветвление, цикл, процедура), либо предметной областью операционной системы — файл, директория, адресация, сокет. Создавая программы, мы оперируем терминами компьютера, а не предметной области. Специализированный язык программирования сразу может представить сущность сотрудника, налога и т.д. В результате любую бухгалтерскую проблему просто программировать, так как оперировать можно терминами предметной области. Поэтому на рубеже 80-90-х годов прошлого столетия произошло взрывообразное увеличение языков программирования. Специализированные языки используются по сей день, например, язык запросов к базам данных SQL или язык специализированных систем бухгалтерского учета SAP и 1С.

Язык программирования, предназначенный для предметной области, крайне эффективен. Он позволяет сложную проблему решать просто. Реализовав язык программирования и подготовив специалистов, которые им владеют, можно очень эффективно и быстро решать любые проблемы предметной области. Сложность проблемы при этом никуда не девается, она заложена в реализацию языка и его изучение. В этом же заключается слабость такого подхода: создание языка программирования занимает длительный срок — как правило, несколько лет. Программисты не могут изучать и исследовать бесконечное количество разных языков программирования. Более того, чтобы процесс подготовки проходил успешно, необходима устойчивая группа из программистов на этом языке. Из этой группы будут постоянно уходить разработчики, и она должна генерировать достаточное количество статей, книг, конференций, открытых проектов для вовлечения новых членов сообщества. В противном случае язык программирования не будет поддерживаться, а специалистов, знакомых с ним, будет трудно нанять.

В 90-х годах прошлого столетия компьютеры и сеть Интернет проникли буквально во все области жизни цивилизации. Количество предметных областей, где используются компьютеры, не только возросло на порядок, но и постоянно увеличивается уже в результате трансляции части культуры человечества внутрь сети и гаджетов. Путь создания новых специализированных языков программирования оказался затратным. При этом программы, созданные на одном языке программирования, по-прежнему недоступны на другом языке программирования без адаптации.

Трудность создания и поддержки специализированных языков программирования привела к развитию объектно-ориентированного программирования (ООП) как в уже существующих, так и в новых универсальных языках программирования. Идея ООП заключается в создании объектов, которые разработчик мог бы заранее снабдить заданным поведением, научить откликаться на вызовы и операторы, ранее реализуемые только создателем языка программирования. В результате разработчики, использующие объектно-ориентированные программы, получили возможность создания мини-языков программирования для своей предметной области с одной стороны и возможность повторного использования своего кода с другой. Сначала создается мини-язык предметной области при помощи объектов и классов, а затем с использованием этого языка разрабатывается программа. Такой подход позволяет сократить количество существующих языков программирования.

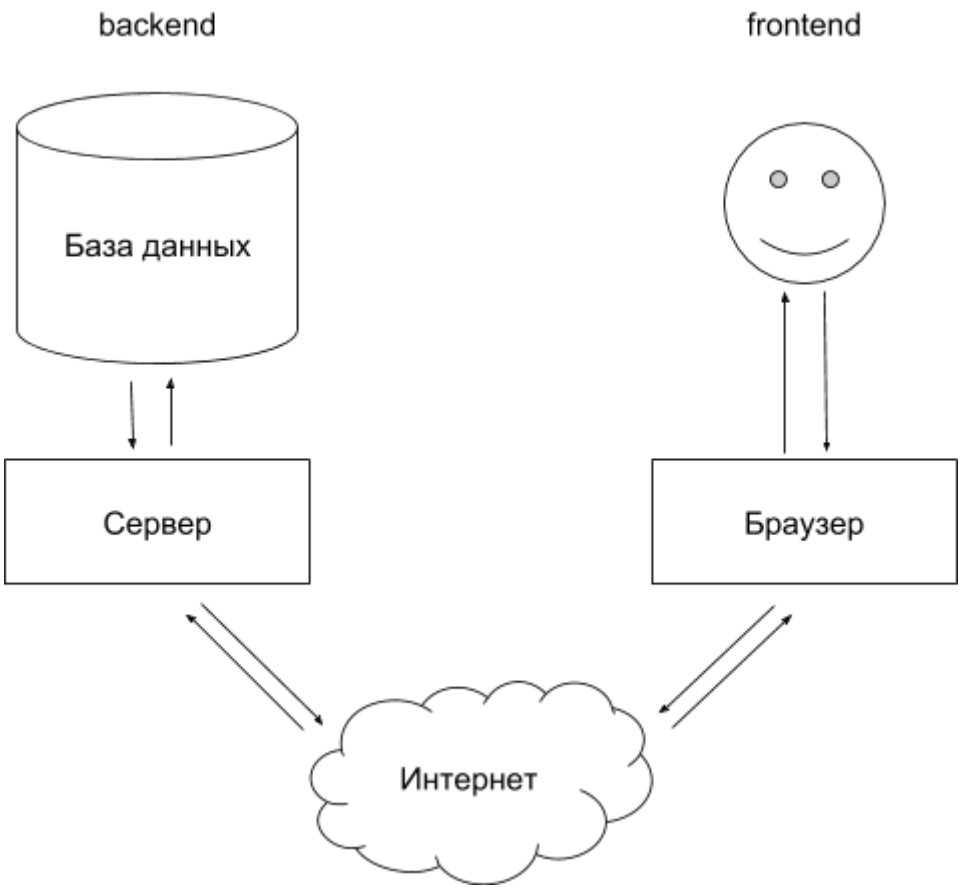
Паттерны проектирования

Так как идея объектно-ориентированного программирования не зависит от конкретного ЯП, в области разработки объектно-ориентированных программ стали накапливаться удачные приемы и архитектурные решения. Такие типовые решения стали называть паттернами или шаблонами проектирования.

Паттернов довольно много: стратегия, одиночка, шаблонный метод, строитель, адаптер и т.д. Они делятся на определенные классы: порождающие, структурные, поведенческие, составные. Паттерны проектирования — это специальная область проектирования и архитектуры приложений, которой можно посвятить отдельный курс. Однако мы не сможем обойти один из паттернов: MVC — Model-View-Controller. Ruby on Rails представляет собой реализацию этого паттерна проектирования. Для того, чтобы можно было понять, какой компонент за какую часть языка отвечает, необходимо изучить его более детально.

