COVERS RAILS 4.2

THE RUBY ON RAILS TUTORIAL

LEARN WEB DEVELOPMENT WITH RAILS

THIRD EDITION

BOOK AND SCREENCASTS BY MICHAEL HARTL

Ruby on Rails Tutorial

Learn Web Development with Rails

Michael Hartl

Оглавление

Предисловие

Моя бывшая компания (CD Baby) была одной из первых, громко перешедших на Ruby on Rails, а затем еще громче вернувшейся обратно на PHP (Google расскажет вам об этой драме). Эту книгу, написанную Майклом Хартлом, так высоко рекомендовали, что я должен был попробовать её, и *Ruby on Rails Tutorial* --- это именно то, что я использовал для возвращения к Rails.

Хотя я уже прошел через много книг по Rails, это одна из тех немногих, что, наконец, ``зацепила'' меня. Здесь все делается в рамках ``the Rails way'' --- способом, который казался мне крайне искусственным, но теперь, после изучения этой книги, я наконец ощутил комфортность и естественность этого подхода. К тому же, это единственная книга по Rails, которая соблюдает методику разработки через тестирование на всем своем протяжении; именно этот подход строго рекомендуется специалистами, но ранее я нигде не встречал такой его отчетливой демонстрации. Наконец, включение в демо-примеры таких тем как Git, GitHub, и Heroku --- автор дает вам почувствовать, что из себя представляет разработка проектов в реальном мире. Примеры кода в учебнике не являются ``вещью в себе'', они не изолированы от окружающего мира.

Линейное повествование --- отличный формат. Лично я прошел *Rails Tutorial* за три долгих дня,[1] выполняя все примеры и задачи в конце каждой главы. Делайте всё от начала и до конца, не перескакивая от одной темы к другой, и вы получите максимальную пользу.

Наслаждайтесь!

Derek Sivers (sivers.org)
Основатель CD Baby

[1] Это не стандартная ситуация! Обычно прохождение всего учебника занимает *гораздо* больше, чем три дня.

Благодарности

Ruby on Rails Tutorial во многом обязан моей предыдущей книге по Rails --- *RailsSpace*, и, следовательно, моему соавтору Aurelius Prochazka. Я хотел бы поблагодарить Aure как за работу, которую он проделал над прошлой книгой, так и за поддержку этой. Я также хотел бы поблагодарить Debra Williams Cauley, редактора обеих этих книг; до тех пор, пока она не прекратит брать меня на бейсбол, я буду продолжать писать книги для нее.

Я хотел бы поблагодарить огромное количество Ruby-стов, учивших и вдохновлявших меня на протяжении многих лет: David Heinemeier Hansson, Yehuda Katz, Carl Lerche, Jeremy Kemper, Xavier Noria, Ryan Bates, Geoffrey Grosenbach, Peter Cooper, Matt Aimonetti, Mark Bates, Gregg Pollack, Wayne E. Seguin, Amy Hoy, Dave Chelimsky, Pat Maddox, Tom Preston-Werner, Chris Wanstrath, Chad Fowler, Josh Susser, Obie Fernandez, Ian McFarland, Steven Bristol, Pratik Naik, Sarah Mei, Sarah Allen, Wolfram Arnold, Alex Chaffee, Giles Bowkett, Evan Dorn, Long Nguyen, James Lindenbaum, Adam Wiggins, Tikhon Bernstam, Ron Evans, Wyatt Greene, Miles Forrest, Sandi Metz, Ryan Davis, Aaron Patterson, хороших людей из Pivotal Labs, команду Heroku, ребят из thoughtbot и команду GitHub. Наконец, многих, многих читателей --- слишком многих чтобы перечислять их всех здесь --- внёсших большое количество предложений по улучшению и сообщавших об ошибках во время написания этой книги, и я с благодарностью признаю их помощь в написании ее настолько хорошей, насколько это было возможно.

Об авторе

Майкл Хартл --- автор *Ruby on Rails Tutorial*, одного из лидирующих введений в веб-разработку, соосновательь Softcover --- платформы для самостоятельной публикации. Его предыдущий опыт включает в себя написание и разработку *RailsSpace* --- чрезвычайно устаревшего учебника по Rails, и разработку Insoshi, некогда популярной, а ныне устаревшей социальной сети, написанной на Ruby on Rails. В 2011 году Майкл получил Ruby Hero Award (Премия --- герой Ruby) за вклад в развитие Ruby-сообщества. Он закончил Гарвардский колледж, имеет степень кандидата физических наук, присвоенную в Калифорнийском технологическом институте[2], и является выпускником предпринимательских курсов Y Combinator[3].

[2]# Тут можно прочесть о нем на русском

[3]# Тут можно прочесть об этом фонде на русском

Копирайт и лицензия

[4]# Тут можно прочесть о ней на русском

[5]# Тут можно прочесть о ней на русском

Глава 1

От нуля к развертыванию

Добро пожаловать в *Учебник Ruby on Rails: Изучение веб-разработки с Rails*. Данная книга имеет своей целью научить вас разработке собственных веб-приложений, и нашим инструментом будет популярный фреймворк Ruby on Rails. Если эта тема является новой для вас, *Ruby on Rails Tutorial* даст вам полное представление о разработке веб-приложений, включая основополагающие знания о Ruby, Rails, HTML & CSS, базах данных, контроле версий, тестировании и развертывании --- достаточно для начала вашей карьеры веб-разработчика или предпринимателя в сфере компьютерных технологий. С другой стороны, если вы уже знакомы с веб-разработкой, эта книга быстро научит вас необходимым основам Rails-фреймворка, включая MVC и REST, генераторы, миграции, роутинг и встроенный Ruby. В любом случае, к тому времени, когда вы закончите *Ruby on Rails Tutorial*, вы будете готовы извлечь максимум пользы из более продвинутых книг, блогов и скринкастов (видеоуроков), являющихся частью процветающей образовательной экосистемы.[1]

Ruby on Rails Tutorial использует комплексный подход к веб-разработке: в процессе обучения вы напишете три примера приложений с возрастающей сложностью; начав с минимального *hello*-приложения (Раздел 1.3), перейдем к чуть более продвинутому *мини*-приложению (Глава 2), а далее мы полностью сфокусируемся на разработке большого *учебного* приложения (с Главы 3 по Главу 12). Как следует из их весьма общих названий, рарабатываемые в этом учебнике приложения не привязаны к какой-либо категории веб-сайтов; хотя финальный пример и будет иметь значительное сходство с весьма популярной социальной сетью (которая, по совпадению, изначально была также написана на Rails), в *Ruby on Rails Tutorial* акцент смещен в сторону общих принципов разработки, поэтому полученные знания станут для вас надежным фундаментом вне зависимости от того, какие виды приложений вы хотите создавать.

Одним из часто задаваемых вопросов является ``какая квалификация и предварительная подготовка нужны для изучения веб-разработки с *Ruby on Rails Tutorial*?''. Как мы увидим в Разделе 1.1.1, веб-разработка --- это сложная тема, особенно для полных новичков. Хотя изначально учебник предназначен для читателей с некоторым опытом программирования, фактически, он нашел множество читателей среди весьма

[1] Самая свежая версия *Ruby on Rails Tutorial* может быть найдена на сайте книги http://www.railstutorial.org/. Если вы читаете эту книгу оффлайн, сверьте версию своего экземпляра с онлайн-версией на http://www.railstutorial.org/book.

начинающих разработчиков. Учитывая это, в данном (третьем) издании *Rails Tutorial* значительно снижен порог входа в разработку на Ruby on Rails (Блок 1.1).

Блок 1.1. Снижение порога входа

Для снижения порога входа для начинающих разработчиков в третьем издании *Ruby on Rails Tutorial* сделано следующее:

- Использование стандартной среды разработки, размещенной в облаке (Раздел 1.2), что позволяет обойти множество проблем, связанных с установкой и конфигурацией новой системы
- Использование ``предустановленного стека'' Rails, включая встроенный тестовый фреймворк ``minitest''
- Уменьшение количества внешних зависимостей (RSpec, Cucumber, Capybara, Factory Girl)
- Облегченный и более гибкий подход к тестированию
- Избавление от сложных конфигурационных опций (Spork, RubyTest)
- Уменьшение акцента на особенностях, свойственных данной версии Rails, и вместе с тем, усиление акцента на общих принципах веб-разработки

Я надеюсь, что эти изменения сделают третье издание *Ruby on Rails Tutorial* доступным для еще более широкой аудитории, чем предыдущие версии.

В этой первой главе мы начнем изучение Ruby on Rails с установки необходимого программного обеспечения и настройки нашего рабочего окружения (Раздел 1.2). Затем мы создадим наше первое Rails-приложение, названное `hello_app`. *Rails Tutorial* придает особое значение хорошим привычкам в программировании, так что сразу после создания нового Rails-проекта мы поместим его в систему контроля версий Git (Раздел 1.4). И, верите вы в это или нет, в этой главе мы даже разместим наше первое приложение в сети, *развернув* его в production (Раздел 1.5).

В Главе 2 мы создадим второй проект, целью которого будет демонстрация основ работы Rails-приложения. Для того, чтобы быстрее стартовать, в этом *мини-приложение* (названном `toy_app`) мы применим ``scaffolding'' (Блок 1.2) для генерации кода; поскольку этот код получится одновременно и уродливым, и сложным, то в Главе 2 мы сосредоточимся на взаимодействии с мини-приложением через *URI* (часто говорят *URL*)[2], используя веб-браузер.

Остальная часть учебника сфокусирована на разработке единственного большого *учебного приложения* (названного `sample_app`), на сей раз мы будем писать весь код с нуля. При этом мы применим сочетание

[2] *URI* --- сокращение от Uniform Resource Identifier (Единый Идентификатор Ресурсов), в то время как немного менее обобщающее *URL* является сокращением для Uniform Resource Locator (Единый Указатель Ресурсов). На практике это обычно эквивалентно ``то, что я вижу в адресной строке браузера''.

макетов, *разработки через тестирование (TDD)* и *интеграционных тестов*. Начнем с Главы 3, в которой создадим статические страницы, а затем добавим в них немного динамического контента. Мы немного познакомимся с языком Ruby, лежащим в основе Rails, в Главе 4. Затем, с Главы 5 по Главу 10, мы завершим базовую часть приложения, создав макет сайта, модель данных пользователя, а также системы регистрации и аутентификации (подтверждения подлинности) (включая активацию аккаунта и сброс пароля). Наконец, в Главе 11 и Главе 12 мы добавим микроблог и немного социальных функций, с тем, чтобы сделать рабочий пример сайта.

Блок 1.2. Scaffolding: Быстрее, проще, обольстительней

С самого начала Rails привлек к себе внимание знаменитым видеороликом о том, как создать микроблог за 15 минут, от David Heinemeier Hansson --- создателя Rails. Это видео и его потомки --- отличный способ познакомиться с мощью Rails, и я рекомендую посмотреть их. Но предупреждаю: они совершают свой удивительный пятнадцатиминутный подвиг, используя функцию под названием *scaffolding*[3], которая в значительной степени опирается на *сгенерированный код*, волшебным образом создаваемый Rails-командой `generate scaffold`.

При написании учебника по Ruby on Rails очень заманчиво положиться на scaffolding-подход --- это быстрее, проще, обольстительней. Но сложность и огромный объем кода, продуцируемый генераторами, могут стать непереодолимым препятствием для начинающего Rails-разработчика; вы, вероятно, сможете использовать его, но практически наверняка не сможете понять его. Увлекшись scaffolding-ом, вы рискуете превратиться в виртуозного генератора скриптов с пустяковым (и очень поверхностным) знанием Rails.

В *Ruby on Rails Tutorial* мы будем придерживаться (почти) полярно противоположного подхода: хотя в Главе 2 мы и разработаем мини-приложение с помощью сгенерированного кода, ядром *Rails Tutorial* является учебное приложения, которое мы начнем писать в Главе 3. На каждом этапе его разработки мы будем писать *небольшие, съедобные* куски кода --- достаточно простые для понимания, но все же требующие от вас некоторых усилий для их усвоения. Результатом станет более глубокое знание Rails, которое, в свою очередь, даст вам хорошую базу для написания практически любых типов веб-приложений.

1.1. Введение

``Ruby on Rails'' (или просто ``Rails'') --- это фреймворк для разработки веб-приложений, написанный на языке программирования Ruby. Со времен своего дебюта в 2004 году, Ruby on Rails довольно быстро стал одним из самых мощных и популярных инструментов для создания динамических веб-приложений. Rails используется множеством совершенно различных компаний: Airbnb, Basecamp, Disney, GitHub, Hulu, Kickstarter, Shopify, Twitter и Yellow Pages. Помимо этого, существует множество компаний по разработке ПО, специализирующихся на Rails, таких как ENTP, thoughtbot, Pivotal Labs, Hashrocket и HappyFunCorp, плюс бесчисленные независимые консультанты, преподаватели и контрагенты.

Что же делает Rails таким замечательным? Во-первых, Ruby on Rails --- это на 100% открытый исходный код, доступный под MIT Лицензией[4], и, как следствие, его загрузка и использование совершенно бесплатны. Rails также обязан своим успехом своему изящному и компактному дизайну; используя податливость лежащего в его основе языка предметно-ориентированный язык для написания веб-приложений. В результате множество часто встречающихся задач веб-программирования -- таких как генерация HTML, создание моделей данных и маршрутизация URL -- легки в Rails, а результирующий код приложений краток и читаем.

Rails также очень быстро адаптируется к новым тенденциям в веб-технологиях. Например, Rails был одним из первых, кто полностью реализовал архитектурный REST-стиль для структурирования веб-приложений (мы будем изучать этот стиль на протяжении учебника). И когда другие фреймворки успешно разрабатывают новые техники, создатель Rails, David Heinemeier Hansson[5], и рабочая группа Rails не стесняются использовать их идеи. Пожалуй, наиболее ярким и драматичным примером является слияние Rails и Merb (конкурирующий веб-фреймворк), благодаря которому Rails получил модульный дизайн Merb, стабильный API и улучшенную производительность.

Наконец, Rails выигрывает от необычайно увлечённого и разнообразного сообщества: сотни соразработчиков, весьма представительные конференции, огромное количество гемов (gem-пакетов) (автономные решения конкретных проблем, таких как постраничный вывод и загрузка изображений), богатый набор информативных блогов, и рог изобилия форумов и IRC-каналов. Большое количество Rails-программистов также облегчает работу с (неизбежными) ошибками, возникающими в процессе разработки: алгоритм ``Ищи в Google сообщение об ошибке" практически всегда выдает подходящую статью в блоге или относящееся к делу сообщение на форуме.

1.1.1. Необходимая квалификация

Для этой книги нет формального набора необходимых знаний --- *Ruby on Rails Tutorial* содержит интегрированные учебники не только по Rails, но также и по лежащему в его основе языку Ruby, предустановленному в Rails фреймворку для тестирования (minitest), командной строке Unix, HTML, CSS, немного JavaScript и даже чуть-чуть SQL. Это очень объемный материал и, в общем случае, желательно иметь небольшой опыт в HTML и программировании, прежде чем начать работу с этим учебником. Тем не менее, удивительное количество новичков использовали *Ruby on Rails Tutorial* для изучения веб-разработки с нуля, так что, даже если у вас мало опыта, я все же советую попробовать. Если содержимое учебника ошеломит вас, вы всегда можете сделать шаг назад и начать с одного из ресурсов, приведенных ниже. Другой возможной стратегией (рекомендованной многими читателями) является прохождение учебника дважды; возможно вы будете удивлены тому, как много вы узнали в первый раз (и насколько проще проходить учебник по второму кругу).

В настоящее время я работаю над серией учебных пособий, которые будут составлять полный набор книг, предваряющих *Rails Tutorial*, начиная с *Изучи™ командную строку достаточно, чтобы быть опас-*

[4]# Тут можно прочесть о ней на русском

[5]# Тут можно прочесть о нем на русском

ным --- введение в командную строку Unix для начинающих. Список последующих учебников можно найти на learnenough.com, там же можно зарегистрироваться и подписаться на уведомления об их готовности.

Стоит ли вначале изучить Ruby? Ответ зависит от вашего личного стиля обучения и опыта в программировании. Если вы предпочитаете изучать все систематически, с самых основ, или если вы не программировали ранее, то вам возможно стоит начать с изучения Ruby, и в этом случае я рекомендую *Учись программировать* Криса Пайна и *Начало Ruby* Peter Cooper. С другой стороны, множество начинающих Rails-разработчиков имеют своей целью создание *веб*-приложений, и, скорее всего, предпочтут не корпеть над толстенной книгой по Ruby для того, чтобы написать одну единственную веб-страницу. В этом случае я рекомендую поработать с коротким интерактивным учебником Try Ruby[6] для того, чтобы получить общее представление о Ruby, прежде чем погрузиться в *Rails Tutorial*. Если после этого учебник все же окажется для вас слишком сложным, вы можете попробовать начать с *Learn Ruby on Rails* Daniel Kehoe или One Month Rails --- и то, и другое более ориентировано на начинающих, нежели данный учебник.

Независимо от того, где вы начнете, по окончании этого учебника вы будете готовы к множеству довольно продвинутых ресурсов по Rails. Вот некоторые из тех, что я особенно рекомендую:

- Code School: Хорошие интерактивные онлайн-курсы по программированию.
- Turing School of Software & Design: очная, 27-недельная обучающая программа в Денвере, Колорадо, промокод на $500 скидку для читателей Rails Tutorial: RAILSTUTORIAL500.
- Bloc: образовательная онлайн-площадка со структурированным учебным планом, персональным наставничеством, обучение ведется на примере конкретных проектов. Используйте код купона BLOCLOVESHARTL, чтобы получить $500 скидку на вступительный взнос.
- Launch School: хорошая образовательная онлайн-площадка, посвященная Rails-разработке (включая продвинутый материал).
- Firehose Project: образовательная онлайн-площадка, сфокусированная на получении реальных навыков программирования (разработка через тестирование, алгоритмы, создание продвинутого веб-приложения в команде). Бесплатный двухнедельный вводный курс.
- Thinkful: Онлайн-занятия в партнерстве с профессиональным инженером, работа по учебному плану, основанному на конкретном проекте.
- Pragmatic Studio: Онлайн Ruby и Rails курсы от Mike и Nicole Clark. Вместе с Dave Thomas, автором *Programming Ruby*, Mike преподавал первый курс по Rails, который я прошел в 2006.
- RailsCasts от Ryan Bates: Отличные (в основном бесплатные) видеоуроки по Rails (хотя многие из них уже устарели).
- RailsApps: Большой выбор подробных, посвященных различным темам, Rails-проектов и учебников.