Как устроено Web-приложение?

Для того, чтобы понять паттерн, рассмотрим типичное Web-приложение. Ниже представлена упрощенная модель клиент-серверного взаимодействия.



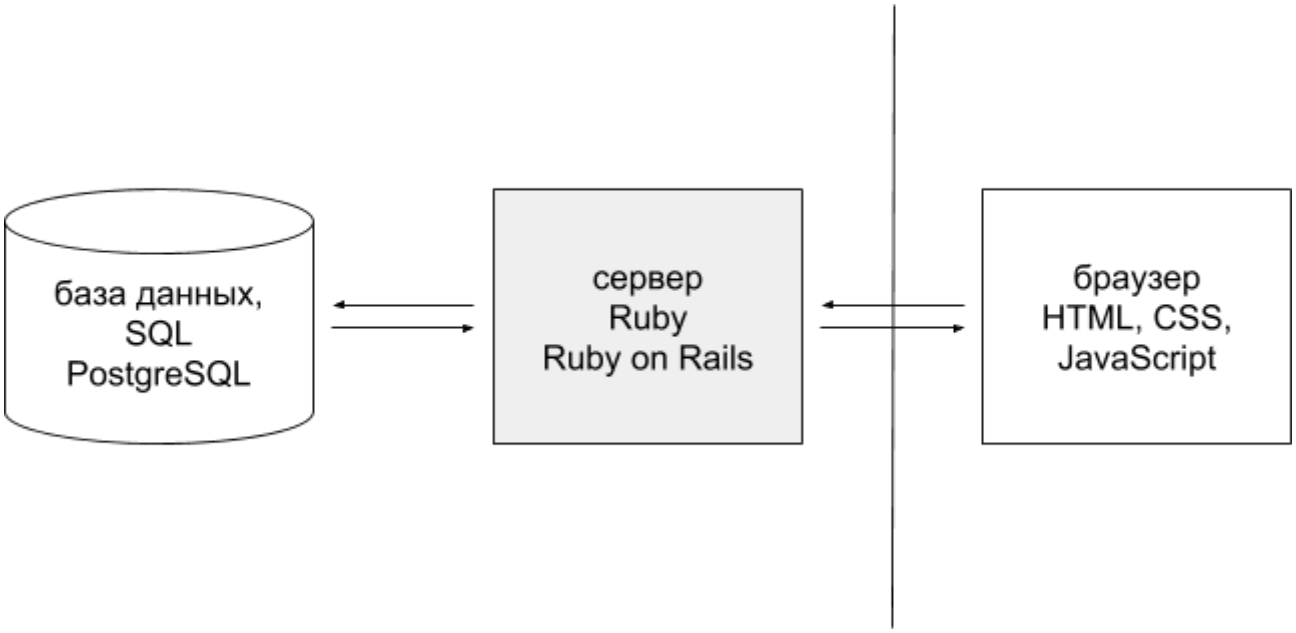
Пользователь просматривает сайт через браузер, который через сеть Интернет отправляет HTTP-запросы к серверу и получает от него HTTP-ответы. Таким образом, Web-приложение всегда является распределенным.

Как минимум одна часть его работает на сервере с использованием одного из серверных языков программирования (Ruby, PHP, Python, Java и т.д.). Серверная часть при этом может обращаться к одной или нескольким базам данных или к другим серверам и сервисам.

В браузере пользователя работает другая часть Web-приложения, которая использует клиентские технологии: язык разметки HTML, графическое оформление при помощи каскадных таблиц стилей, язык программирования JavaScript. Клиентская часть может обращаться не к одному, а к нескольким разным серверам.

Все, что расположено на серверах, обычно называют backend или серверной частью сайта. Все, что отображается в браузере, обычно называют frontend или клиентской частью сайта. Соответственно, разработчики делятся на backend- и frontend-специалистов.

При разработке сайта Ruby-код всегда выполняется только на серверной части, так как браузер может интерпретировать только HTML-разметку, применять CSS-стили и выполнять JavaScript-код.



Когда мы разрабатываем сайт на Ruby on Rails, мы имеем дело в основном с серверной частью, представленной на схеме серым квадратом. Все, о чем мы будем говорить далее, будет сосредоточено в этом сером квадрате. Это не значит, что мы совершенно не будем касаться баз данных или клиентских технологий. Однако, в курсе они будут рассматриваться именно с точки зрения RoR-разработчика.

Введение в MVC

В схеме, представленной на рисунке выше, каждая часть занимается своим собственным делом. База данных осуществляет долговременное хранение данных: зарегистрированные пользователи, написанные ими тексты и т.п. Сервер − обработку поступающих запросов и формирование ответов с использованием данных из базы. Браузер представляет информацию пользователю в виде, доступном для просмотра обычным человеком, а также элементы управления для ввода информации: регистрационные данные или логин с паролем.

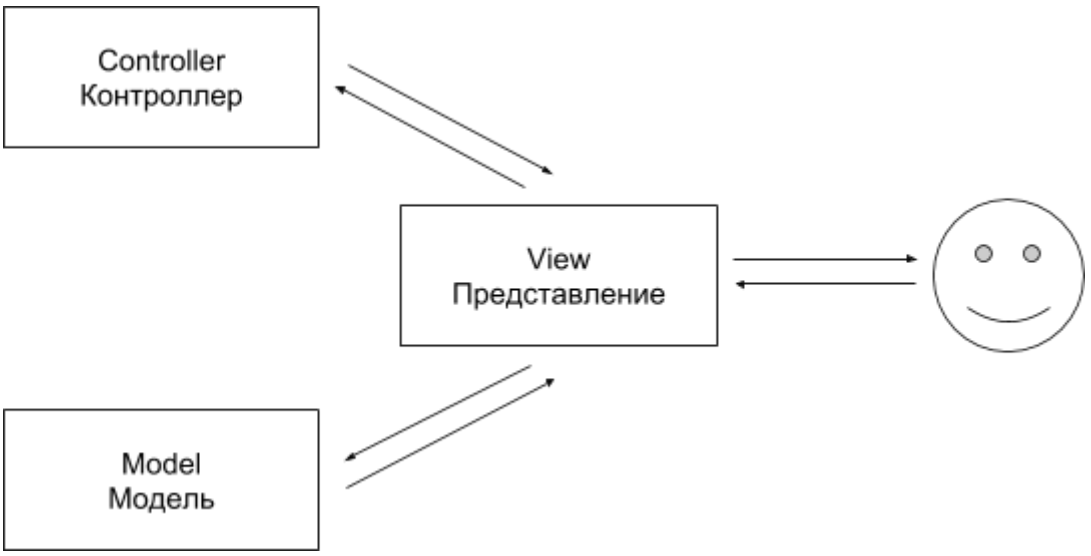
Таким образом, информация, которой оперирует приложение, находится в трех разных местах: в базе данных, сервере и браузере. Их нужно синхронизировать друг с другом. Если что-то появляется в базе данных, информация должна становиться доступной пользователю. Если пользователь ввел сообщение в браузере и отправил запрос на его сохранение, это сообщение должно попасть в базу данных. Если сообщение было скрыто администратором, а у пользователя нет прав на просмотр скрытых сообщений, оно не должно быть доступно для просмотра.