[6] http://tryruby.org/

- Rails Guides[7]: Актуальные руководства по Rails.

1.1.2. Соглашения в этой книге

Соглашения в этой книге главным образом очевидны. В этом разделе я упомяну лишь те, которые таковыми не являются.

В этой книге присутствует множество примеров использования командной строки. Для простоты, все они используют приглашение командной строки в Unix-стиле (знак доллара):

```
$ echo "hello, world"
hello, world
```

Как отмечено в Разделе 1.2, я рекомендую всем пользователям всех операционных систем (особенно пользователям Windows) использовать облачную среду разработки (Раздел 1.2.1), в которой есть встроенная Unix (Linux) командная строка. Это особенно полезно, поскольку в Rails есть множество команд, которые могут быть запущены из командной строки. Например, в Разделе 1.3.2 мы будем запускать локальный веб-сервер для разработки посредством команды **`rails server`**:

```
$ rails server
```

Как и в случае с приглашением командной строки, *Rails Tutorial* использует Unix-соглашение для разделителей директорий (то есть, косую черту, наклоненную вправо **`/`**). Например, конфигурационный файл **`production.rb`** учебного приложения будет представлен следующим образом:

```
config/environments/production.rb
```

Следует понимать, что это *относительный* путь к файлу, который строится относительно корневой папки приложения, абсолютный путь к которой зависит от системы; в облачной интегрированной среде разработки (IDE) (Раздел 1.2.1) этот путь выглядит следующим образом:

```
/home/ubuntu/workspace/sample_app/
```

Таким образом, полный путь к **`production.rb`**:

```
/home/ubuntu/workspace/sample_app/config/environments/production.rb
```

Для краткости, я обычно буду опускать путь к корневой папке приложения и буду писать просто **`config/environments/production.rb`**.

[7]# Тут можно найти в переводе

Rails Tutorial часто показывает вывод различных программ (команды оболочки, системы контроля версий, Ruby-программ и т.д.). Из-за неисчислимого количества небольших различий между компьютерными системами, вывод, который вы увидите, возможно, не всегда совпадет в точности с тем, который показан в тексте, но это не повод для беспокойства. Некоторые команды могут вызывать ошибки, связанные с особенностями вашей системы; вместо того, чтобы пытаться выполнить Сизифову задачу документирования всех таких погрешностей в этом учебном руководстве, я делегирую к алгоритму ``Ищи в Google сообщение об ошибке", который, между прочим, является весьма хорошей практикой для реального программирования. Если вы столкнетесь с проблемами в процессе обучения, я советую проконсультироваться на ресурсах, список которых приведен в разделе Помощь.[8]

Поскольку в *Rails Tutorial* (помимо всего прочего) рассматривается вопрос тестирования Rails-приложений, часто бывает полезным знать о том, что данный кусок кода приводит к провальному (обозначается красным цветом) или наоборот успешному результату тестов (обозначается зеленым цветом). Для удобства, код помечен соответственно **КРАСНЫЙ**и **ЗЕЛЕНЫЙ**.

К каждой главе учебника прилагаются упражнения, выполнение которых необязательно, но крайне рекомендуется. Для того, чтобы сделать основной текст независимым от упражнений, решения обычно не включены в последующие листинги кода. В тех редких случаях, когда решение упражнения используется в дальнейшем, оно будет приведено в основном тексте.

Наконец, для удобства *Ruby on Rails Tutorial* применяет два соглашения, призванные облегчить понимание большого количества примеров кода. Во-первых, в некоторых листингах выделены одна или несколько строк, как показано ниже:

```
class User < ActiveRecord::Base
  validates :name,  presence: true
  validates :email, presence: true
end
```

Такое выделение строк обычно указывает на наиболее важный новый код в данном примере, и часто (хотя и не всегда) отражает разницу между данным и предыдущим листингом. Во-вторых, для краткости и простоты, во многих листингах в книге встречаются вертикальные точки, например:

```
class User < ActiveRecord::Base
  .
  .
  .
  has_secure_password
end
```

Эти точки обозначают пропущенный код, и их не нужно копировать буквально.

[8] http://railstutorial.ru/4_0/help

1.2. За работу

Даже для опытных Rails-разработчиков установка Ruby, Rails и всего сопутствующего программного обеспечения поддержки может стать тем упражнением, что приводит в отчаяние. Проблема осложняется большим разнообразием окружений: различные операционные системы, номера версий, предпочтения в текстовых редакторах и интегрированных средах разработки (IDE), и т. д. Пользователи, у которых уже установлена среда разработки на локальной машине, могут использовать предпочитаемые настройки, а (как указано в Блоке 1.1) новым пользователям я предлагаю избежать всех этих проблем с установкой и настройкой за счёт использования *cloud integrated development environment (облачной интегрированной среды разработки)*. Облачная IDE запускается внутри обычного браузера, и, следовательно, одинаково хорошо работает на любой платформе, особенно это полезно для тех операционных систем (например, Windows), в которых исторически Rails-разработка была весьма сложной. Если же, несмотря на все предполагаемые трудности, вы по-прежнему хотите закончить *Ruby on Rails Tutorial* в локальной среде разработки, то я рекомендую инструкции на InstallRails.com.[9]

1.2.1. Среда разработки

Принимая во внимание множество своеобразных настроек и предпочтений, вероятно, существует столько вариантов сред разработки, сколько и Rails-программистов. Чтобы избежать этих сложностей, я стандартизировал *Ruby on Rails Tutorial* под превосходную облачную среду разработки Cloud9. В частности, мне было очень приятно сотрудничать с Cloud9 для того, чтобы предложить вам среду разработки, ориентированную на потребности именно этого, третьего издания учебника. Получившееся в результате Rails Tutorial Cloud9 workspace (рабочее пространство), поставляется предварительно настроенным, с большинством необходимого для профессиональной Rails-разработки программного обеспечения, в том числе Ruby, RubyGems, Git. (Отдельно мы будем устанавливать только одну большую часть ПО --- собственно Rails --- и это сделано намеренно (Раздел 1.2.2).) Облачная IDE также включает в себя три основных компонента, необходимых для разработки веб-приложений: текстовый редактор, навигатор файловой системы и терминал командной строки (Рисунок 1.1). Кроме всего прочего, ее текстовый редактор поддерживает глобальный поиск ``Найти в файлах", который я считаю необходимым в навигации по любым большим Ruby- или Rails-проектам.[10] И наконец, даже если вы решите в реальной жизни использовать не только облачную IDE (конечно же, я рекомендую изучение и других инструментов), в любом случае сейчас вы получите великолепное введение в изучение основных возможностей текстовых редакторов и других инструментов разработки.

Вот пошаговая инструкция для того, чтобы начать работу с облачной средой разработки:

1. Зарегистрируйтесь на бесплатном аккаунте в Cloud9[11]
2. Нажмите ``Go to your Dashboard" (``Перейти к панели управления")

[9] При этом спешу предупредить пользователей Windows о том, что установщик Rails, рекомендованный InstallRails, скорее всего устарел и будет несовместим с настоящим учебником.

[10] Например, чтобы найти определение функции `foo`, вы можете запустить глобальный поиск по ``def foo".

[11] https://c9.io/web/sign-up/free

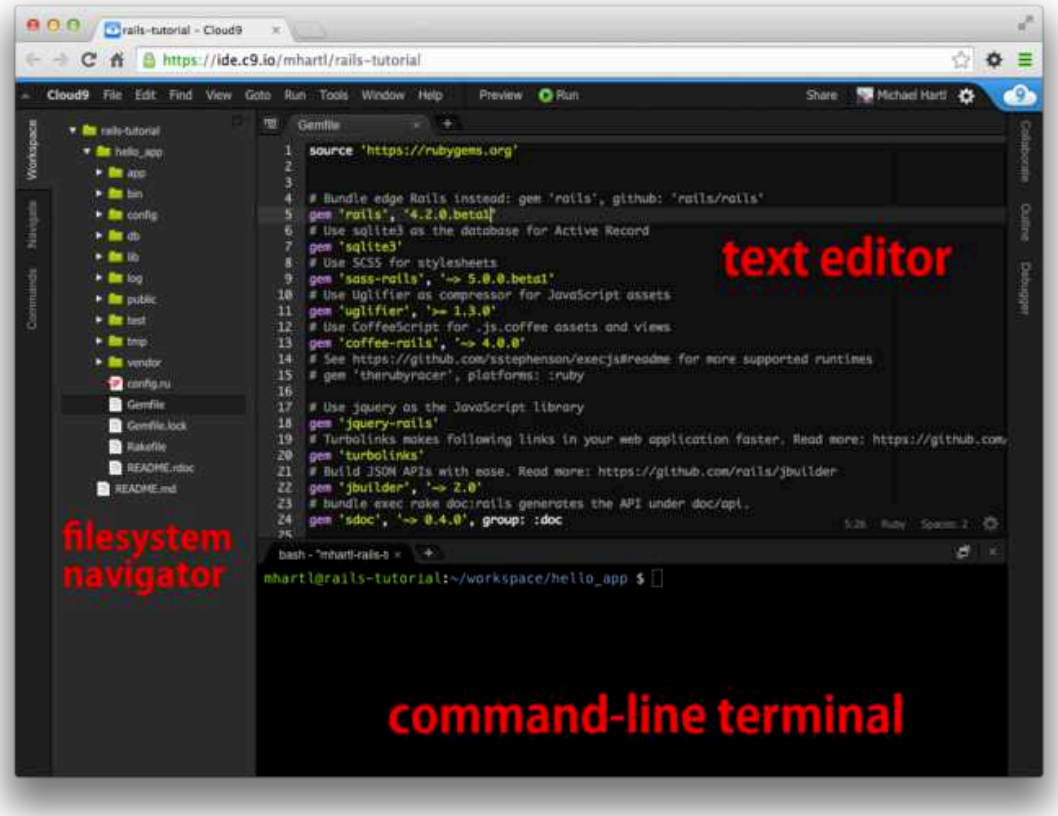


Рис. 1.1: Анатомия облачной IDE.

3. Выберите ``Create New Workspace'' (``Создать новое рабочее пространство'')
4. Как показано на Рисунке 1.2, создайте рабочее пространство ``rails-tutorial'' (*а не* ``rails_tutorial''), установите в настройках ``Private to the people I invite'' (``Только для приглашённых''), и выберите значок Rails Tutorial (*а не* значок Ruby on Rails)
5. Нажмите ``Create'' (``Создать'')
6. После того, как Cloud9 закончит резервировать рабочее пространство, выберите его и нажмите ``Start editing'' (``Начать редактирование'')

Так как использование двух пробелов для отступа --- это практически универсальное соглашение в Ruby, я также рекомендую изменить настройки редактора (по умолчанию используются четыре пробела). Как показано на Рисунке 1.3, нажмите значок шестерёнки в верхнем правом углу и выберите ``Code Editor (Ace)'', чтобы изменить настройку ``Soft Tabs''. (Обратите внимание, что настройки применяются мгновенно, нет необходимости нажимать кнопку ``Save'' (``Сохранить'').)

1.2.2. Установка Rails

Среда разработки из Раздела 1.2.1 включает в себя всё ПО, которо необходимо нам для старта, кроме самого Rails.[12] Чтобы установить Rails, мы будем использовать команду `gem`, которую обеспечивает менеджер пакетов *RubyGems*. Вам необходимо набрать команду из Листинга 1.1 в командной строке терминала. (Если вы работаете в локальной системе, то необходимо использовать обычное окно терминала; если же в облачной IDE, --- тогда область командной строки, как на Рисунке 1.1.)

Листинг 1.1: Установка Rails с определённым номером версии.

```
$ gem install rails -v 4.2.2
```

Метка `-v` гарантирует установку определённой версии Rails, и это очень важно для получения результатов, совместимых с этим учебником.

1.3. Первое приложение

Следуя давним традициям[13] в программировании, целью нашего первого приложения будет программа для написания фразы ``hello, world'' (``Привет, мир!''). В частности, мы создадим простое приложение, которое отобразит строку ``hello, world!'' на веб-странице, как в окружении разработки на вашем компьютере (Раздел 1.3.4), так и в интернете (Раздел 1.5).

[12] На данный момент в Cloud9 более старая версия Rails, и она несовместима с настоящим учебником, поэтому так важно установить его самостоятельно.

[13] # Тут можно прочесть об этом на русском

Рис. 1.2: Создание нового рабочего пространства в Cloud9.

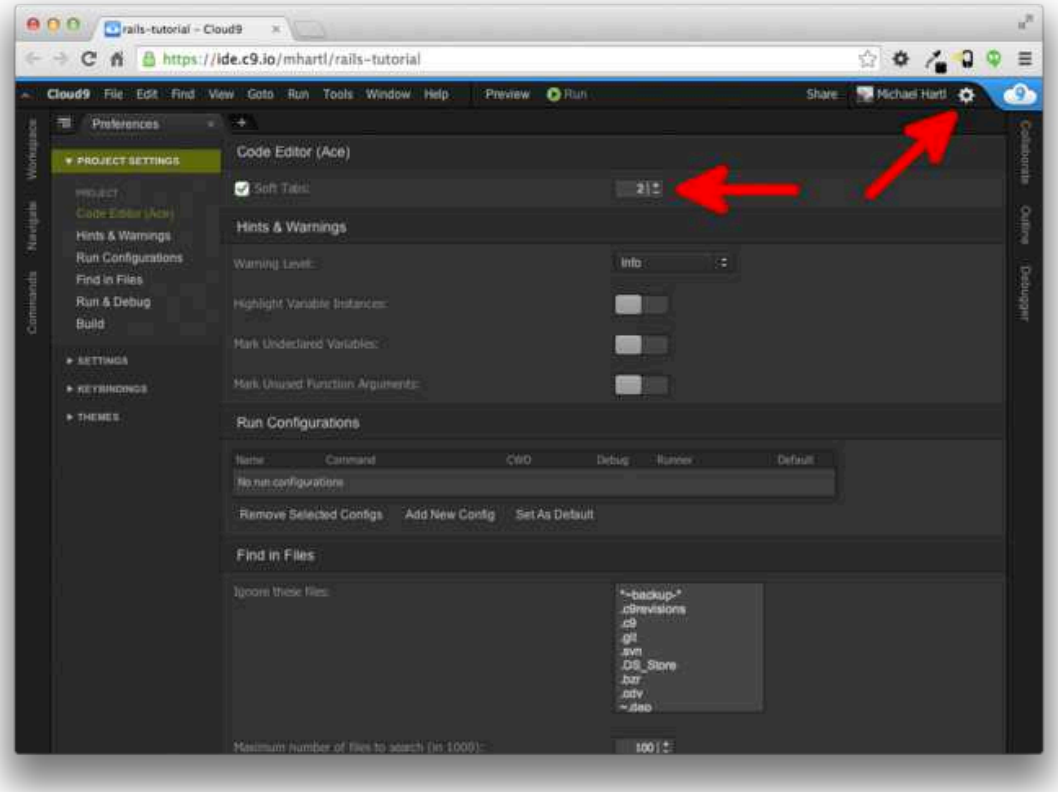


Рис. 1.3: Настройка использования двух пробелов для отступа в Cloud9.

Практически все приложения Rails начинаются одинаково --- с команды **`rails new`**. Эта удобная команда создает скелет приложения Rails в выбранном вами каталоге. Тем, кто *не* стал использовать Cloud9 IDE, рекомендованную в Разделе 1.2.1, для начала необходимо создать каталог **`workspace`** для Rails-проекта, если он ещё не существует (Листинг 1.2), и перейти в него. (В Листинге 1.2 показаны Unix-овые комманды **`cd`** и **`mkdir`**; вам поможет Блок 1.3, если вы с ними пока не очень знакомы.)

Листинг 1.2: Создание каталога **`workspace`** для Rails-проекта (не нужно в облаке).

```
$ cd                # Перейти в домашний каталог.
$ mkdir workspace   # Создать каталог workspace.
$ cd workspace/     # Перейти в него.
```

Блок 1.3. Краткий курс командной строки Unix

Для читателей, пришедших из Windows или (в меньшей, но всё ещё значительной степени) Macintosh OS X, командная строка Unix может быть непривычной. К счастью, если вы используете рекомендованную облачную среду разработки, вы автоматически получаете доступ к командной строке Unix (Linux) в стандартной оболочке интерфейса, известной как Bash.

Основная идея командной строки очень простая: выдавая короткие команды, пользователь может выполнять множество операций, таких как создание каталогов (`mkdir`), перемещение и копирование файлов (`mv` и `cp`), перемещение по файловой системе за счёт смены каталогов (`cd`). Хотя командная строка может показаться примитивной тем пользователям, кто в основном знаком с графическим интерфейсом, внешность обманчива: это один из наиболее мощных инструментов в наборе разработчика. Действительно, крайне редко удаётся увидеть рабочий стол опытного разработчика, на котором не было бы нескольких открытых окон терминала с оболочкой командной строки.

Вообще, тема очень глубокая, но для задач нашего учебника понадобится лишь немного наиболее часто встречающихся Unix-команд, они кратко показаны в Таблице 1.1. Для более глубокого изучения командной строки Unix, читайте *Conquering the Command Line* (Покоряя командную строку) Марка Бэйтса (доступна как бесплатная онлайн версия, так и электронная книга и видеоуроки).

Следующим шагом и в локальной, и в облачной системе, будет создание первого приложения с помощью команды из Листинга 1.3. Обратите внимание, что в нём явно обозначен номер версии Rails (**`_4.2.2_`**) в виде части команды. Это гарантирует, что для создания файловой структуры первого приложения, будет использована та же самая версия Rails, что была установлена в Листинге 1.1. (Если команда из этого листинга возвращает ошибку вроде ``Could not find 'railties'" (``Не могу найти'railties'"), значит установлена неверная версия Rails, и необходимо перепроверить, что команда из Листинга 1.1 была исполнена в точности так, как написано.)

Описание	Команда	Пример
list contents (список содержимого)	`ls`	`$ ls -l`
make directory (создать каталог)	`mkdir <имя>`	`$ mkdir workspace`
change directory (перейти в каталог)	`cd <имя>`	`$ cd workspace/`
перейти на один каталог вверх		`$ cd ..`
перейти в домашний каталог		`$ cd ~` или просто `$ cd`
перейти в каталог внутри домашнего		`$ cd ~/workspace/`
переместить (переименовать) файл	`mv <источник> <место назначения>`	`$ mv README.rdoc README.md`
скопировать файл	`cp <источник> <место назначения>`	`$ cp README.rdoc README.md`
удалить файл	`rm <файл>`	`$ rm README.rdoc`
удалить пустой каталог	`rmdir <каталог>`	`$ rmdir workspace/`
удалить непустой каталог	`rm -rf <каталог>`	`$ rm -rf tmp/`
объединить[14] & показать содержимое файла	`cat <файл>`	`$ cat ~/.ssh/id_rsa.pub`

Таблица 1.1: Некоторые часто используемые команды Unix.

Листинг 1.3: Запуск `rails new` (с указанием номера версии).

```
$ cd ~/workspace
$ rails _4.2.2_ new hello_app
      create
      create  README.rdoc
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      create  app
      create  app/assets/javascripts/application.js
      create  app/assets/stylesheets/application.css
      create  app/controllers/application_controller.rb
      .
      .
      .
      create  test/test_helper.rb
      create  tmp/cache
      create  tmp/cache/assets
      create  vendor/assets/javascripts
      create  vendor/assets/javascripts/.keep
      create  vendor/assets/stylesheets
      create  vendor/assets/stylesheets/.keep
         run  bundle install
Fetching gem metadata from https://rubygems.org/..........
Fetching additional metadata from https://rubygems.org/..
Resolving dependencies...
Using rake 10.3.2
Using i18n 0.6.11
.
.
.
Your bundle is complete!
Use `bundle show [gemname]` to see where a bundled gem is installed.
         run  bundle exec spring binstub --all
* bin/rake: spring inserted
* bin/rails: spring inserted
```

В конце Листинга 1.3 видно, что зауск `rails new` автоматически запускает команду `bundle install`,

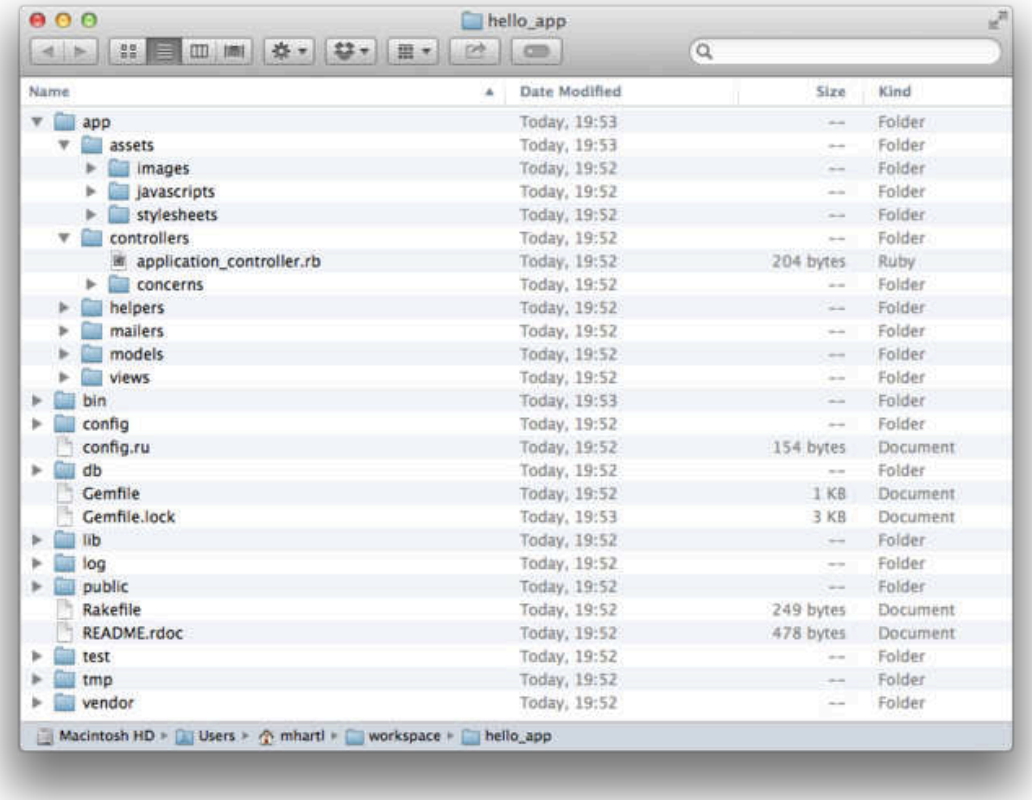


Рис. 1.4: Структура каталогов вновь созданного Rails-приложения.

после завершения создания файлов. Мы более детально обсудим, что это значит, в Разделе 1.3.1.

Отметьте, сколько файлов и каталогов создает команда `rails`. Эта стандартная структура файлов и каталогов (Рисунок 1.4) является одним из многих преимуществ Rails --- немедленный перенос от нуля к (минимально) действующему приложению. Кроме того, так как эта структура является общей для всех приложений Rails, можно сразу сориентироваться, глядя на чей-либо код. Обзор дефолтных Rails-файлов представлен в Таблице 1.2; мы узнаем о большинстве этих файлов и каталогов в остальной части книги. В частности, в Разделе 5.2.1 мы обсудим каталог `app/assets`, часть *asset pipeline (файлопровода)*, который значительно упрощает организацию и развёртывание активно используемых файлов (ассетов), таких как каскадные таблицы стилей и JavaScript-файлы.

Файл/Каталог	Назначение
`app/`	Основной код приложения (app), включает модели, представления, контроллеры и хелперы
`app/assets`	Ассеты приложения, такие как каскадные таблицы стилей (CSS), JavaScript-файлы и изображения
`bin/`	Бинарные исполняемые файлы
`config/`	Конфигурация приложения
`db/`	Файлы базы данных
`doc/`	Документация для приложения
`lib/`	Модули библиотеки
`lib/assets`	Библиотека ассетов приложения, таких как каскадные таблицы стилей (CSS), JavaScript-файлы и изображения
`log/`	Логи приложения
`public/`	Публично доступные данные (например, через браузер), такие как страницы ошибок
`bin/rails`	Программа для генерации кода, открытия консольных сессий, или запуска локального веб-сервера
`test/`	Тесты приложения
`tmp/`	Временные файлы
`vendor/`	Код сторонних разработчиков, такой как плагины и гемы
`vendor/assets`	Сторонние ассеты, такие как каскадные таблицы стилей (CSS), JavaScript-файлы и изображения
`README.rdoc`	Краткое описание приложения
`Rakefile`	Служебные задачи, доступные посредством команды `rake`
`Gemfile`	Гемы, необходимые приложению
`Gemfile.lock`	Список гемов, обеспечивающий использование всеми копиями приложения абсолютно одинаковых версий гемов
`config.ru`	Файл конфигурации для Rack middleware
`.gitignore`	Шаблоны файлов, которые должны игнорироваться Git

Таблица 1.2: Обзор начальной структуры Rails-каталогов.

1.3.1. Bundler

После создания нового Rails-приложения, следующий этап --- запуск *Bundler*, для установки и включения гемов, необходимых приложению. Как было вкратце отмечено в Разделе 1.3, Bundler автоматически запускается (через `bundle install`) командой `rails`, но в этом разделе мы немного изменим дефолтные гемы приложения и вновь запустим Bundler. Для этого нужно открыть `Gemfile` в текстовом редакторе. (В облачной IDE нажмите на стрелочку в навигаторе файлов для открытия каталога приложения и дважды кликните на значке `Gemfile`.) Хотя детали и цифры версий могут немного отличаться, результат должен выглядеть примерно как на Рисунке 1.5 и Листинге 1.4. (В этом файле используется Ruby, но не обращайте пока внимания на синтаксис; Глава 4 расскажет о Ruby более подробно.) Если файлы и каталоги не выглядят так, как на Рисунке 1.5, то нажмите на значок шестерёнки в навигаторе файлов и выберите ``Refresh File Tree'' (``Обновить дерево файлов''). (Как правило, необходимо обновлять файловое дерево каждый раз, когда файлы или каталоги не появляются в нужном месте.)

Листинг 1.4: Начальный `Gemfile` в каталоге `hello_app`.

```
source 'https://rubygems.org'


# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.2.2'
# Use sqlite3 as the database for Active Record
gem 'sqlite3'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
```

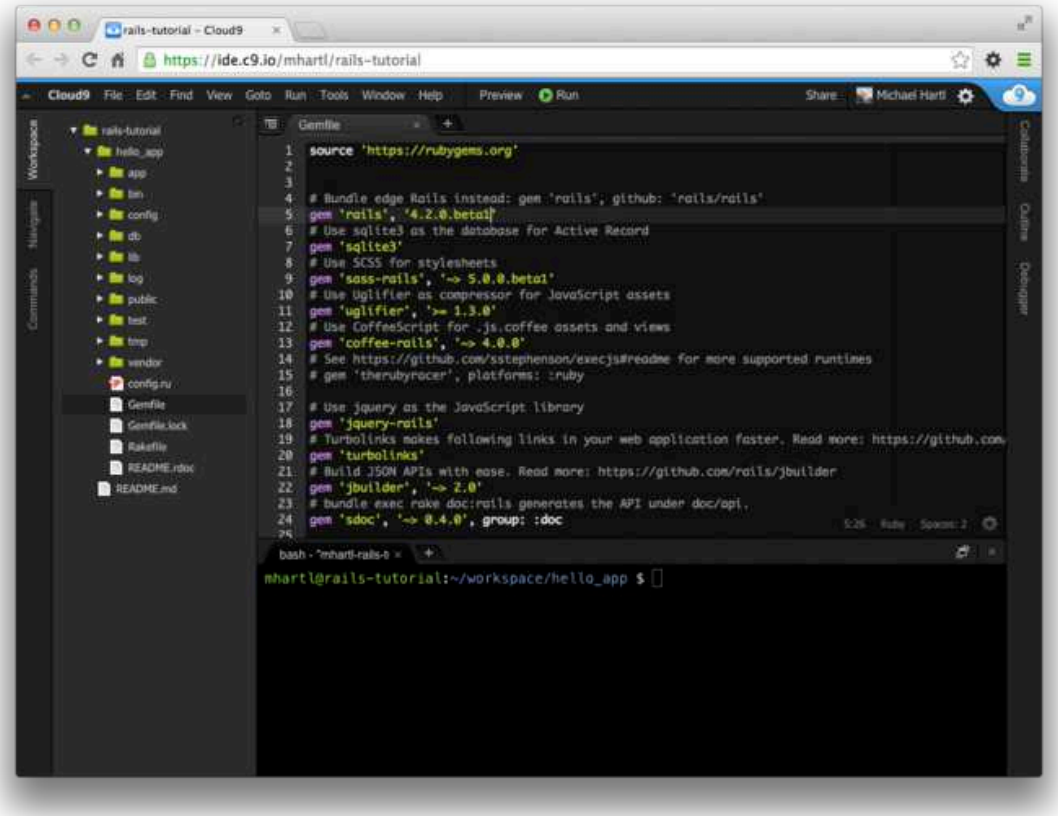


Рис. 1.5: Начальный `Gemfile` *в текстовом редакторе.*

```
gem 'uglifier', '>= 1.3.0'
# Use CoffeeScript for .js.coffee assets and views
gem 'coffee-rails', '~> 4.1.0'
# See https://github.com/sstephenson/execjs#readme for more supported runtimes
# gem 'therubyracer', platforms: :ruby

# Use jquery as the JavaScript library
gem 'jquery-rails'
# Turbolinks makes following links in your web application faster. Read more:
# https://github.com/rails/turbolinks
gem 'turbolinks'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.0'
# bundle exec rake doc:rails generates the API under doc/api.
gem 'sdoc', '~> 0.4.0', group: :doc

# Use ActiveModel has_secure_password
# gem 'bcrypt', '~> 3.1.7'

# Use Unicorn as the app server
# gem 'unicorn'

# Use Capistrano for deployment
# gem 'capistrano-rails', group: :development

group :development, :test do
  # Call 'debugger' anywhere in the code to stop execution and get a
  # debugger console
  gem 'byebug'

  # Access an IRB console on exceptions page and /console in development
  gem 'web-console', '~> 2.0'

  # Spring speeds up development by keeping your application running in the
  # background. Read more: https://github.com/rails/spring
  gem 'spring'
end
```

Большинство строк закомментированы хеш символом **#**; они здесь для того, чтобы показать некоторые часто используемые гемы, а также пример синтаксиса Bundler. Сейчас нам не понадобятся никакие гемы, кроме дефолтных.

Если вы не укажете номер версии в команде **gem**, то Bundler автоматически установит самую последнюю версию, например:

```
gem 'sqlite3'
```

Есть два основных способа указания диапазона версий гема для контроля того, что именно будет использовано Rails. Первый:

```
gem 'uglifier', '>= 1.3.0'
```

При этом устанавливается самая последняя версия гема **uglifier** (занимается сжатием файлов для файлопровода), при этом она обязана быть больше или равной версии **1.3.0** --- пусть даже это будет **7.2**.

Второй метод:

```
gem 'coffee-rails', '~> 4.0.0'
```

При таком способе записи установится гем **coffee-rails** не старше, чем версия **4.0.0**, и *не* новее **4.1**. Другими словами, запись >= всегда устанавливает самый последний гем, тогда как ~> 4.0.0 установит только незначительное (минорное) обновление (например, с **4.0.0** до **4.0.1**), но не позволит значительных (мажорных) обновлений (например, с **4.0** до **4.1**). К сожалению, практика показывает, что даже минорные релизы гемов могут что-нибудь сломать, так что для *Ruby on Rails Tutorial* мы будем ошибаться в пользу осторожности, явно прописывая конкретную версию для всех гемов. Вы можете использовать самую последнюю версию любого гема с помощью конструкции ~> в **Gemfile** (что я рекомендую в основном более продвинутым пользователям), но имейте в виду, что это может привести к совершенно непредсказуемому поведению кода учебника.

Прописав в **Gemfile** из Листинга 1.4 точные версии гемов, мы получим файл, показанный в Листинге 1.5. (Вы можете отпределить точный номер версии каждого гема, запустив в командной строке команду **gem list <название гема>**, но этот листинг избавляет вас от такой необходимости.) Кроме того, здесь мы указываем гему sqlite3 на то, что он должен работать только в окружеии разработки или тестов (Раздел 7.1.1), и это предотвращает потенциальные конфликты с базой данных, которую использует Heroku (Раздел 1.5). **Важно: Если вы читаете эту книгу не на railstutorial.org (а в бумажном издании, например), то вам необходимо использовать Gemfile, находящийся на gemfiles-3rd-ed.railstutorial.org, а не тот, что указан в книге.**

Листинг 1.5: **Gemfile** с явным указанием версии каждого гема Ruby.

```
source 'https://rubygems.org'

gem 'rails',                '4.2.2'
gem 'sass-rails',           '5.0.2'
gem 'uglifier',             '2.5.3'
gem 'coffee-rails',         '4.1.0'
gem 'jquery-rails',         '4.0.3'
gem 'turbolinks',           '2.3.0'
gem 'jbuilder',             '2.2.3'
gem 'sdoc',                 '0.4.0', group: :doc

group :development, :test do
  gem 'sqlite3',     '1.3.9'
  gem 'byebug',      '3.4.0'
  gem 'web-console', '2.0.0.beta3'
  gem 'spring',      '1.1.3'
end
```

После того, как вы поместили содержимое Листинга 1.5 в **Gemfile** приложения, установите гемы с помощью **bundle install**:[15]

[15] Как отмечено в Таблице 3.1, вы можете опускать **install**, т.к. команда **bundle** сама по себе является сокращением для **bundle install**.

```
$ cd hello_app/
$ bundle install
Fetching source index for https://rubygems.org/
.
.
.
```

Команда **`bundle install`** может занять некоторое время, но по ее завершении приложение будет готово к работе.

1.3.2. `rails server`

Благодаря запуску **`rails new`** в Разделе 1.3 и **`bundle install`** в Разделе 1.3.1, у нас уже есть приложение, которое мы можем запустить --- но как? К счастью, Rails поставляется с программой командной строки, или *скриптом*, который запускает *локальный* веб-сервер для того, чтобы помочь в разработке приложения. Точная команда зависит от того окружения, которое вы используете: на локальной машине просто **`rails server`** (Листинг 1.6), в то время как в Cloud9 необходимо дополнительно указать *IP binding address* и *номер порта*, тогда Rails-сервер будет знать адрес, который он может использовать, чтобы сделать приложение видимым для внешнего мира (Листинг 1.7).[16] (Cloud9 использует специальные *переменные окружения* **`$IP`** и **`$PORT`**, чтобы динамически назначать IP-адрес и номер порта. Если вам нужны значения этих переменных, наберите в командной строке **`echo $IP`** или **`echo $PORT`**.) Если ваша система жалуется на отсутствие JavaScript runtime, посетите execjs страницу на GitHub, чтобы ознакомиться со списком возможных решений этой проблемы. Я особенно рекомендую установку Node.js.