Такой синхронизацией данных, координацией деятельности, занимается центральная часть приложения на Ruby on Rails. Чтобы упорядочить процесс синхронизации, сервер реализуется с использованием паттерна проектирования MVC: Model-View-Controller (Модель-Представление-Контроллер).

Задача MVC-паттерна заключается в разделении данных (модель), кода, ответственного за формирование представления (HTML, CSS, JavaScript) и бизнес-логики (контроллеры).

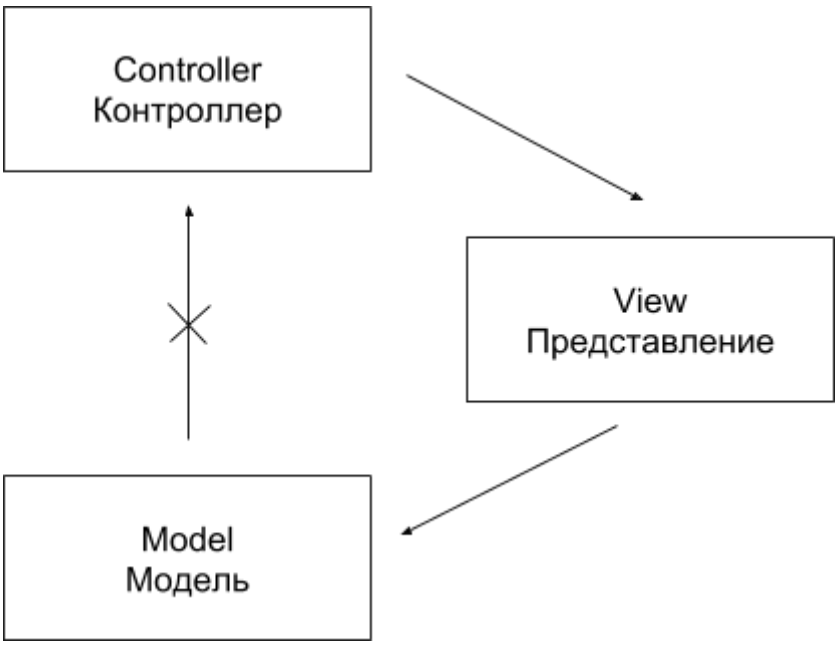
MVC — это очень старый паттерн, который впервые был реализован в языке программирования Smalltalk в 80-х годах прошлого столетия. Сети Интернет тогда не существовало и паттерн

использовался для программирования графического интерфейса десктопных приложений, например редактора Microsoft Word.

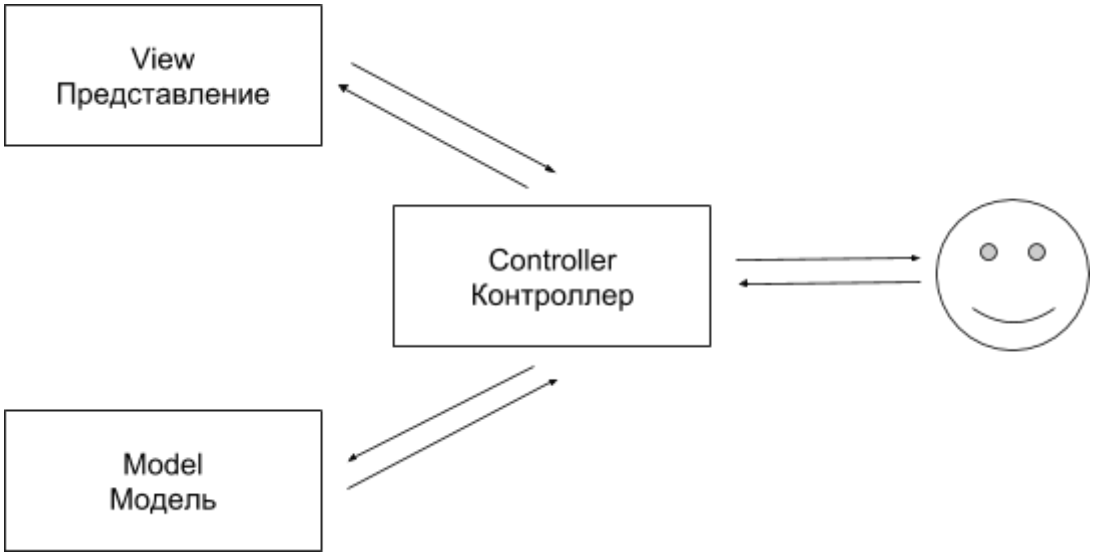


Пользователь взаимодействует с представлением: вводит текст, выбирает пункты меню, нажимает кнопки. Представление обращается к слою модели, который ответственен за полное представление документа от начала до конца — пользователь видит в представлении только часть документа. Как правило, документ может быть либо файлом, либо базой данных, эксклюзивный доступ к которым осуществляется через объект модели. Контроллер в этой схеме содержит классы и объекты, которые ответственны за бизнес-логику и вычисления: подсчет количества строк, поиск ошибок в тексте и т.д. и т.п.

Обратите внимание, что контроллер напрямую никогда не взаимодействует с моделью − все обращения идут через представление. С одной стороны, это позволяет избежать циклических синхронизаций, когда одно событие в приложении запускает цепочку событий, опять запускающих исходное событие. С другой − позволяет избежать связанности кода, когда изменения в моделях приводят к необходимости менять и представления, и контроллер.

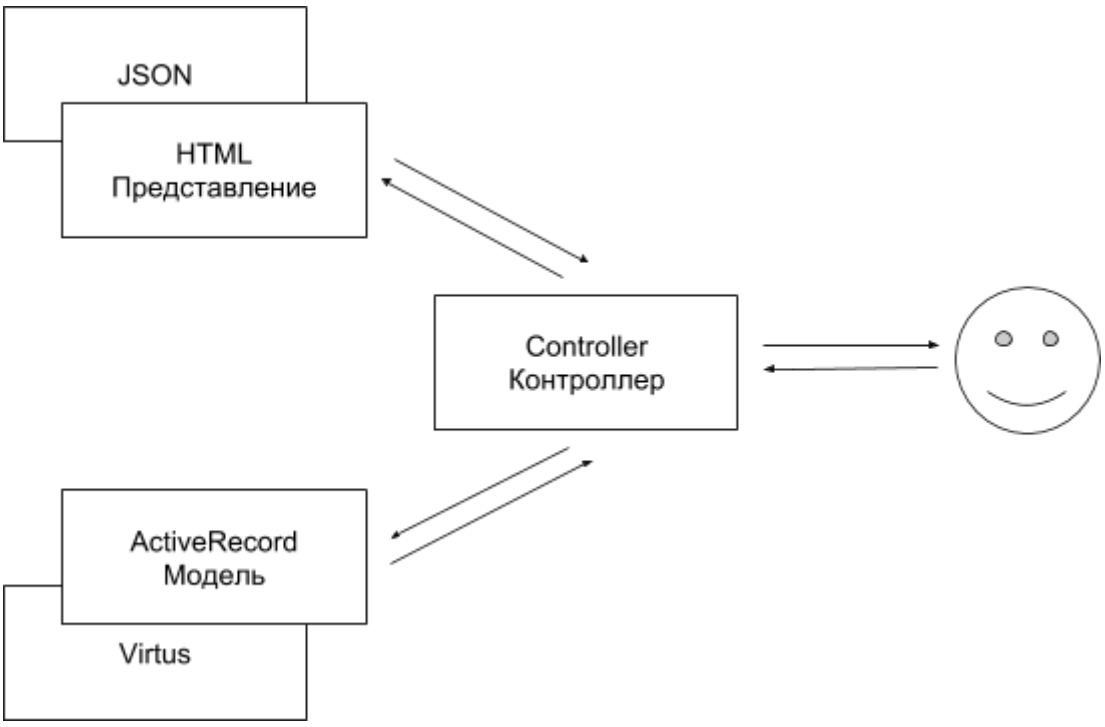


Рассмотренный вариант организации приложения называется MVC Model I. Он никогда не используется в Web-программировании. Вместо него применяется паттерн проектирования MVC Model II, у которого взаимодействие с пользователем осуществляется через контроллер.



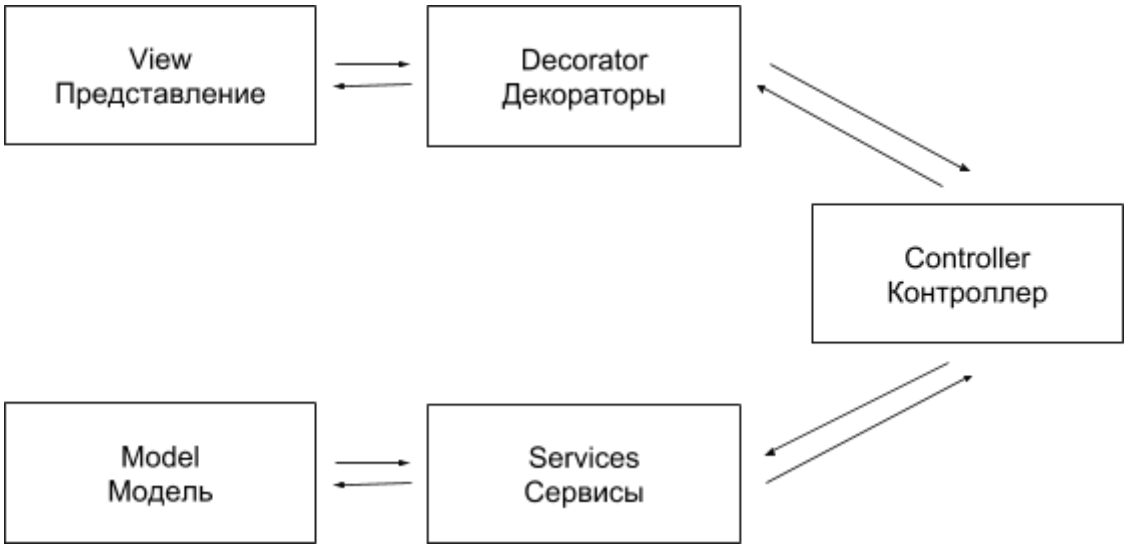
Обратите внимание, что идея разделения слоев приложения присутствует и в этом варианте MVC. Контроллер может обращаться к модели для получения или сохранения данных, например в базе данных. А также может обращаться к представлению для формирования ответа пользователю. При этом представление не может напрямую обращаться к модели, а модель − хранить в своем составе представления.

Разделение кода упрощает сопровождение и масштабирование сайта. Например, нам необходимо, чтобы информация для посетителей отображалась в виде HTML-страниц, а для обращений со стороны JS-кода − в JSON-формате. Тогда мы можем, не затрагивая слои контроллера или модели, просто добавить альтернативную реализацию представления:



Точно так же, если помимо базы данных мы захотим обращаться в JSON-API, мы можем задействовать Virtus-модели, а не модели ActiveRecord, предназначенные для работы с базами данных. Таким образом, мы можем локализовать работу с представлениями и источниками данных в классах определенной подсистемы. Изменения сайта в дальнейшем, расширение его функционала или масштабирование не приведут к тому, что нам придется переписывать всю систему с нуля.