Листинг 1.6: Запуск Rails-сервера на локальной машине.

```
$ cd ~/workspace/hello_app/
$ rails server
=> Booting WEBrick
=> Rails application starting on http://localhost:3000
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
```

Листинг 1.7: Запуск Rails-сервера в cloud IDE.

```
$ cd ~/workspace/hello_app/
$ rails server -b $IP -p $PORT
=> Booting WEBrick
=> Rails application starting on http://0.0.0.0:8080
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
```

Какой бы вариант вы не выбрали, рекомендую запускать команду **`rails server`** во второй вкладке терминала, тогда в первой вкладке вы по-прежнему сможете выполнять команды, как показано на Рисунке 1.6

[16] Обычно, веб-сайты запускаются на 80-м порту, но чаще всего для этого требуются специальные права, поэтому стараются использовать менее ограниченные порты с бОльшим номером для сервера разработки.

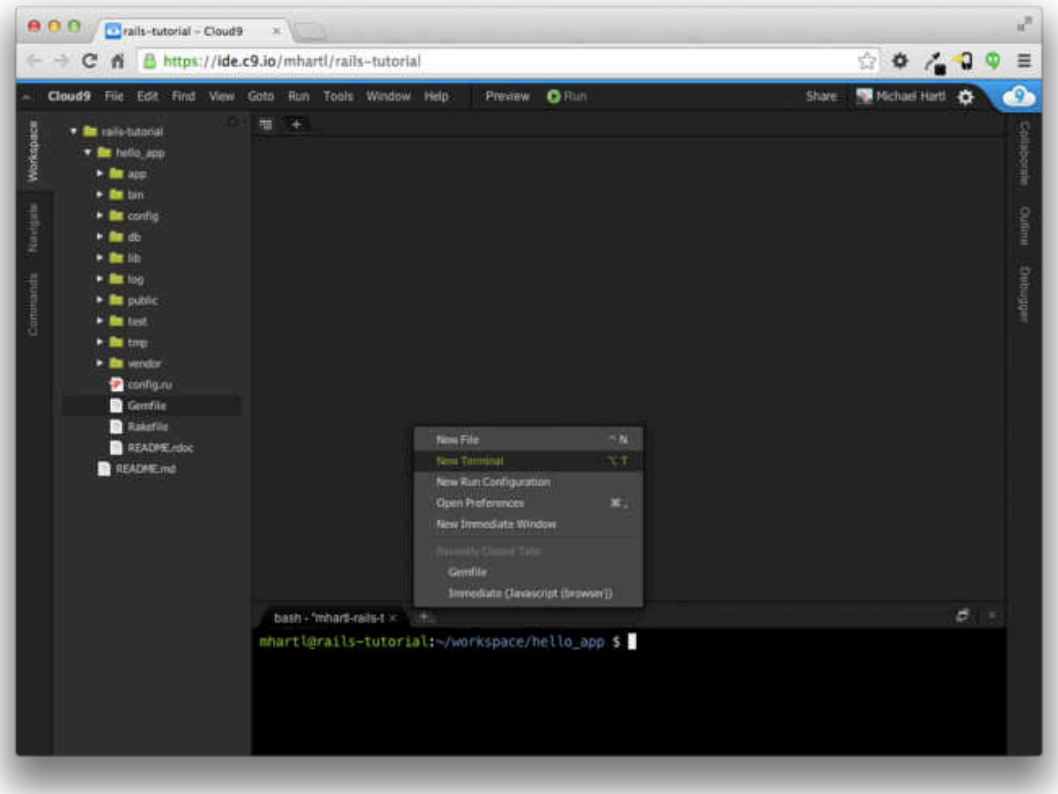


Рис. 1.6: Открытие новой вкладки терминала.

и Рисунке 1.7. (Если вы уже запустили сервер в первой вкладке, нажмите Ctrl-C, чтобы выключить его.)[17] Если вы используете локальный сервер, то наберите в браузере адрес http://localhost:3000/; в облачной IDE, перейдите к Share и нажмите на Application address (Рисунок 1.8). В любом случае, результат должен быть похож на то, что вы видите на Рисунке 1.9.

Чтобы увидеть информацию о нашем первом приложении, кликните ссылку ``About your application's environment'' (``Об окружении вашего приложения''). Хотя конкретные номера версий могут отличаться, в целом результат должен примерно как на Рисунке 1.10. Очевидно, что, в конечном итоге, начальная страница Rails нам не нужна, но сейчас приятно видеть ее работающей. Мы удалим её (и заменим созданной домашней страницей) в Разделе 1.3.4.

[17] ``C'' обозначает клавишу на клавиатуре, а не заглавную букву, поэтому не нужно зажимать ещё и Shift.

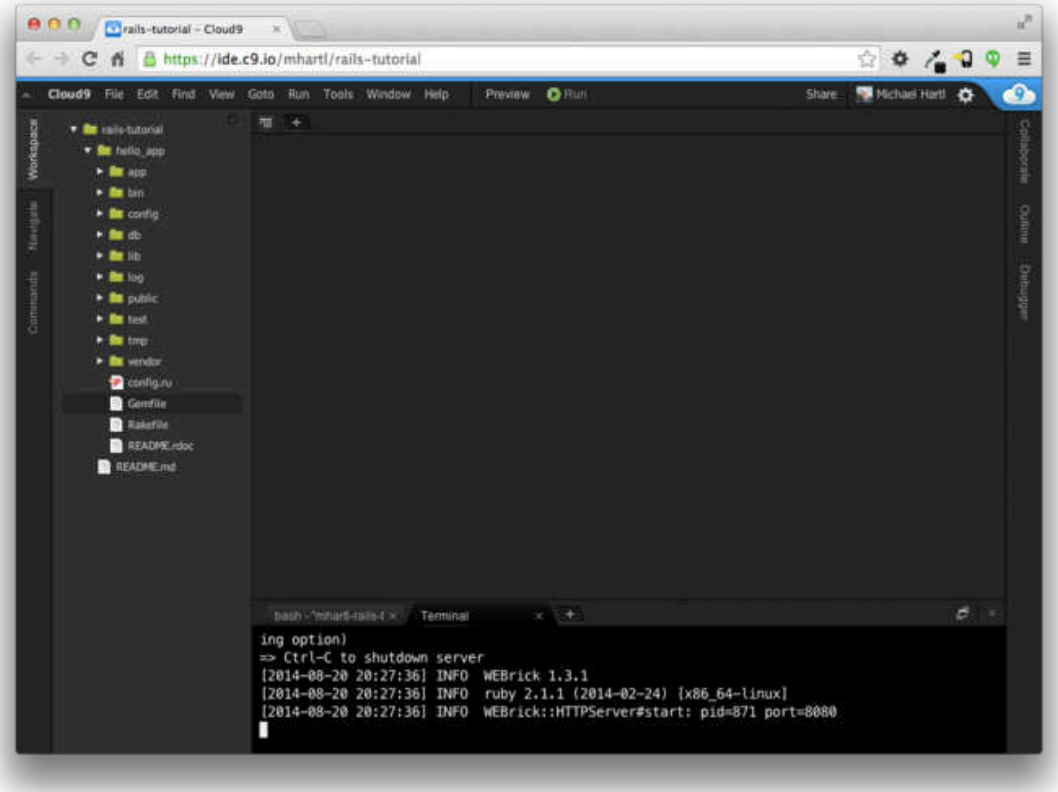


Рис. 1.7: Запуск Rails-сервера в отдельной вкладке.

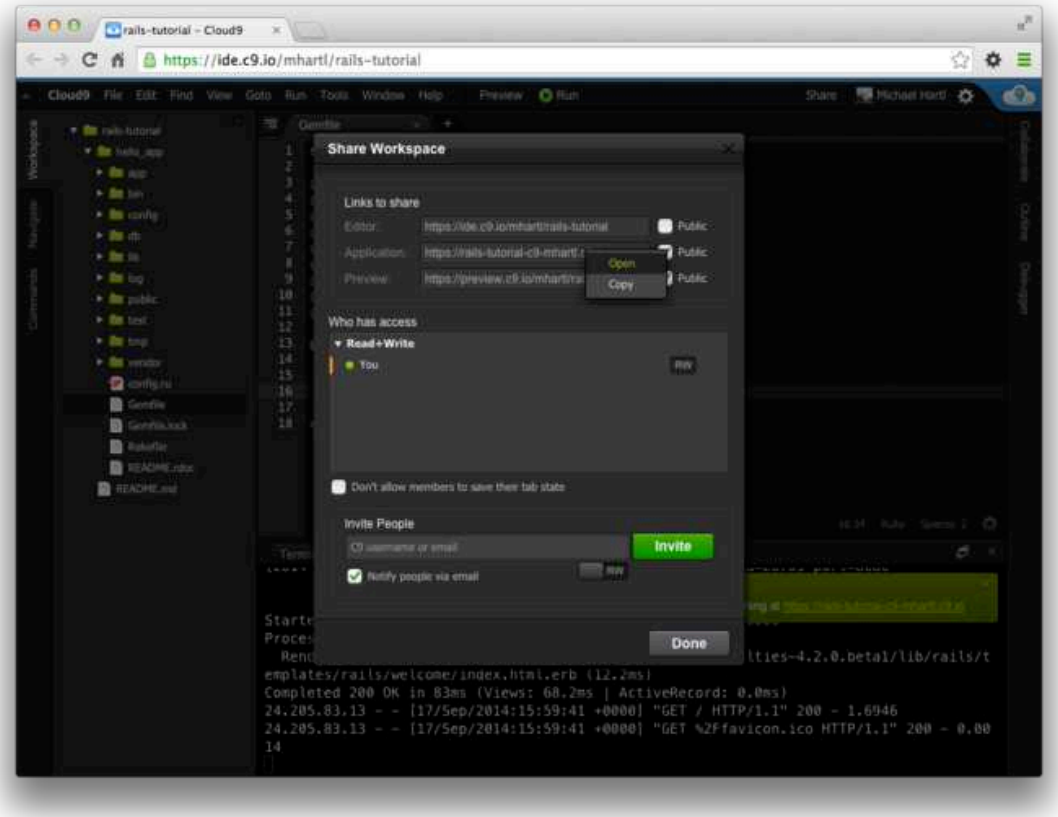


Рис. 1.8: Запуск локального сервера в облачном рабочем пространстве.

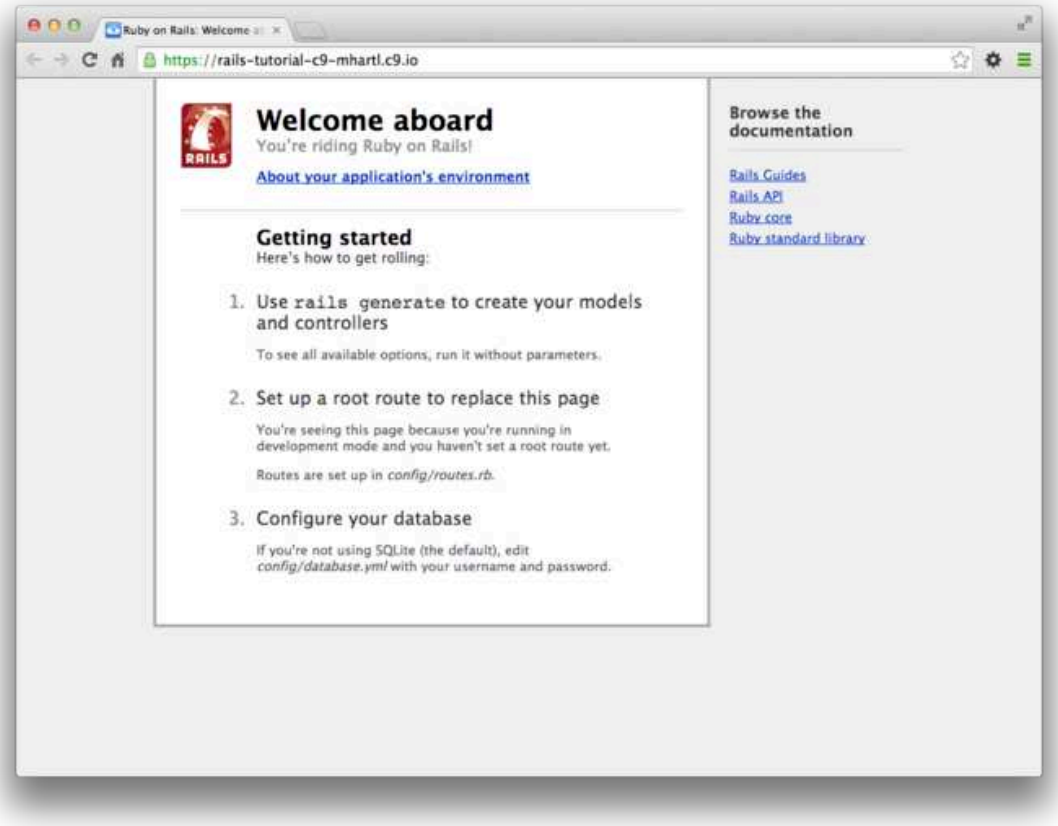


Рис. 1.9: Начальная Rails-страница, выданная `rails server`.

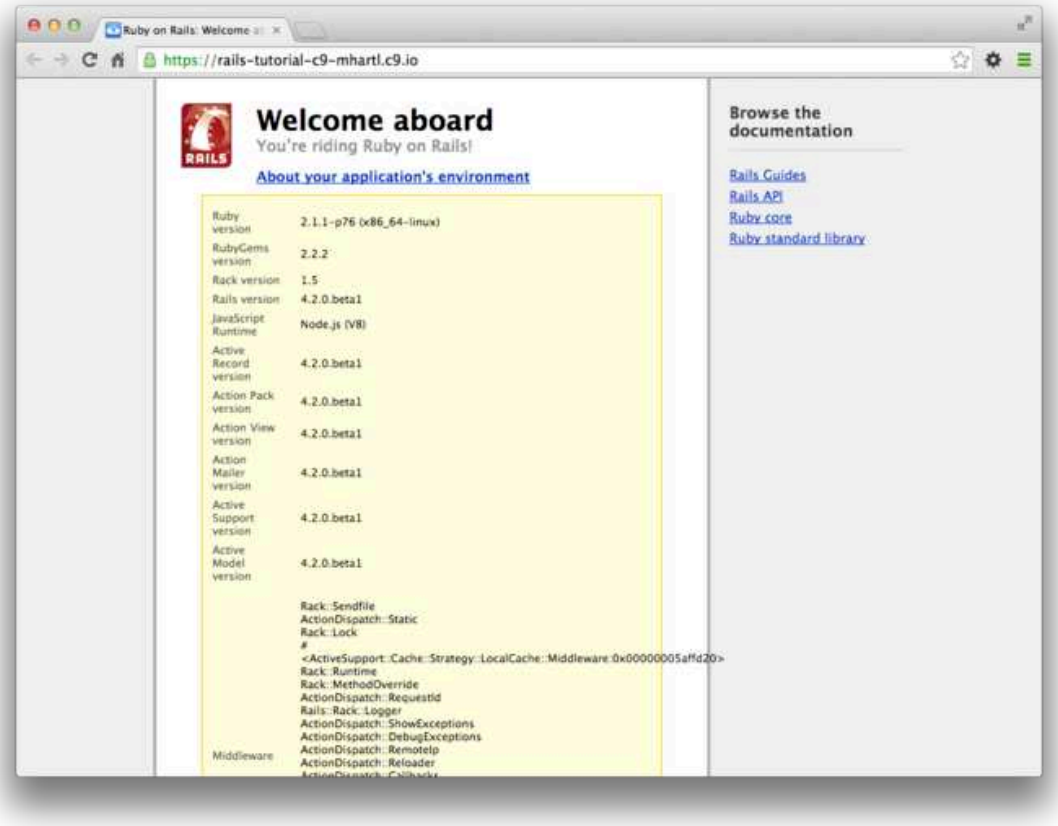


Рис. 1.10: Начальна страница с окружением приложения.

1.3.3. Модель-Представление-Контроллер (MVC)

Даже на этой ранней стадии, полезно получить общее представление о том, как работают Rails-приложения (Рисунок 1.11). Вы, возможно, заметили, что в стандартной структуре Rails-приложения (Рисунок 1.4) есть каталог под названием `app/` с тремя подкаталогами: `models`, `views`, и `controllers`. Это намек на то, что Rails следует архитектурной схеме модель-представление-контроллер (MVC), которая осуществляет отделение ``логики предметной области'' (или ``бизнес-логики'') от логики ввода и логики представления, связанной с графическим интерфейсом пользователя (GUI). В случае веб-приложений, ``логика предметной области'' обычно состоит из модели данных для таких вещей как пользователи, статьи, продукты, а GUI --- это просто веб-страница в браузере.

Взаимодействуя с приложением Rails, браузер отправляет *запрос*, веб-сервер принимает этот запрос, и передаёт его в Rails-*контроллер*, отвечающий за то, что делать дальше. В некоторых случаях контроллер сразу визуализирует *представление*, --- шаблон, которые преобразуется в HTML и отправлется назад в браузер. В динамических сайтах гораздо чаще контроллер взаимодействует с *моделью* --- объектом Ruby, который представляет собой элемент сайта (такой, как пользователь) и отвечает за связь с базой данных. После вызова модели контроллер визуализирует представление и возвращает полную веб-страницу браузеру в виде HTML.

Если это обсуждение сейчас кажется вам немного абстрактным, не беспокойтесь; мы будем часто возвращаться к этому разделу. В Разделе 1.3.4 показано первое пробное применение MVC, но уже в Разделе 2.2.2 мы будем гораздо более подробно обсуждать MVC в контексте мини-приложения. Наконец, учебное приложение будет использовать все аспекты MVC; мы осветим контроллеры и представления в Разделе 3.2, модели в Разделе 6.1, и увидим совместную работу всей этой троицы в Разделе 7.1.2.

1.3.4. Hello, world!

В качестве первого применения структуры MVC, мы внесём очень тонкие изменения в первое приложение, добавив *действие контроллера*, чтобы отобразить строку ``hello, world!''. (Мы узнаем больше о действиях контроллера в Разделе 2.2.2.) В результате мы заменим начальную Rails-страницу из Рисунка 1.9 страницей ``hello, world'' --- это и есть цель всего раздела.