В больших и сложных проектах каждый из компонентов MVC может дополнительно разбиваться на отдельные слои, например:



MVC выступает в качестве архитектурного каркаса, при помощи которого можно управлять масштабированием сложности Web-приложения.

Адрес сайта

Так как любое Web-приложение по своей природе является распределенным, при его создании мы вынуждены учитывать Web-среду. Если походить по любому сайту, можно обратить внимание, что при переходе с одной страницы на другую меняется адрес в адресной строке браузера. Например, мы можем находиться на главной странице GeekBrains, адрес которой:

https://geekbrains.ru/

При переходе в раздел «Курсы» адрес изменится на:

https://geekbrains.ru/courses

Давайте рассмотрим подробнее, как устроен адрес и как Web-приложение переключается между разными разделами и страницами сайта. В Интернет-адресе первым компонентом является протокол, как правило, http или https. Он сообщает, что используется обычная или защищенная SSL-версия протокола HTTP. Очень редко можно увидеть в адресе другой протокол, например FTP:

ftp://de.releases.ubuntu.com/xenial/ubuntu-16.04.3-server-amd64.iso

Далее следует последовательность **://**, после которой указывается доменное имя сайта. Доменное имя — это зарегистрированное в службе DNS имя. С его помощью можно найти хост, на котором размещен сайт. Домены разбиваются по зонам:

- ru — домен первого уровня;
- geekbrains.ru — домен второго уровня;
- projects.geekbrains.ru — домен третьего уровня.

Как правило, за каждым стандартным протоколом закреплен его порт, который может быть размещен после домена. Например, протоколу HTTP соответствует 80 порт, и можно было бы записать адрес следующим образом:

http://geekbrains.ru:80/

Протоколу HTTPS соответствует 443 порт, и https-адрес можно было бы записать так:

https://geekbrains.ru:443/courses

Если используется стандартный порт, он всегда опускается, поэтому в адресах вы его никогда не видите. Тем не менее мы будем использовать порты для работы с локальной копией сайта, так как сайт будем запускать на нестандартном порту 3000.

http://localhost:3000/

Доменное имя localhost предназначено для локальной разработки и прописано в файле */etc/hosts* в UNIX-подобных операционных системах или в файле *C:\System32\Drivers\Etc\Hosts в Windows*.

После доменного имени и порта следует путь к ресурсу сайта уже на сервере. Разным путям соответствуют разные разделы или страницы сайта.

Введение в протокол HTTP и роутинг

Когда в адресную строку браузера помещается ссылка вида **https://geekbrains.ru/courses**, браузер формирует HTTP-запрос. В упрощенном виде его можно представить следующим образом:

```
GET /courses
Host: geekbrains.ru
...
```

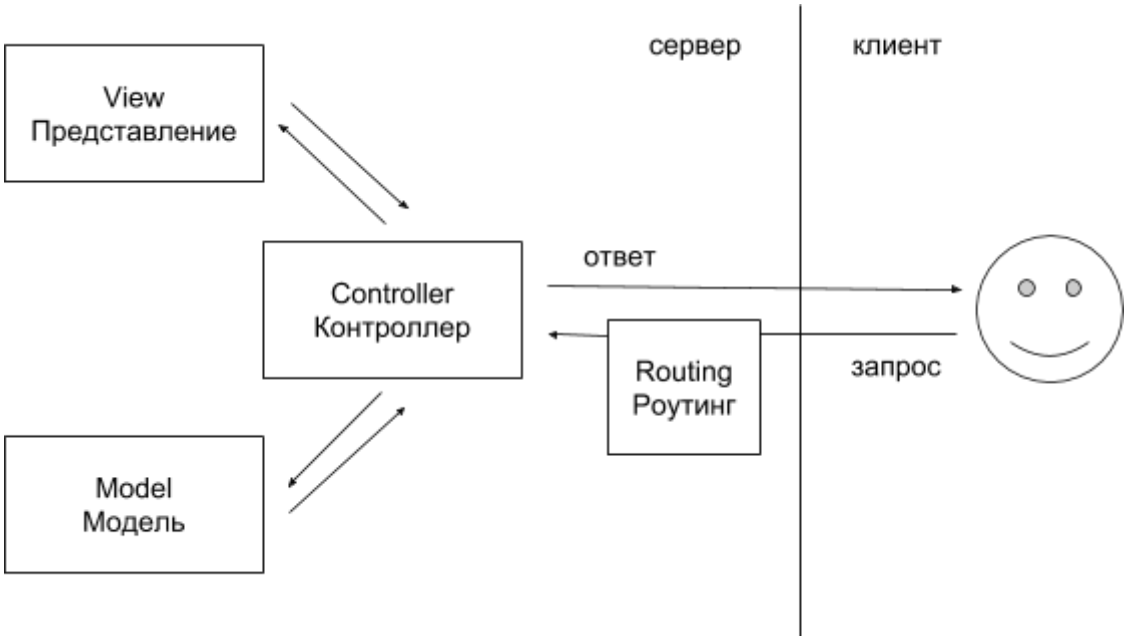
GET — это метод протокола HTTP. С разными методами мы более подробно познакомимся на втором занятии.

/courses — это путь к запрашиваемому ресурсу.

Host: geekbrains.ru — HTTP-заголовок Host, который сообщает, какому сайту адресован запрос. Вообще HTTP-заголовков может быть довольно много. Все они имеют примерно одинаковую организацию: ключ-значение.

Вся информация из HTTP-запроса попадает на сервер, обрабатывается и передается внутрь RoR-приложения. Прежде чем передать приложению, ее обрабатывает гем rack, который представляет своего рода адаптер для всех современных Web-серверов в экосистеме языка Ruby. Rack служит мостом между внешним миром и RoR-приложением.

После этого запрос отправляется в подсистему роутинга, где, исходя из метода протокола HTTP (в примере выше это GET), а также пути (/courses), принимается решение, какой из контроллеров будет обрабатывать этот запрос. Таким образом, схему MVC можно дополнить подсистемой роутинга:



RoR-приложение и сайты могут быть устроены гораздо более сложным способом. Схема, представленная выше, минимальна, она есть в любом RoR-сайте.