Как и подразумевается в названии, действия контроллера определяются внутри контроллера. Мы назовем это действие `hello` и поместим в Application-контроллер (контроллер приложения). Действительно, сейчас это единственный контроллер, который есть в наличии, и в этом можно убедиться, если запустить

```
$ ls app/controllers/*_controller.rb
```

и посмотреть текущие контроллеры. (Мы начнём создавать собственные контроллеры в Главе 2.) В Листинге 1.8 показано определение `hello` с использованием функции `render` для возвращения текста ``hello, world!''. (Не беспокойтесь сейчас о Ruby-синтаксисе; разберёмся с ним в Главе 4.)

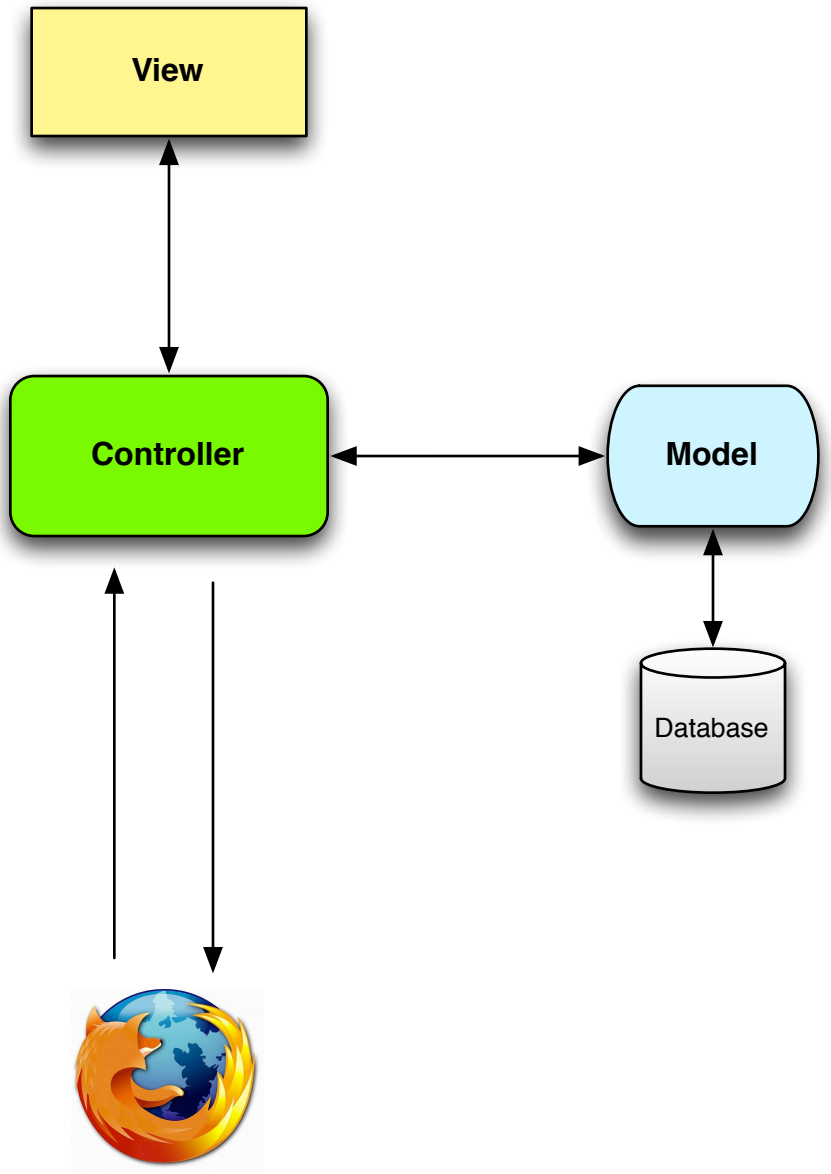


Рис. 1.11: Схематичное изображение архитектуры модель-представление-контроллер (MVC).

Листинг 1.8: Добавление действия `hello` в Application-контроллер.
app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception

  def hello
    render text: "hello, world!"
  end
end
```

Определив действие, которое возвращает нужную строку, необходимо сказать Rails, что именно оно должно быть использовано вместо начальной страницы на Рисунке 1.10. Для этого отредактируем *маршрутизатор* Rails, который находится перед контроллером на Рисунке 1.11, и определим, куда посылать запросы, приходящие от браузера. (Для простоты я опустил маршрутизатор на Рисунке 1.11, но мы обсудим его более детально в Разделе 2.2.2.) В частности, мы хотим изменить начальную страницу, *корневой маршрут*, именно он определяет --- какая страница будет показана при переходе по *корневому URL*. Так как это URL для адреса вроде http://www.example.com/ (когда больше ничего нет после слэша), корневой URL часто для краткости называют / (``слэш'').

В Листинге 1.9 видно, что файл Rails-маршрутов (`config/routes.rb`) содержит закомментированную строку, в которой говорится о том, как построить корневой маршрут. ``Welcome'' --- имя контроллера, ``index'' --- действие внутри этого контроллера. Чтобы активировать корневой маршрут, раскомментируйте эту строку, удалив хэш-символ, и замените кодом из Листинга 1.10, который говорит Rails направить корневой маршрут к действию `hello` Application-контроллера. (Как было отмечено в Разделе 1.1.2, вертикальные точки обозначают пропущенный код, их не нужно копировать буквально.)

Листинг 1.9: Начальный (закомментированный) корневой маршрут.
config/routes.rb

```
Rails.application.routes.draw do
  .
  .
  .
  # You can have the root of your site routed with "root"
  # root 'welcome#index'
  .
  .
  .
end
```

Листинг 1.10: Установка корневого маршрута.
config/routes.rb

```
Rails.application.routes.draw do
  .
```

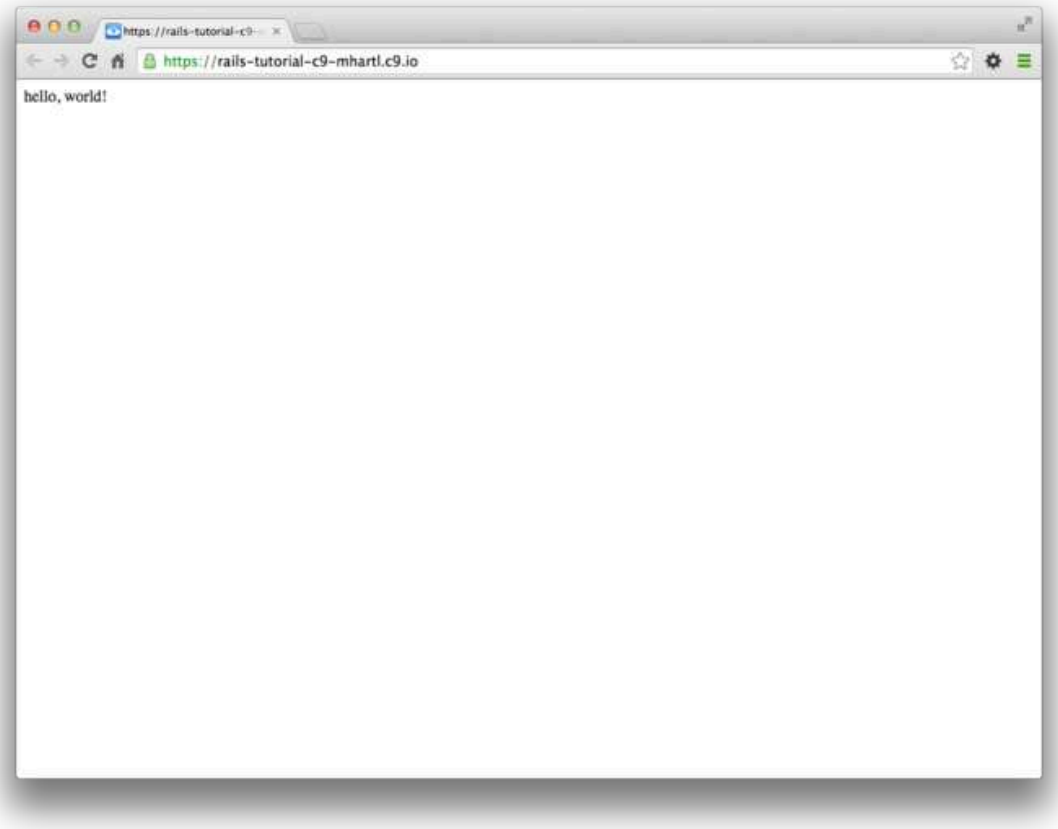


Рис. 1.12: Отображение ``hello, world!'' в браузере.

```
  .
  .
  # You can have the root of your site routed with "root"
  root 'application#hello'
  .
  .
  .
end
```

После этих изменений корневой маршрут возвращает ``hello, world!'' как и требовалось (Рисунок 1.12).

1.4. Управление версиями с Git

Теперь, когда у нас есть новое и рабочее приложение Rails, мы воспользуемся моментом и сделаем то, что (хотя технически необязательно) видится многим разработчиками Rails как практически необходимый шаг --- поместим исходный код нашего приложения в *систему управления версиями*. Системы управления

версиями позволяют отслеживать изменения кода проекта, облегчают совместную работу, а также могут откатывать любые непреднамеренные погрешности (такие, как случайное удаление файлов). Грамотное использование системы управления версиями --- необходимый навык для каждого профессионального разработчика программного обеспечения.

Есть много вариантов управления версиями, но сообщество Rails в значительной степени стандартизировано под Git, распределенную систему управления версиями, первоначально разработанную Линусом Торвальдсом (Linus Torvalds) для размещения ядра Linux. Git --- большая тема, и мы лишь слегка коснемся ее в этой книге, но есть много хороших бесплатных онлайн ресурсов; я особенно рекомендую Начинаем работу с Bitbucket для краткого обзора и *Pro Git* автора Scott Chacon для более подробного изучения. Помещение вашего исходного кода в систему управления версиями Git *строго* рекомендуется, не только потому, что это почти универсальная практика в мире Rails, но также и потому, что это позволит вам легко создать резервную копию или открыть доступ к вашему коду (Раздел 1.4.3), а также развернуть ваше приложение прямо здесь, в первой главе (Раздел 1.5).

1.4.1. Установка и настройка

Облачная IDE, рекомендованная в Разделе 1.2.1, по умолчанию уже содержит Git, поэтому в этом случае нет необходимости в установке. В противном случае, InstallRails.com (Раздел 1.2) содержит инструкции по установке Git в систему.

Первоначальная настройка системы

Прежде, чем использовать Git, следует выполнить ряд разовых настроек. Это *системные* настройки, а значит, их необходимо сделать лишь единожды:

```
$ git config --global user.name "Ваше имя"
$ git config --global user.email ваша@поч.та
$ git config --global push.default matching
$ git config --global alias.co checkout
```

Имя и адрес электронной почты, которые вы используете при настройке Git, будут доступны в любом вашем открытом репозитории. (Строго необходимы только первые две строки. Третья строка обеспечивает совместимость с будущими релизами Git. Четвертая строка позволяет использовать `co` вместо более длинной команды `checkout`. Для максимальной совместимости с системами, в которых не настроено `co`, в учебнике будет использоваться полная команда `checkout`, но в жизни я почти всегда пишу `git co`.)

Первоначальная настройка репозитория

Сейчас мы разберём этапы, необходимые при создании каждого нового *репозитория* (иногда его называют *репо* для краткости). Сначала перейдите в корневой каталог первого приложения, и инициализируйте новый репозиторий:

```
$ git init
Initialized empty Git repository in /home/ubuntu/workspace/hello_app/.git/
```

Теперь добавьте все файлы проекта в репозиторий с помощью **`git add -A`**:

```
$ git add -A
```

Эта команда добавит все файлы из текущего каталога, кроме тех, которые соответствуют шаблонам (правилам) в специальном файле **`.gitignore`**. Команда **`rails new`** автоматически создает файл **`.gitignore`**, соответствующий Rails-проекту, но в него так же можно добавить и дополнительные правила.[18]

Добавленные файлы сначала помещаются в *зону ожидания*, в ней находятся незавершённые изменения в проекте. Можно посмотреть, какие файлы сейчас находятся там, с помощью команды **`status`**:

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

  new file:   .gitignore
  new file:   Gemfile
  new file:   Gemfile.lock
  new file:   README.rdoc
  new file:   Rakefile
  .
  .
  .
```

(Результат очень длинный, поэтому здесь вертикальные точки вместо бОльшей части текста.)

Для того, чтобы сообщить Git о необходимости сохранения изменений, используйте команду **`commit`**:

```
$ git commit -m "Initialize repository"
[master (root-commit) df0a62f] Initialize repository
.
.
.
```

Метка **`-m`** позволяет вам добавлять сообщение для фиксации (коммита); если вы пропустите **`-m`**, то Git откроет редактор, установленный в системе по умолчанию, и предложит в нём ввести сообщение. (Во всех примерах в книге будет использован флаг **`-m`**.)

Важно отметить, что коммиты Git *локальны*, они записываются только на машине, на которой сделаны. Далее мы разберём, как отправить изменения в удалённый репозиторий (с помощью **`git push`**) в Разделе 1.4.4.

[18] Хотя нам и не понадобится редактировать его в этом учебнике, пример добавления правила в файл **`.gitignore`** будет показан в Разделе 3.7.3, как часть дополнительных продвинутых настроек тестирования в Разделе 3.7.

Кстати, список коммитов можно вызвать командой **log**:

```
$ git log
commit df0a62f3f091e53ffa799309b3e32c27b0b38eb4
Author: Michael Hartl <michael@michaelhartl.com>
Date:   Wed August 20 19:44:43 2014 +0000

    Initialize repository
```

В зависимости от длины логов репозитория, может понадобиться нажать **q** для выхода.

1.4.2. Что хорошего Git делает для вас?

Если раньше вы никогда не использовали систему контроля версий, то вам может быть не очень понятно сейчас, зачем это нужно, поэтому позвольте мне привести всего один пример. Предположим, вы произвели некоторые случайные изменения, например (О нет!) удалили крайне необходимый каталог **app/controllers/**.

```
$ ls app/controllers/
application_controller.rb  concerns/
$ rm -rf app/controllers/
$ ls app/controllers/
ls: app/controllers/: No such file or directory
```

Здесь использована Unix-команда **ls** для отображения списка содержимого каталога **app/controllers/**, и команда **rm** для его удаления (Таблица 1.1). Метка **-rf** означает ``recursive force" (``рекурсивно форсированно"), и рекурсивно удаляет все файлы, каталоги, подкаталоги, и так далее, не запрашивая явного подтверждения для каждого удаления.

Давайте проверим статус и посмотрим, что изменилось:

```
$ git status
On branch master
Changed but not updated:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

      deleted:    app/controllers/application_controller.rb

no changes added to commit (use "git add" and/or "git commit -a")
```

Мы видим , что файл был удален, но изменения находятся только на ``рабочем дереве"; они еще не зафиксированы. Это означает, что мы все еще можем легко отменить изменения командой **checkout** с флагом **-f** для форсированной перезаписи текущих изменений:

```
$ git checkout -f
$ git status
# On branch master
```

```
nothing to commit (working directory clean)
$ ls app/controllers/
application_controller.rb  concerns/
```

Пропавшие каталоги и файл вернулись. Какое облегчение!

1.4.3. Bitbucket

Теперь, когда ваш проект помещен в систему управления версиями Git, пришло время отправить код на Bitbucket[19] --- сайт, оптимизированный для хостинга и совместного использования Git-репозиториев. (В предыдущем издании учебника вместо него был использован GitHub[20]; читайте в Блоке 1.4 о причинах замены.) Отправка копии Git-репозитория на Bitbucket служит двум целям: полное резервное копирование вашего кода (включая полную историю коммитов), и более легкое любое будущее сотрудничество в разработке.

Блок 1.4. GitHub и Bitbucket

Наиболее популярные сайты для хостинга Git-репозиториев --- GitHub и Bitbucket. Они во многом похожи: позволяют хостинг и совместное использование кода, а также предоставляют удобные способы для просмотра и поиска по репозиторию. Самое главное отличие (с точки зрения этого учебника) в том, что GitHub бесплатно предлагает неограниченное количество репозиториев с открытым исходным кодом (с возможностью совместной работы), но берёт плату за закрытые (приватные) репозитории, в то время как Bitbucket позволяет бесплатно завести неограниченное количество приватных репозиториев, но при этом ограничивает число совместно работающих людей (увеличить его можно за плату). Таким образом, выбор сервиса для конкретного репозитория зависит от ваших потребностей.

В предыдущем издании этого учебника был использован GitHub из-за его акцента на поддержку открытого исходного кода, но всё возрастающая обеспокоенность по поводу безопасности привела меня к тому, что я рекомендую *все* репозитории веб-приложений делать приватными по умолчанию. Вопрос в том, что такие репозитории могут содержать конфиденциальную информацию, такую как криптографические ключи или пароли, которая может быть использована для компрометирования безопасности сайтов, исполняющих этот код. Конечно, вполне возможно организовать безопасную обработку информации (путем настроек игнорирования Git, например), но эти методы предрасположены к ошибкам, да и требуют значительно опыта.

На самом деле, учебное приложение из этого учебника вполне безопасно для того, чтобы выложить его в сеть, но не стоит всегда полагаться на этот факт. Таким образом, чтобы быть максимально безопасными, мы будем ошибаться в сторону осторожности и по умолчанию использовать приватные репозитории. Поскольку GitHub берет плату за приватность, а Bitbucket предлагает безлимитное количество приватных репозиториев бесплатно, то для наших целей Bitbucket подходит лучше, чем GitHub.

[19]# Тут можно прочесть о нем на русском

[20]# Тут можно прочесть о нем на русском

Начать работу с Bitbucket очень просто:

1. Создайте аккаунт на Bitbucket, если его у вас ещё нет.
2. Скопируйте ваш *открытый ключ* в буфер обмена. Как показано в Листинге 1.11, пользователи облачной IDE могут увидеть его посредством команды **cat**, после чего его можно выбрать и скопировать. Если вы используете другую систему, и запуск команды из Листинга 1.11 не даёт никакого результата, тогда следуйте инструкции о том, как установить открытый ключ в аккаунт Bitbucket[21].
3. Добавьте ваш открытый ключ на Bitbucket, нажав на аватар в правом верхнем углу и выбрав ``Manage account'' (``Управление аккаунтом''), а затем ``SSH keys'' (``SSH-ключи'') (Рисунок 1.13).

Листинг 1.11: Вывод открытого ключа с помощью **cat**.

```
$ cat ~/.ssh/id_rsa.pub
```

После того, как вы добавили ключ, нажмите ``Create'' (``Создать''), чтобы создать новый репозиторий, как показано на Рисунке 1.14. Заполнив информацию о проекте, не забудьте отметить галочкой пункт ``This is a private repository.'' (``Это приватный репозиторий''). После нажатия на ``Create repository'', следуйте инструкциям под ``Command line > I have an existing project'' (``У меня уже есть проект''), они должны выглядеть примерно как Листинг 1.12. (Если они выглядят по-другому, то скорее всего открытый ключ не был корректно добавлен, тогда стоит попробовать добавить его ещё раз.) При отправке репозитория ответьте yes, если видите вопрос ``Are you sure you want to continue connecting (yes/no)?'' (``Вы уверены, что хотите продолжить подключение'')

Листинг 1.12: Добавление Bitbucket и отправка репозитория.

```
$ git remote add origin git@bitbucket.org:<username>/hello_app.git
$ git push -u origin --all # отправляет репозиторий в первый раз
```

Команды в Листинге 1.12 указывают Git на то, что вы хотите добавить Bitbucket в качестве *origin (источника)* для вашего репозитория, а затем собственно отправить репозиторий в удалённый источник. (Не беспокойтесь о значении флага **-u**; если вам любопытно, поищите в сети ``git set upstream''.) Конечно, следует заменить `<username>` вашим фактическим именем пользователя. Например, команда которую запустил я:

```
$ git remote add origin git@bitbucket.org:mhartl/hello_app.git
```

Результатом является страница на Bitbucket для репозитория hello_app, с браузером файлов, полной историей коммитов, и большим количеством прочих приятных мелочей (Рисунок 1.15).

[21] # Тут есть инструкция на русском, может быть полезна

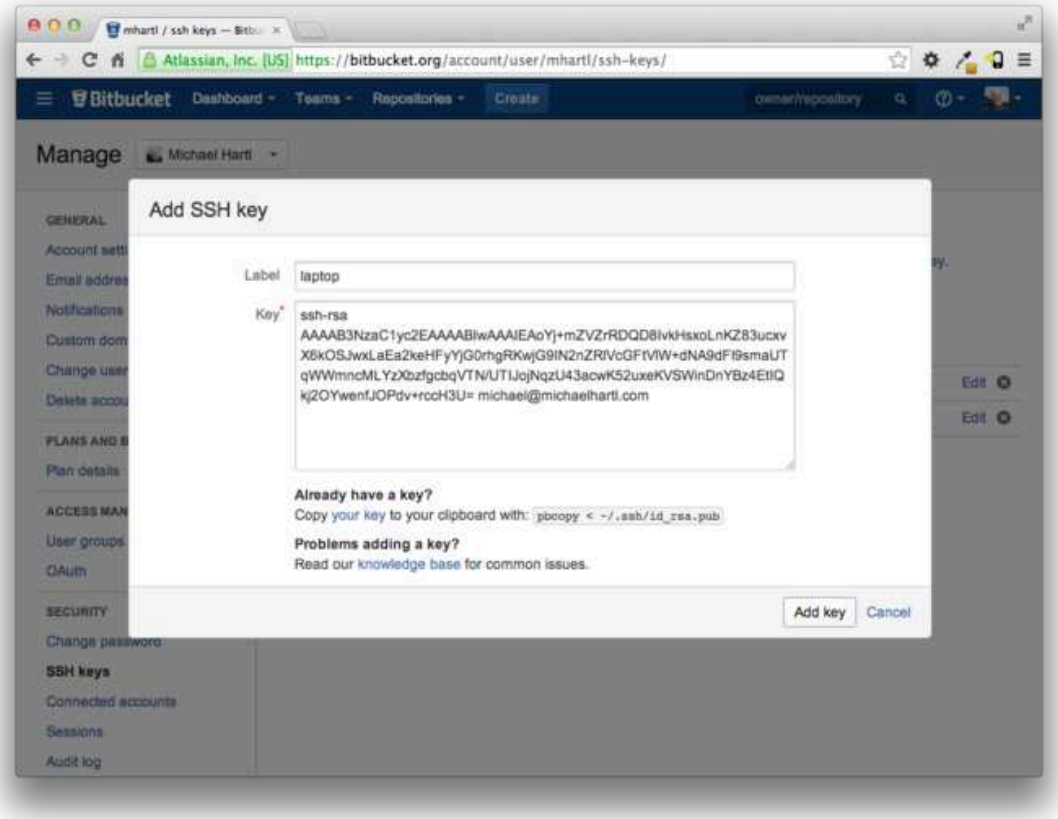


Рис. 1.13: Добавление открытого SSH-ключа.

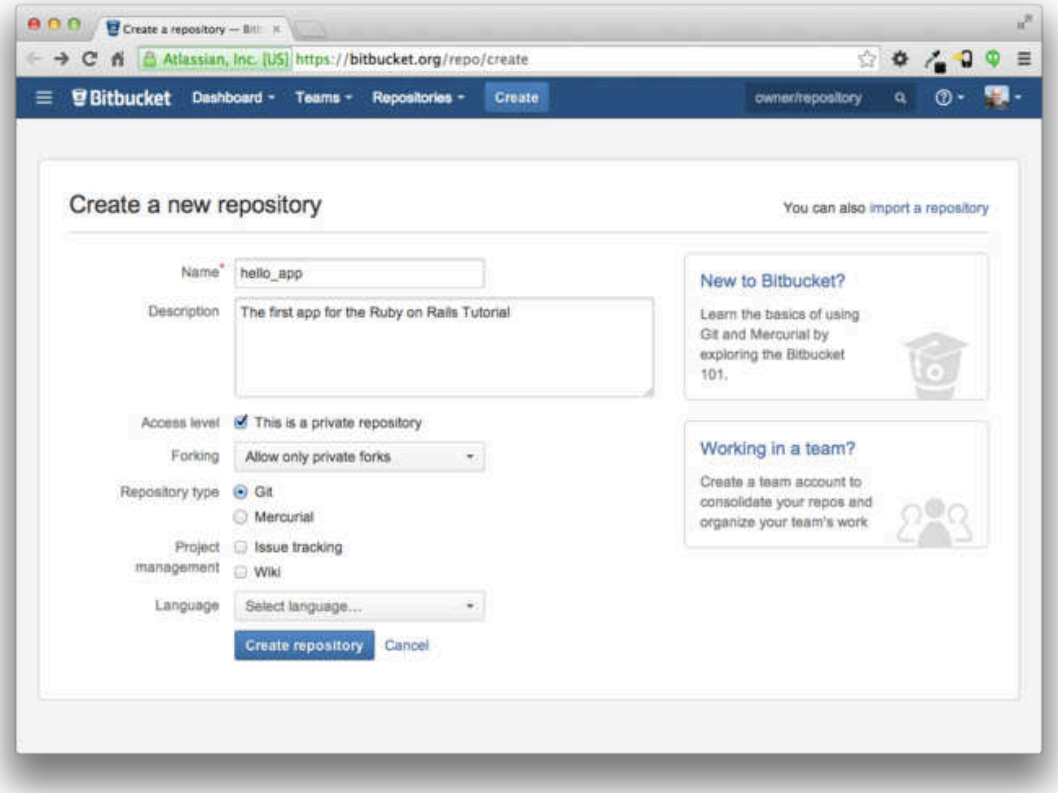


Рис. 1.14: Создание первого репозитория на Bitbucket.

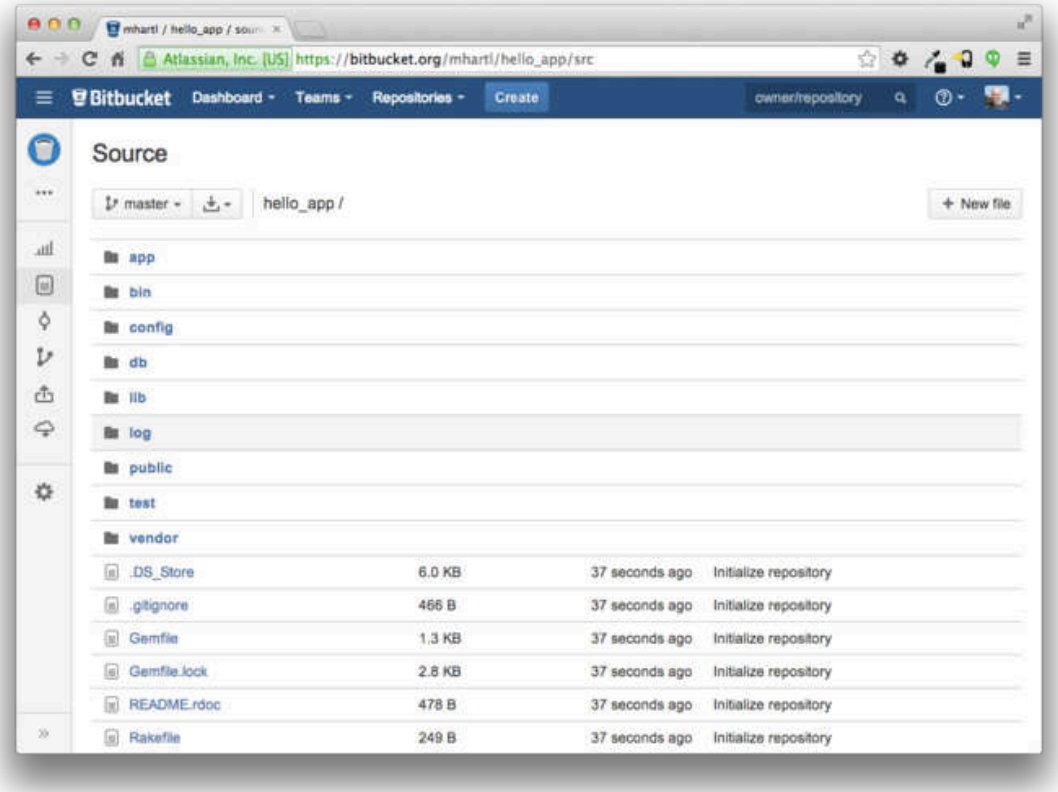


Рис. 1.15: Страница Bitbucket-репозитория.

1.4.4. Ветвление, редактирование, фиксация, слияние

Если вы выполнили все действия в Разделе 1.4.3, то могли заметить, что Bitbucket не определяет автоматически файл `README.rdoc` в нашем репозитории, вместо этого на главной странице он жалуется на его отсутствие (Рисунок 1.16). Это указывает на недостаточно широкую распространённость формата `rdoc`, чтобы Bitbucket поддерживал его автоматически, и на самом деле, я и практически все известные мне разработчики предпочитают вместо него использовать *Markdown*. В этом разделе мы заменим файл `README.rdoc` на `README.md`, а также добавим немного Rails Tutorial-специфичного содержимого в него. В процессе мы увидим первый пример ветвления, редактирования, фиксации и слияния --- именно такой рабочий процесс я рекомендую использовать в работе с Git.[22]

Ветвление

Git невероятно хорош в создании *веток*, которые фактически являются копиями репозитория, где мы можем произвести (возможно экспериментальные) изменения, не модифицируя родительские файлы. В большинстве случаев родительский репозиторий --- это *master* ветка, и мы можем создать новую рабочую ветку, используя `checkout` с флагом `-b`:

```
$ git checkout -b modify-README
Switched to a new branch 'modify-README'
$ git branch
  master
* modify-README
```

Вторая команда, `git branch`, только перечисляет все локальные ветки, а звездочкой `*` отмечена текущая ветка. Обратите внимание, `git checkout -b modify-README` одновременно создает новую ветку и переключает на нее, на что указывает звездочка перед `modify-README`. (Если вы настроили `co` в Разделе 1.4, то вместо этого можете писать `git co -b modify-README`.)

Полностью оценить достоинства ветвления можно, только работая над проектом совместно с множеством разработчиков,[23] но ветки полезны даже для единственного разработчика, как в нашем случае. В частности, ветка master изолируется от любых изменений, которые мы производим в рабочих ветках, так что даже если мы *действительно* наворотили лишнего, можно всегда отказаться от изменений, вернувшись в master и удалив ту ветку, в которой работали до этого. Мы увидим, как это делается, в конце раздела.

Между прочим, для столь малых изменений обычно я бы не стал озадачиваться новой веткой, но никогда не бывает слишком рано, чтобы начать практиковать хорошие привычки.

[22] Чтобы вам было удобнее представить себе Git-репозитории, взгляните на Приложение SourceTree от Atlassian; Тут можно прочесть о нем на русском.

[23] Более подробно в главе Git Branching в *Pro Git*.

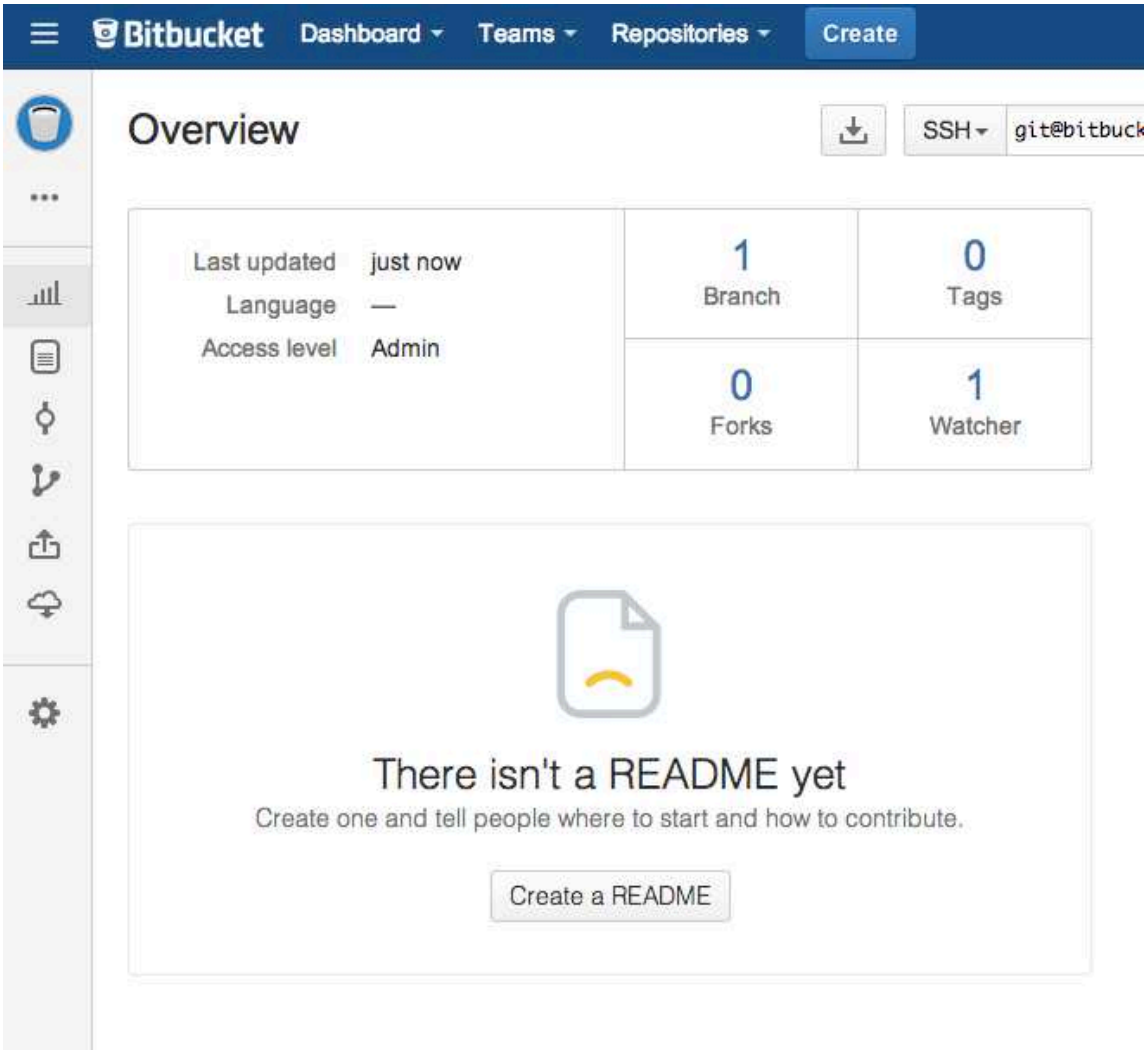


Рис. 1.16: Сообщение Bitbucket об отсутсвии README.

Редактирование

После создания локальной ветки мы отредактируем файл readme, чтобы сделать его немного более наглядным. Я предпочитаю использовать язык разметки Markdown[24] для этих целей, и если вы используете расширение файла `.md`, то Bitbucket будет автоматически форматировать его в приятный для вас вид. Итак, сначала запустим Git-версию Unix-команды `mv` (move) чтобы изменить название:

```
$ git mv README.rdoc README.md
```

Затем заполним `README.md` содержимым Листинга 1.13.

Листинг 1.13: Новый файл `README`, `README.md`.

```
# Ruby on Rails Tutorial: "hello, world!"

Это первое приложение для
[*Ruby on Rails Tutorial*](http://www.railstutorial.org/)
 [Майкл Хартл](http://www.michaelhartl.com/).
```

Фиксация

Когда изменения сделаны, можем взглянуть на статус ветки:

```
$ git status
On branch modify-README
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  renamed:    README.rdoc -> README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   README.md
```

Сейчас мы могли бы использовать `git add -A` как в Разделе 1.4.1, но `git commit` предусматривает флаг `-a` как сокращение для (очень частого) случая фиксации всех изменений существующих файлов (или файлов, созданных с использованием `git mv`, которые не считаются новыми для Git):

```
$ git commit -a -m "Improve the README file"
2 files changed, 5 insertions(+), 243 deletions(-)
delete mode 100644 README.rdoc
create mode 100644 README.md
```

[24]# Тут можно прочесть о нем на русском

Будьте осторожны с ошибочным использованием флага **-a**; если вы добавили какие-либо новые файлы в проект после последней фиксации, вы должны сообщить Git о них, выполнив сначала **git add -A**.