Стек технологий

До 90-х годов прошлого столетия изучения лишь одного языка программирования было достаточно, чтобы создавать конечные программные продукты. Сами программы не имели длительного жизненного цикла, и работа с компьютером подразумевала постоянную разработку мини-программ. По мере того, как программы становились сложнее и жили дольше, их разработка стала профессией. Более того, благодаря специализации в дальнейшем появилось несколько профессий: дизайнеры, архитекторы, программисты, технические писатели, менеджеры проектов и т.п. Сам же процесс превратился в технологическую цепочку, в которой используются различные языки программирования, текстовые и графические редакторы, различные вспомогательные утилиты. Таким образом, для создания современного сайта требуется не один язык программирования, а несколько, не одна технология, а целая группа. Для разного типа программного обеспечения требуются разные технологические цепочки. Например, в Web-программировании такие цепочки подразумевают:

- Сбор требований и оформление их в техническое задание.
- Создание дизайна и оформление его в макет.
- Верстку по макетам и предоставление HTML-кода.
- Программирование серверной части сайта.
- Программирование клиентской части сайта.
- Тестирование приложения.
- Развертывание приложения в продакшн-среде.
- Техническое обслуживание и поддержку.

Многие из этих стадий зависят от других и не могут быть начаты раньше, чем будет достигнут прогресс по предыдущим стадиям. Каждый из пунктов представленного выше списка может быть разбит на стадии (процесс *декомпозиции*) и требовать нескольких технологий.

Фреймворк Ruby on Rails может использоваться для разработки Web-приложений (сайтов) от начала до конца. Его возможности сосредоточены в основном на серверной, клиентской частях, в области тестирования и развертывания приложения.

Фреймворк не покрывает следующие пункты:

- Сбор требований и оформление их в техническое задание.
- Создание дизайна и оформление его в макет.
- Верстку по макетам и предоставление HTML-кода.

Ruby on Rails не монолитен: многие его части, которые идут из коробки, могут быть заменены альтернативными решениями. Например, для тестирования кода по умолчанию используется фреймворк MiniTest, который в 85% случаев заменяется на RSpec. Шаблонизатор ERB почти всегда заменяется на Haml или Slim. По сути, Ruby on Rails выступает в виде лего-конструктора, в котором различные компоненты очень легко могут быть заменены на более удобные и подходящие к текущему проекту.

Web-приложение — это важная часть программного обеспечения, необходимого для работы сайта, но лишь часть. Для хранения данных необходима база данных. Перед приложением часто ставят более производительный Web-сервер, например, nginx. Для командной работы необходима система контроля версий, как правило, git. Все эти программы должны работать в какой-либо операционной системе: в случае Web это часто Linux. Таким образом, для запуска большого сайта требуется большое количество технологий, программного обеспечения.

Группу связанных технологий, необходимых для разработки программного обеспечения, часто называют *стеком технологий*.

Например, при разработке сайта на Ruby технологический стек может выглядеть следующим образом:

- Ruby on Rails;
- MySQL;
- Memcached;
- Haml;
- CoffeeScript;
- MiniTest;
- Unicorn.

Такой стек расшифровывается следующим образом: сайт, разработанный на Ruby on Rails, будет использовать базу данных MySQL, систему кэширования memcached, шаблонизатор Haml, CoffeeScript для JavaScript-кода, MiniTest для тестирования и Ruby-сервер Unicorn для запуска сайта.

На курсе мы будем использовать следующий технологический стек:

- Ruby on Rails;
- PostgreSQL;
- Slim;
- ES6;
- RSpec;
- Puma.

Будем использовать базу данных PostgreSQL, шаблонизатор Slim, JavaScript-код в ES6-варианте, фреймворк RSpec для тестирования и Ruby-сервер Puma для запуска приложения.

Базы данных

База данных — это специальным образом организованный файл или несколько файлов для долговременного хранения информации. Например, списка регистрационных данных пользователей сайта. Обработкой этих файлов занимается система управления базами данных (СУБД), которую очень часто называют просто базой данных. Использование СУБД предоставляет определенные преимущества. В частности, СУБД выполняет много работы, которую иначе пришлось бы каждый раз программировать:

- Реализует клиент-серверную технологию доступа к данным;
- Предоставляет специальный язык запросов, например SQL, в случае реляционных баз данных;
- Решает проблему синхронизированного доступа к файлам, разрешая конфликты одновременного обращения нескольких клиентов;
- Кэширует данные в более быстрой оперативной памяти.

Базы данных могут быть реляционными, как правило, реализующими стандартный язык запросов SQL, а могут быть NoSQL-базами данных. У NoSQL СУБД есть свои особенности: как правило, они не используют жесткий диск в качестве основного хранилища, полностью располагая данные в оперативной памяти. Данные в основном хранятся в виде документов или пар «ключ-значение». NoSQL-базы данных не поддерживают язык запросов SQL: либо реализуют собственный язык запросов, либо ориентируются на язык высокого уровня (Lua, JavaScript). В результате, такие базы данных не связаны жесткими ограничениями стандарта SQL.

Существует большое количество самых разнообразных баз данных, как коммерческих, так и свободных. Подавляющее большинство Rails-разработчиков использует PostgreSQL, самую популярную и развитую на сегодняшний момент базу данных с открытым кодом.

Шаблонизаторы

В мире Rails редко используется HTML. Для разметки страниц используют какие-либо препроцессоры, обеспечивающие более компактный код и интеграцию результатов вычисления ruby-кода. Чаще всего это haml или slim. Они значительно упрощают конструкцию html. Рассмотрим аналогичный код на slim и html.

HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= full_title(yield(:title)) %></title>
     <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track'
=> true %>
    <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
    <%= csrf_meta_tags %>
  </head>
  <body>
    <header class="navbar">
      <div class="logo">
        <%= link_to "sample app", 'root_path', id: "logo" %>
        <nav>
          <ul class="navbar-right">
            <li><%= link_to "Home",    'root_path' %></li>
            <li><%= link_to "Help",    'help_path' %></li>
            <li><%= link_to "Log in", 'login_path' %></li>
          </ul>
        </nav>
      </div>
    </header>
    <div class="main">
      <%= yield %>
    </div>
  </body>
</html>
```

Аналогичный код на slim:

```
doctype html
html
  head
    title = full_title(yield(:title))
     = stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track'
=> true
    = javascript_include_tag 'application', 'data-turbolinks-track' => true
    = csrf_meta_tags
  body
    header.navbar
      .logo
        = link_to "sample app", 'root_path', id: "logo"
        nav
```

```
        ul.navbar-right
          li
            = link_to "Home",    'root_path'
          li
            = link_to "Help",    'help_path'
          li
            = link_to "Log in", 'login_path'
  .main
    = yield
```

Код на slim выглядит гораздо компактнее и легче читается. Если вы много работали с html, вам может показаться обратное, но, по опыту, практика использования препроцессоров окупается. Рекомендуется использовать именно slim. Он работает быстрее по сравнению с аналогичными шаблонизаторами-препроцессорами. Синтаксис языка очень прост и похож на синтаксис ruby, и его можно освоить, изучив пример страницы на официальном сайте проекта.

CoffeeScript и ES6

Нередко можно встретить, что в проектах на rails вместо JavaScript используется CoffeeScript. Это язык программирования, транслируемый в JavaScript. Он добавляет много синтаксического сахара и позволяет писать клиентскую часть веб-приложений в стиле Ruby. С примерами и синтаксисом можно глубже познакомиться на официальном сайте.

Впрочем, в качестве альтернативы все чаще и чаще используется нативный JavaScript в варианте ES6 или ES2017. Это новые версии JavaScript, которые пока не полностью поддерживаются браузерами. Однако, благодаря библиотеке Babel, можно уже сейчас использовать новый JavaScript, переводя его перед использованием в традиционный JavaScript ES5, который браузеры отлично понимают.

Ruby on Rails может поддерживать любые новые варианты языка JavaScript, как при самостоятельном управлении проектом, так и при совместной работе с фронт-частью, разрабатываемой на Node.js.

CSS препроцессоры

Как и в случае с HTML и Javascript, для CSS тоже существуют препроцессоры, упрощающие разработку. Их достаточно много. Фаворитами являются scss, sass, stylus.

Системы контроля версий

Нужны для версионирования проекта и комфортной совместной разработки одного проекта. Самые популярные системы контроля версий на сегодня – git, mercurial, svn. В подавляющем большинстве случаев новые проекты разрабатываются с использованием системы контроля версий git. Познакомиться с git можно по книге GitPro.

Настройка рабочего окружения

Разработка на Ruby on Rails довольно чувствительна к рабочему окружению, так как большинство компонентов, используемых для создания приложения, ориентировано для работы в UNIX-подобных операционных системах.

К UNIX-подобным операционным системам относятся Linux и Mac OS X. Если есть возможность использовать одну из этих операционных систем, это наиболее предпочтительный вариант.

Если вы используете операционную систему Windows версии 10, следует рассмотреть возможность установки подсистемы UNIX.

Если ни один из предыдущих вариантов не подходит, можно настроить UNIX-окружение, используя виртуальную машину. Это значит выделить часть ресурсов компьютера для изолированной операционной системы Для этого необходимо установить систему виртуализации VirtualBox. Для удобства работы с проектом из командной строки можно использовать Vagrant.

Vagrant

Vagrant — это интерфейс для управления и систематизации виртуальных машин. Он позволяет сконфигурировать виртуальное рабочее окружение внутри виртуальной машины, чтобы упростить процесс создания этого окружения для разных машин и разработчиков. Такой подход позволяет гарантировать одинаковое окружение у любого разработчика, какую бы операционную систему он ни использовал, и эмулировать окружение будущего сервера, на котором будет развернуто приложение. Также Vagrant отличается удобством конфигурации: файл настроек представляет собой скрипт Ruby.

Итак, чтобы настроить UNIX-окружение, необходимо установить на свою машину:

1. VirtualBox.
2. Vagrant.
3. Если вы пользователь Windows, вам дополнительно нужен эмулятор терминала с поддержкой ssh, например git bash или cygwin.

Вместо VirtualBox вы можете выбрать любую другую систему управления виртуальными машинами, поддерживаемую Vagrant, но VirtualBox значительно проще альтернатив.

После того, как Vagrant установлен, создадим на компьютере папку нового проекта и, перейдя в нее, с помощью консоли выполним команду:

```
vagrant init ubuntu/xenial64
```

В результате в рабочей директории появится файл с названием *Vagrantfile*. В нём содержатся настройки будущей виртуальной машины на ubuntu 16.04 (кодовое название xenial64).

Прежде чем запустить эту машину, подумаем о будущем и пробросим порт Rails сервера на хост. Это можно сделать, добавив сразу после *config.vm.box = "ubuntu/xenial64"* строчку:

```
config.vm.network "forwarded_port", guest: 3000, host: 3000
```

.

Далее инициализируем и запустим виртуальную машину с помощью команды:

```
vagrant up
```

После этой команды Vagrant скачает и развернет образ виртуальной машины с минимальной конфигурацией. Это может быть долго. Все зависит от скорости вашего соединения с интернетом. Для того, чтобы получить доступ к виртуальной машине, следует воспользоваться командой:

```
vagrant ssh
```

В качестве альтернативы Vagrant в настоящий момент чаще используется Docker. Это легковесная система виртуализации, которая используется для построения изолированных контейнеров в промышленной разработке и облачных системах.

RVM: установка Ruby

RVM (Ruby Version Maschine) позволяет легко управлять различными версиями Ruby, установленными на одной машине. Это может быть удобно, чтобы зафиксировать одну и ту же версию Ruby у разных разработчиков и на сервере с продакшеном приложения. Кроме того, это упрощает процесс миграции приложения с одной версии на другую.

Воспользуемся инструкциями с официального сайта RVM и выполним по очереди команды:

```
gpg --keyserver hkp://keys.gnupg.net --recv-keys
409B6B1796C275462A1703113804BB82D39DC0E3
sudo apt-get install curl
curl -sSL https://get.rvm.io | bash
source /home/ubuntu/.rvm/scripts/rvm
rvm install 2.4
```

В результате выполнения этих команд на виртуальной машине будут установлены RVM и Ruby версии 2.4. Важно устанавливать Ruby от текущего пользователя, а не от суперпользователя, поэтому при выполнении команды rvm не следует использовать sudo.

rbenv: установка Ruby

Существует альтернативный способ установить Ruby при помощи rbenv, который в последнее время чаще используется для установки Ruby, особенно в продакшн окружении на серверах:

```
git clone https://github.com/rbenv/rbenv.git ~/.rbenv
cd ~/.rbenv && src/configure && make -C src
echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
~/.rbenv/bin/rbenv init
curl -fsSL https://github.com/rbenv/rbenv-installer/raw/master/bin/rbenv-doctor |
bash
mkdir -p "$(rbenv root)"/plugins
git clone https://github.com/rbenv/ruby-build.git "$(rbenv
root)"/plugins/ruby-build
rbenv install 2.3.1
```

Здесь также приводится последовательность команд, необходимых для установки Ruby версии 2.3.1.

Установка Ruby on Rails

После того, как Ruby установлен, в командной строке становится доступна утилита ruby для запуска ruby-программ, а также утилита gem, позволяющая загружать библиотеки языка Ruby — гемы. Ruby on Rails в виде гема rails можно установить при помощи следующей команды:

```
gem install rails
```

Установка дополнительного ПО

Гем Rails имеет в качестве зависимостей ряд других гемов. Те, в свою очередь, могут зависеть от дополнительного программного обеспечения и библиотек. Поэтому, чтобы избежать ошибок при запуске и эксплуатации проекта, рекомендуется сразу установить ряд наиболее популярных библиотек и программ. Установим NodeJS. Это нужно для выполнения JavaScript-кода на стороне сервера:

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Устанавливаем менеджер пакетов yarn:

```
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee
/etc/apt/sources.list.d/yarn.list
sudo apt-get update && sudo apt-get install yarn
```

Кроме того, следует сразу установить базу данных PostgreSQL и библиотеки, необходимые для установки соединения с ней из Rails-проекта:

```
sudo apt-get install postgresql postgresql-contrib libpq-dev
```

После этого необходимо создать нового пользователя и базу данных для него. По умолчанию в PostgreSQL лишь один пользователь postgres, поэтому необходимо переключиться на него при помощи команды:

```
sudo -i -u postgres
```

После этого создаем нового пользователя (вместо ubuntu подставьте имя вашего пользователя):

```
createuser --interactive
Enter name of role to add: ubuntu
Shall the new role be a superuser? (y/n) y
```

При помощи команды exit выходим из сессии пользователя postgres и создаем базу данных при помощи команды createdb:

```
createdb
```

Если все прошло нормально, в системе будет доступен клиент psql, который позволит подключаться к базе данных PostgreSQL:

```
psql
psql (9.5.10)
Type "help" for help.

ubuntu=# SELECT current_user;
 current_user
--------------
 ubuntu
(1 row)
```

Первый проект

Все приложения в Rails начинаются с команды *rails new,* которая служит для быстрой генерации приложения-заготовки.

Команда Rails имеет многочисленные параметры, которые позволяют настроить будущий проект. В текущем курсе мы не сможем подробно рассмотреть тестирование, поэтому его можно отключить при помощи параметра **--skip-test**. Кроме этого, мы не будем использовать язык CoffeeScript для написания JavaScript-кода, поэтому используем параметр **--skip-coffee**, чтобы избежать его установки. Вместо этого будет использоваться гем webpacker для управления JavaScript-частью проекта, поэтому задействуем параметр **--webpack**, который настроит наш проект на его использование.

Мы также не будем использовать турболинки (перезагрузка перехода по страницам при помощи AJAX) и spring-сервер для быстрой перезагрузки части классов приложения без остановки сервера и последующего запуска. Поэтому задействуем параметры **--skip-turbolinks, --skip-sprin**g. Кроме того, сразу сообщим Ruby on Rails, что мы хотим использовать базу данных PostgreSQL (по умолчанию используется SQLite), и для этого используем параметр **--database=postgresql**.

Команда *rails new* для создания нового приложения может выглядеть следующим образом:

```
rails new blog --skip-test --skip-turbolinks --skip-spring --skip-coffee
--skip-action-cable
 --webpack --database=postgresql
```

Вы увидите, как эта команда создает заготовку нового приложения и устанавливает дополнительные зависимости. Чтобы убедиться, что все получилось, перейдём в директорию этого приложения и запустим Rails-сервер командой

```
rails s
```

Теперь можно открыть браузер, перейти по адресу http://localhost:3000 и убедиться, что всё работает, увидев сообщение об этом. Если по какой-то причине страница не загружается и в логах сервера нет сообщений, что было обращение к нему, попробуйте запустить rails s следующим образом:

```
rails s -b 0.0.0.0
```

Домашнее задание

1. Установите Ruby при помощи RVM или rbenv.
2. Установите гем rails и разверните приложение с именем blog.
3. Установите СУБД PostgreSQL.
4. Познакомьтесь с синтаксисом slim и sass.

Дополнительные материалы

1. Хабрахабр. Установка подсистемы UNIX в Windows 10.
2. VirtualBox.
3. Vagrant.
4. Введение в Docker.
5. Slim-Lang.
6. Сравнение haml, jade, slim.
7. Sass Scss.
8. Книга на русском языке по системе контроля версий Git: GitPro
9. Официальный сайт RVM.
10. Официальная страница rbenv.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Элизабет Фримен, Эрик Фримен, Кэти Сиерра, Берт Бейтс. Паттерны проектирования. Серия Head First.
2. Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования.
3. GUI Architectures. Martin Fowler. Часть 1.
4. Майкл Хартл. Ruby on Rails для начинающих. Изучаем разработку веб-приложений на основе Rails.
5. Ruby on Rails Guides.
6. Obie Fernandez. The Rails 5 Way.
7. Сэнди Метц. Ruby Объектно-ориентированное программирование.
8. Rails-соглашения.
9. Mitchell Hashimoto. Vagrant: up and run.
10. Эдриен Моуэт. Использование Docker.