Обратите внимание, что мы написали сообщение коммита в *настоящем* времени (и, технически говоря, повелительном наклонении). Git моделирует коммиты как серии правок существующего кода, и в этом контексте имеет смысл описать что каждый коммит *делает*, нежели что он делал. Кроме того, такое использование соответствует сообщениям о коммитах, генерируемым самой командой Git. Более подробно об этом можно почитать в статье ``Блестящий новый стиль коммитов".

Слияние

Теперь, когда мы закончили все изменения, мы готовы *слить* результаты назад в master-ветку:

```
$ git checkout master
Switched to branch 'master'
$ git merge modify-README
Updating 34f06b7..2c92bef
Fast forward
README.rdoc     |  243 ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  --- --
README.md       |    5 +
2 files changed, 5 insertions(+), 243 deletions(-)
delete mode 100644 README.rdoc
create mode 100644 README.md
```

Вывод Git часто включает такие символы, как **34f06b7**, которые связаны с внутренним представлением репозиториев в Git. Ваши конкретные результаты будут отличаться в этих деталях, но остальное по своей сути должно соответствовать выводу, показанному выше.

После слияния изменений можно почистить ветки, удалив рабочую с помощью **git branch -d**, если вы с ней уже закончили:

```
$ git branch -d modify-README
Deleted branch modify-README (was 2c92bef).
```

Этот шаг не является обязательным, и, фактически, это довольно распространено --- оставлять рабочие ветки нетронутыми. Это позволяет переключаться между рабочей и master-ветками, сливая изменения каждый раз, когда вы достигаете естественной точки остановки.

Как упомянуто выше, вы также можете отказаться от изменений, относящихся к рабочей ветке, в таком случае используйте **git branch -D**:

```
# Для иллюстрации; пользуйтесь, только если совсем всё испортили в рабочей ветке
$ git checkout -b topic-branch
$ <действительно всё испортили>
$ git add -A
$ git commit -a -m "Major screw up"
$ git checkout master
$ git branch -D topic-branch
```

В отличие от флага **-d**, **-D** удалит ветку даже в том случае, если вы не слили изменения.

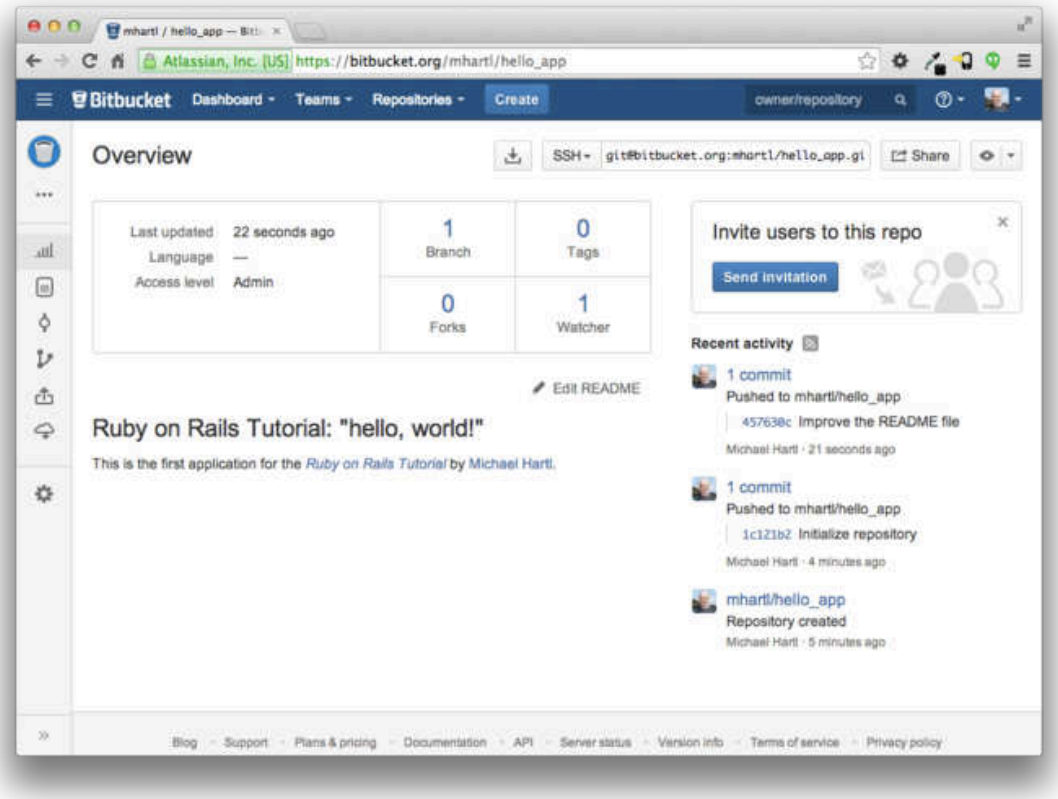


Рис. 1.17: Улучшенны файл **README**, *форматированный в Markdown.*

Отправка

После обновления **README** можно отправить изменения на Bitbucket, чтобы увидеть результат. Так одна отправка уже была сделана (Раздел 1.4.3), на большинстве систем теперь можно опустить **origin master**, и просто выполнить **git push**:

```
$ git push
```

Как было обещано в Разделе 1.4.4, Bitbucket приятно форматирует новый файл, используя Markdown (Рисунок 1.17).

1.5. Развёртывание

Даже на этой ранней стадии мы уже собираемся развернуть[25] наше (все еще пустое) Rails-приложение в production (окружение, используемое при развёртывании вашего приложения для всеобщего использования). Этот шаг не является обязательным, но раннее и частое развертывание позволяет отлавливать проблемы в самом начале нашего цикла разработки. Альтернативный вариант --- развертывание только после напряженных усилий, изолированных в окружении разработки (development environment) --- часто приводит к ужасным головным болям при интегрировании, когда приходит время запуска.[26]

Раньше развертывание Rails-приложений было довольно болезненной процедурой, но экосистема развертывания Rails стремительно развивалась в течение нескольких последних лет, и теперь в ней есть несколько замечательных инструментов. Среди них общедоступные хосты или виртуальные выделенные серверы, использующие Phusion Passenger (модуль для веб-серверов Apache и Nginx[27]), компании, предоставляющие полный комплекс услуг развертывания, такие как Engine Yard и Rails Machine, и облачные сервисы развертывания, такие как Engine Yard Cloud, Ninefold, и Heroku.

Моим любимым сервисом является Heroku, хостинговая платформа, созданная специально для того, чтобы разворачивать Rails и другие веб-приложения. Heroku делает развертывание Rails-приложений смехотворно легким --- пока ваш исходный код находится в системе управления версиями Git. (Это --- еще одна причина выполнить шаги установки Git из Раздела 1.4, если вы до сих пор этого не сделали.) Кроме всего прочего, бесплатного сервиса Heroku более, чем достаточно, для выполнения очень многих задач, в том числе и для этого учебника. Я не заплатил ни цента за хостинг первых его двух изданий на Heroku, обработавшем для меня несколько миллионов запросов.

Остальная часть раздела посвящена развертыванию нашего первого приложения на Heroku. Некоторые понятия весьма продвинуты, поэтому не стоит переживать, если не поймёте всех деталей; важно, чтобы в конце концов мы развернули приложение непосредственно в сети.

1.5.1. Установка Heroku

Heroku использует базу данных PostgreSQL[28] (произносится ``post-gres-cue-ell'', для краткости её часто называют ``Postgres''), а это означает, что в production-окружение необходимо добавить `pg` гем, чтобы позволить Rails общаться с Postgres:[29]

[25] # Пожалуй, это все-таки самый близкий перевод слова ``deploy''

[26] Хотя это и не должно иметь значения для учебных приложений в *Rails Tutorial*, если вы волнуетесь по поводу случайного и/или преждевременного обнародования вашего приложения, есть несколько способов, которые могут помочь вам избежать этого; один из них описан в Разделе 1.5.4.

[27] Произносится ``Engine X [ˈenjən eks]''.

[28] # Тут можно прочесть о ней на русском

[29] Вообще говоря, очень желательно, чтобы окружение разработки максимально соответствовало production (производственному) окружению, в том числе и в использовании базы данных, но для целей этого учебника мы всегда будем использовать SQLite локально и PostgreSQL в production. Больше информации в Разделе 3.1.

```
group :production do
  gem 'pg',             '0.17.1'
  gem 'rails_12factor', '0.0.2'
end
```

Обратите внимание на добавившийся гем `rails_12factor`, который Heroku использует для работы со статическими ассетами, такими как изображения и таблицы стилей. Наконец, обязательно проверьте, внесли ли вы изменения из Листинга 1.5, предотвращающие работу гема `sqlite3` в production-окружении, так как SQLite не поддерживатся на Heroku:

```
group :development, :test do
  gem 'sqlite3',     '1.3.9'
  gem 'byebug',      '3.4.0'
  gem 'web-console', '2.0.0.beta3'
  gem 'spring',      '1.1.3'
end
```

Полученный **`Gemfile`** показан в Листинге 1.14.

Листинг 1.14: **`Gemfile`** с добавлением гемов.

```
source 'https://rubygems.org'

gem 'rails',        '4.2.2'
gem 'sass-rails',   '5.0.2'
gem 'uglifier',     '2.5.3'
gem 'coffee-rails', '4.1.0'
gem 'jquery-rails', '4.0.3'
gem 'turbolinks',   '2.3.0'
gem 'jbuilder',     '2.2.3'
gem 'sdoc',         '0.4.0', group: :doc

group :development, :test do
  gem 'sqlite3',     '1.3.9'
  gem 'byebug',      '3.4.0'
  gem 'web-console', '2.0.0.beta3'
  gem 'spring',      '1.1.3'
end

group :production do
  gem 'pg',             '0.17.1'
  gem 'rails_12factor', '0.0.2'
end
```

Чтобы приготовить систему к развёртыванию в production, выполним **`bundle install`** со специальным флагом для предотвращения локальной инсталляции любых production-гемов (в данном случае `pg` и `rails_12factor`):

```
$ bundle install --without production
```

Так как в Листинг 1.14 были добавлены только гемы для production, то прямо сейчас эта команда не уста-

новит никаких дополнительных локальных гемов, но она добавит в **Gemfile.lock** гемы `pg` и `rails_12factor`. Зафиксируем полученные изменения:

```
$ git commit -a -m "Update Gemfile.lock for Heroku"
```

Затем необходимо создать и настроить новый аккаунт Heroku. Для начала нужно зарегистрироваться на Heroku. Затем проверим, есть ли уже в вашей системе установленный интерфейс командной строки Heroku:

```
$ heroku version
```

Те, кто пользуюся облачной IDE, должны увидеть номер версии Heroku, это говорит о доступности **heroku** CLI, на других системах может потребоваться его установка через Heroku Toolbelt.[30]

После того, как был установлен интерфейс командной строки Heroku, нужно выполнить команду **heroku** для входа в систему и добавления SSH-ключа:

```
$ heroku login
$ heroku keys:add
```

Наконец, команда **heroku create** создаст место на серверах Heroku, где будет жить учебное приложение (Листинг 1.15).

Листинг 1.15: Создание нового приложени на Heroku.

```
$ heroku create
Creating damp-fortress-5769... done, stack is cedar
http://damp-fortress-5769.herokuapp.com/ | git@heroku.com:damp-fortress-5769.git
Git remote heroku added
```

Команда **heroku** создает новый поддомен для нашего приложения, который незамедлительно доступен для просмотра. Однако там пока ничего нет, так что давайте займемся развертыванием.

1.5.2. Развертывание на Heroku, шаг первый

Первым шагом для развертывания является отправка master-ветки приложения на Heroku с помощью Git:

```
$ git push heroku master
```

(Может появиться несколько предупреждающих сообщений, которые сейчас можно проигнорировать. Мы обсудим их позже в Разделе 7.5.)

[30] https://toolbelt.heroku.com/

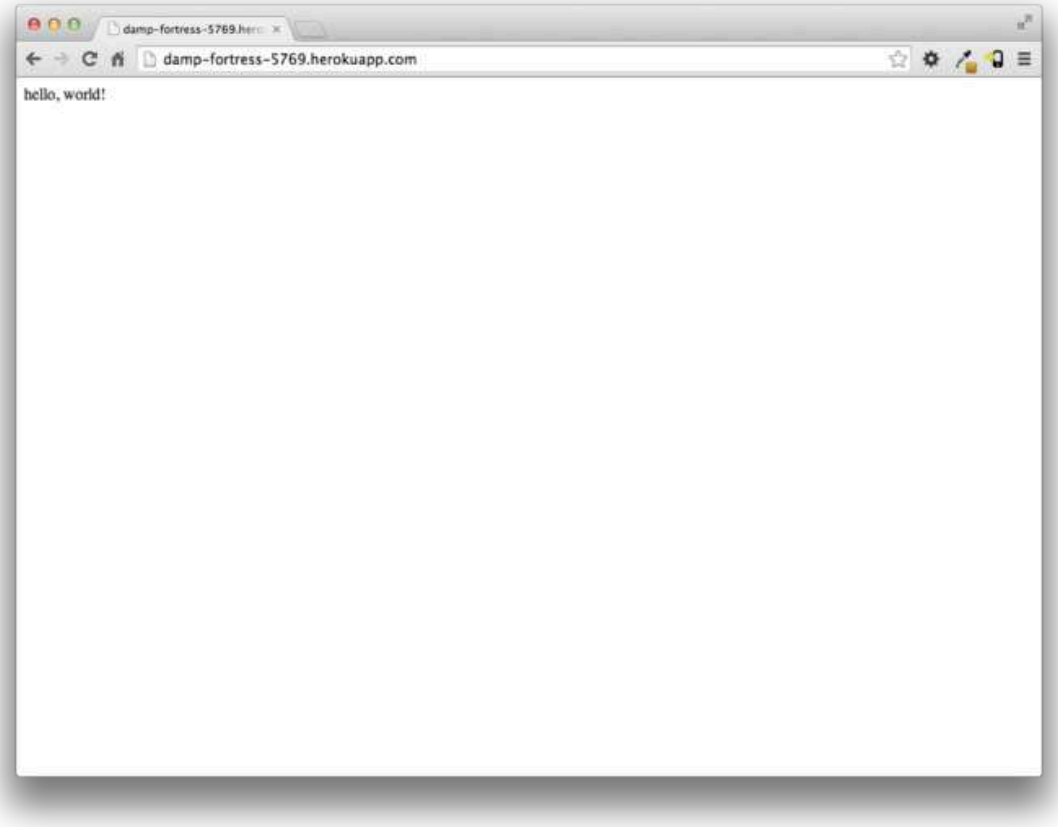


Рис. 1.18: Первое приложение Rails Tutorial, запущенное на Heroku.

1.5.3. Развертывание на Heroku, шаг второй

Нет никакого шага два! Все готово. Чтобы увидеть ваше свежеразвернутое приложение, нужно перейти по адресу, который вы видели при выполнении `heroku create` (то есть, в Листинге 1.15). (Если вы работаете на локальной машине вместо облачной IDE, можно выполнить `heroku open`.) Результат показан на Рисунке 1.18. Страница идентична Рисунку 1.12, но теперь она запущенна в production-окружении непосредственно в сети.

1.5.4. Команды Heroku

Существует великое множество команд Heroku, и мы только слегка коснемся их в этой книге. Уделю минуту, чтобы показать только одну из них, переименовав приложение следующим образом:

```
$ heroku rename rails-tutorial-hello
```

Не используйте сами это имя; я его уже занял! На самом деле, вам вряд ли стоит озадачиваться этим шагом прямо сейчас; использование адреса, по умолчанию предоставленного Heroku, --- это нормально. Но если вы действительно хотите переименовать свое приложение, можно сделать это, а заодно защитить его от непрошенных визитеров, используя случайный или невнятный субдомен, например такой:

```
hwpcbmze.herokuapp.com
seyjhflo.herokuapp.com
jhyicevg.herokuapp.com
```

С таким случайным субдоменом кто-либо сможет посетить ваш сайт, только если вы дадите ему адрес. (Между прочим, в качестве анонса удивительной компактности Ruby, вот код, который я использовал для генерации случайных субдоменов:

```
('a'..'z').to_a.shuffle[0..7].join
```

Довольно мило.)

В дополнение к поддержке субдоменов, Heroku также поддерживает пользовательские домены. (Фактически, сайт Ruby on Rails Tutorial живет на Heroku; если вы читаете эту книгу онлайн, то прямо сейчас смотрите на сайт, размещенный на Heroku!). См. документацию Heroku для получения дополнительной информации о пользовательских доменах и других возможностях Heroku.

1.6. Заключение

Мы проделали длинный путь в этой главе: установка, настройка среды разработки, управление версиями и развертывание. В следующей главе на основе полученных знаний мы построим связанное с базой данных *мини-приложение*, чтобы почуствовать первый реальный вкус того, что доступно Rails.

Если вы хотите поделиться с общественностью своим прогрессом, не стесняйтесь твитнуть или изменить свой статус в Facebook (Вконтакте) на что-то вроде этого:

Я изучаю Ruby on Rails с @railstutorial! http://www.railstutorial.org/

Также я рекомендую вам зарегистрироваться в email-списке Rails Tutorial[31], который гарантирует вам получение очередных обновлений (и эксклюзивных купонов), касающихся *Ruby on Rails Tutorial*.

[31] http://www.railstutorial.org/#email

1.6.1. Что мы изучили в этой главе

- Ruby on Rails --- это фреймворк для веб-разработки, написанный на языке программирования Ruby.
- Установка Rails, создание приложения, редактирование полученных файлов --- всё это очень легко при использовании преднастроенной облачной среды разработки.
- В командную строку Rails встроена команда `rails`, которая может генерировать новые приложения (`rails new`) и запускать локальный сервер (`rails server`).
- Мы добавили действие контроллера и изменили корневой маршрут, чтобы создать приложение ``hello, world".
- При помещении кода приложения в систему контроля версий Git и отправке его в приватный репозиторий на Bitbucket, мы защитили себя от потери данных, а также получили возможность совместной работы над проектом.
- Мы развернули приложение в production-окружении с помощью Heroku.

1.7. Упражнения

- Руководство по решению упражнений бесплатно прилагается к любой покупке на www.railstutorial.org.
- Зарегистрируйтесь в списке адресов электронной почты Rails Tutorial и получите в качестве бонуса шпаргалки для первого приложения и мини-приложения нашего учебника.

1. Измените содержание действия `hello` в Листинге 1.8 на ``hola, mundo!" вместо ``hello, world!". *Дополнительно*: Убедитесь в том, что Rails поддерживает символы не-ASCII, использовав перевернутый восклицательный знак ``¡Hola, mundo!" (Рисунок 1.19).[32]
2. Следуя примеру действия `hello` из Листинга 1.8, добавьте второе действие `goodbye`, которое будет отображать текст ``goodbye, world!". Отредактируйте файл маршрутов из Листинга 1.10, чтобы корневой маршрут приводил к `goodbye` вместо `hello` (Рисунок 1.20).

[32] Редактор можент выдать сообщение вроде ``invalid multibyte character" (``неверный мультибайтный символ"), но это не повод для беспокойства. Можете погуглить это сообщение, если вам интересно, как его устранить.

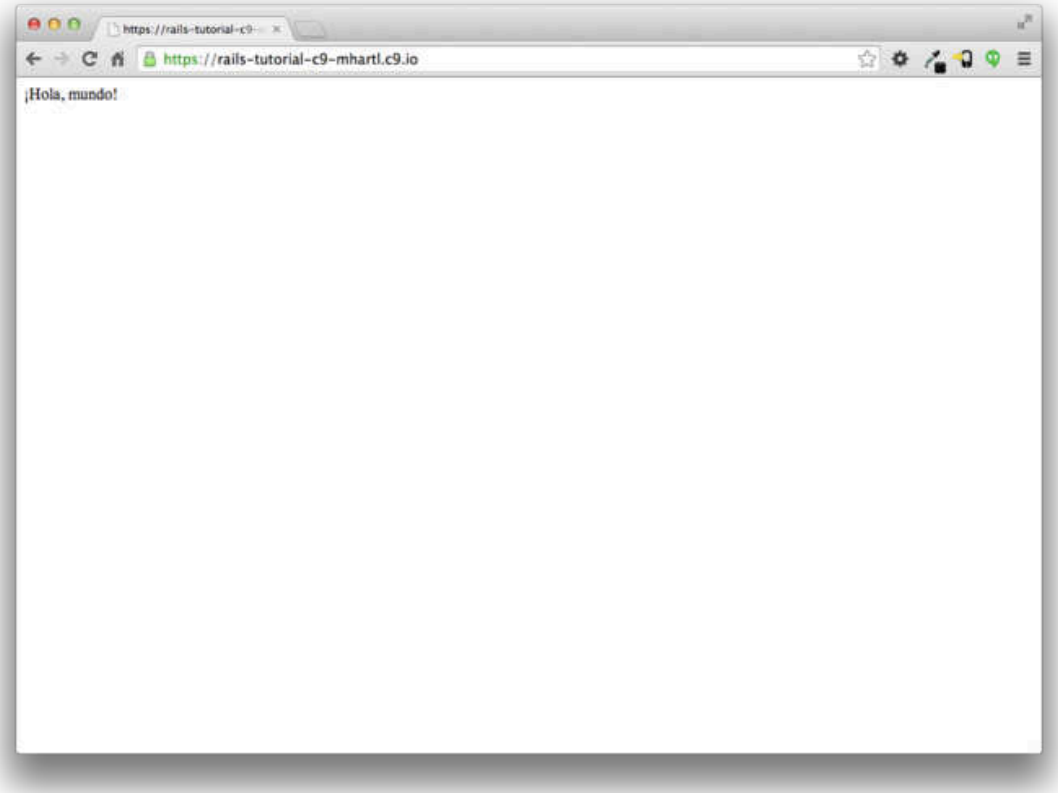


Рис. 1.19: Изменение действия контроллера для отображения ``¡Hola, mundo!''.

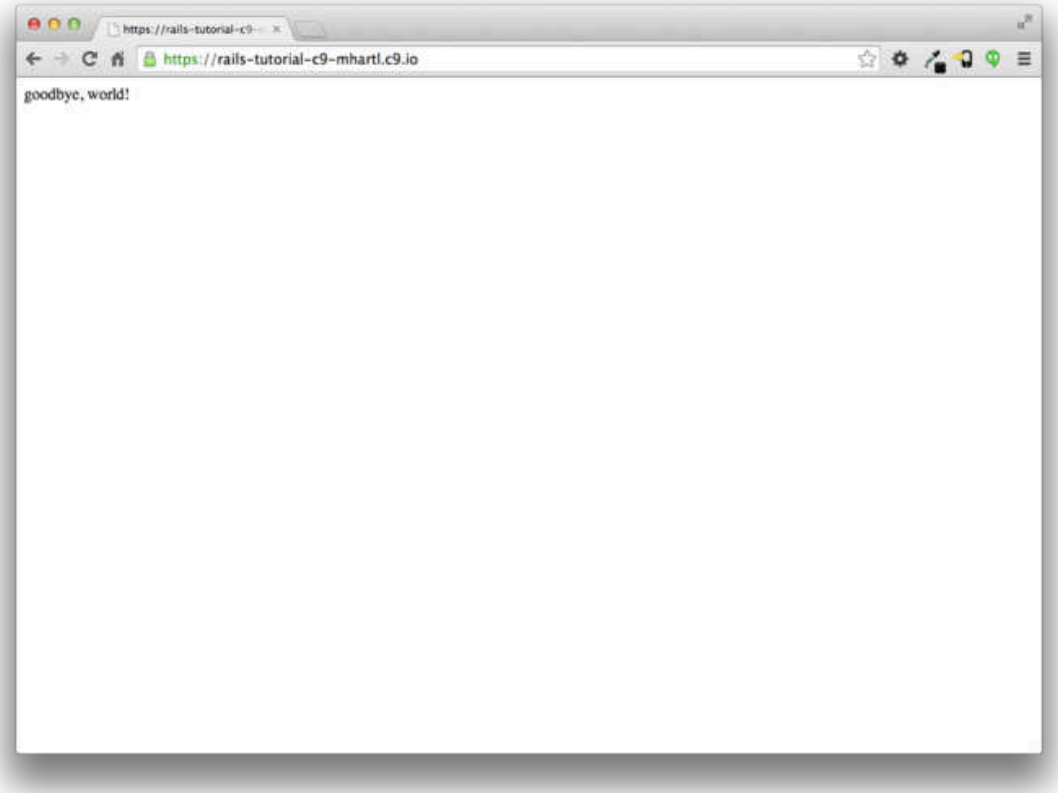


Рис. 1.20: Изменение корневого маршрута для отображения ``goodbye, world!''.

Глава 2

Мини-приложение

В этой главе мы разработаем миниатюрное демонстрационное приложение, чтобы похвастаться некоторыми способностями Rails. Цель --- получить общее представление о программировании на Ruby on Rails (и веб-разработке в целом), быстро сгенерировав приложение с помощью *scaffolding'a*, автоматически создающего большое количество функциональности. Как обсуждалось в Блоке 1.2, остальная часть книги будет применять противоположный подход, постепенно разрабатывая приложение с объяснением всех новых понятий по мере их появления, но для быстрого обзора (и немедленного удовлетворения) нет ничего лучше, чем scaffolding. Полученное в результате мини-приложение позволит взаимодействовать с ним через URL, дав нам понимание структуры Rails-приложений, включая первый пример *REST-архитектуры*, предпочитаемой Rails.

Как и предстоящее учебное приложение, миниатюрное будет состоять из *пользователей* и связанных с ними *микрособщений* (образуя минималистичное приложение в стиле Twitter). Функциональность будет совершенно слаборазвита, и многие из шагов будут походить на волшебство, но не переживайте: полное учебное приложение разработает подобный сайт с нуля, начиная с Главы 3, и я дам многочисленные ссылки на дальнейший материал. Тем временем, имейте терпение и немного веры --- все-таки основная цель данного учебника в том, чтобы удержать вас *вне* этого поверхностного, scaffold-ориентированного подхода для достижения более глубокого понимания Rails.

2.1. Планирование приложения

В этом разделе мы обрисуем в общих чертах наши планы относительно мини-приложения. Как и в Разделе 1.3, мы начнем с создания скелета приложения, используя команду **`rails new`** с указанием конкретного номера версии Rails:

```
$ cd ~/workspace
$ rails _4.2.2_ new toy_app
$ cd toy_app/
```

Если эта команда возвращает ошибку вроде ``Could not find 'railties'", значит у вас не установлена правильная версия Rails, и нужно перепроверить, была ли исполнена команда из Листинга 1.1 в точности так, как написана. (Если вы используете облачную IDE, как рекомендовано в Разделе 1.2.1, обратите внимание, что второе приложение может быть создано в том же рабочем пространстве, что и первое. Нет необходимости создавать новое рабочее пространство. Для того, чтобы отобразились файлы, может понадобиться кликнуть по значку шестерёнки в навигаторе файлов и выбрать пункт ``Обновить файловое дерево".)

Далее, в текстовом редакторе обновим `Gemfile`, необходимый Bundler'у, содержимым из Листинга 2.1. **Важно: Если вы читаете эту книгу не на railstutorial.org (а в бумажном издании, например), то вам необходимо использовать Gemfile, находящийся на gemfiles-3rd-ed.railstutorial.org, а не тот, что указан в книге.**

Листинг 2.1: `Gemfile` для мини-приложения.

```
source 'https://rubygems.org'

gem 'rails',        '4.2.2'
gem 'sass-rails',   '5.0.2'
gem 'uglifier',     '2.5.3'
gem 'coffee-rails', '4.1.0'
gem 'jquery-rails', '4.0.3'
gem 'turbolinks',   '2.3.0'
gem 'jbuilder',     '2.2.3'
gem 'sdoc',         '0.4.0', group: :doc

group :development, :test do
  gem 'sqlite3',     '1.3.9'
  gem 'byebug',      '3.4.0'
  gem 'web-console', '2.0.0.beta3'
  gem 'spring',      '1.1.3'
end

group :production do
  gem 'pg',             '0.17.1'
  gem 'rails_12factor', '0.0.2'
end
```

Отмечу, что Листинг 2.1 идентичен Листингу 1.14.

Как и в Разделе 1.5.1, мы установим локальные гемы и предотвратим установку production-гемов с помощью опции `--without production`:

```
$ bundle install --without production
```

Наконец, поместим мини-приложение в систему контроля версий Git:

```
$ git init
$ git add -A
$ git commit -m "Initialize repository"
```

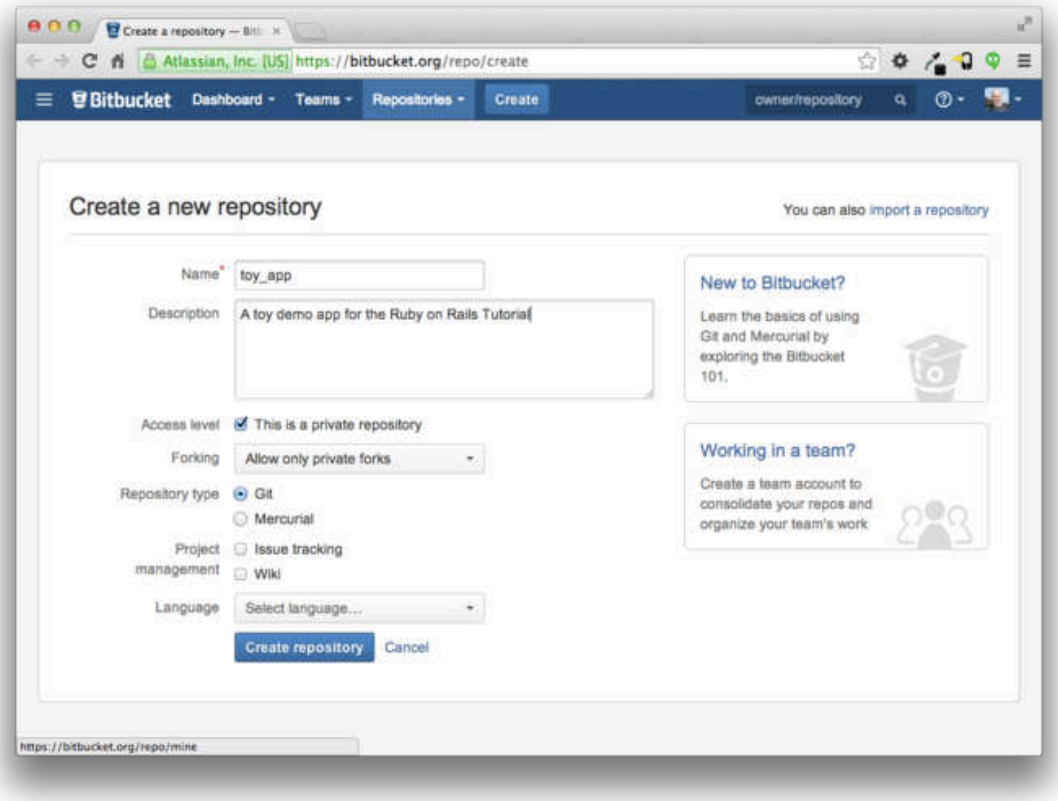


Рис. 2.1: Создание репозитория мини-приложения на Bitbucket.

Вам необходимо также создать новый репозиторий, нажав на кнопку``Create" на Bitbucket (Рисунок 2.1), и отправить всё в удалённый репозиторий:

```
$ git remote add origin git@bitbucket.org:<username>/toy_app.git
$ git push -u origin --all # впервые отправляем репозиторий и ссылки
```

Ну и наконец, никогда не бывает слишком рано для развёртывания, которое я предлагаю делать по той же схеме, что и ``hello, world!" в Листинге 1.8 и Листинге 1.9.[1] Затем зафиксируйте изменения и отправьте на Heroku.

```
$ git commit -am "Add hello"
$ heroku create
$ git push heroku master
```

[1]Созданные по умолчанию Rails-страницы как правило не работают на Heroku, поэтому сложно сказать, было ли успешным развёртвание.

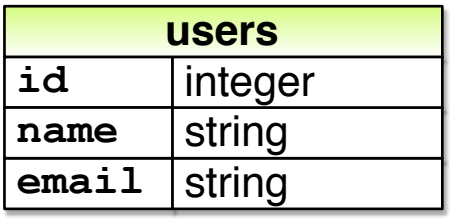


Рис. 2.2: Модель данных для пользователей.

(Как и в Разделе 1.5, вы можете увидеть несколько предупреждающих сообщений, которые на данном этапе можно игнорировать. Избавимся от них в Разделе 7.5.) Кроме адреса приложения на Heroku, результат должен совпадать с Рисунком 1.18.

Теперь мы готовы начать делать само приложение. Типичный первый шаг в разработке --- это создание *модели данных*, которая является представлением структур, необходимых приложению. В нашем случае мини-приложение будет упрощенным микроблогом: только пользователи и короткие (микро)сообщения. Таким образом, мы начнем с модели *пользователей* (Раздел 2.1.1), затем добавим модель *микрособщений* (Раздел 2.1.2).

2.1.1. Мини-модель пользователей

Вариантов для модели данных пользователя так же много, как и различных форм регистрации в сети; мы пойдем весьма минималистичным путем. У пользователей мини-приложения будет уникальный `integer` (целочисленный) идентификатор, называемый `id`, публично видимое `имя` (тип --- `string` (строка)), и адрес электронной почты `email` (тоже `string`), совпадающий с именем пользователя. Итоговая модель данных для пользователей представлена на Рисунке 2.2.

Как мы увидим в Разделе 6.1.1, заголовок `users` соответствует *таблице* в базе данных, а атрибуты `id`, `name`, и `email` --- это *столбцы* в ней.

2.1.2. Мини-модель для микрособщений

Ядро модели данных микрособщений даже проще, чем для пользователей: здесь есть только `id` и поле `content` для текста микрособщения (тип `text`).[2] Есть дополнительная сложность: мы хотим *связать* каждое микрособщение с определенным пользователем. Мы выполним это, записав `user_id` владельца сообщения. Результаты показаны на Рисунке 2.3.

Мы увидим в Разделе 2.3.3 (и более полно в Разделе 11), как атрибут `user_id` позволит нам кратко реализовать идею о том, что у пользователя потенциально есть много связанных с ним микрособщений.

[2] Так как микрособщения планируются короткими, то для их содержимого был бы вполне достаточным тип `string`, но использование типа `text` лучше выражает нашу цель, а также даёт бОльшую гибкость, и позволяет не задумываться об ограничении длины.

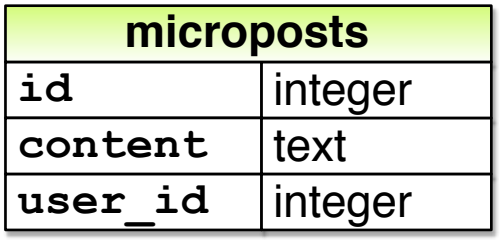


Рис. 2.3: Модель данных для микрособщений.

2.2. Ресурс Users

В этом разделе мы реализуем модель данных пользователей из Раздела 2.1.1, вместе с веб-интерфейсом к ней. Эта комбинация модели и интерфейса образует *ресурс Users*, который позволит нам думать о пользователях, как об объектах, которые могут быть созданы, считаны, обновлены, и удалены через сеть с использованием HTTP-протокола. Как и было обещано во введении, ресурс Users будет создаваться программой scaffold generator, которая стандартно поставляется с каждым проектом Rails. Я настоятельно прошу вас не всматриваться в генерируемый код; на этом этапе он лишь запутает вас.

Rails scaffolding запускается передачей команды `scaffold` скрипту `rails generate`. Аргумент команды `scaffold` --- это имя ресурса в единственном числе (в данном случае, `User`), вместе с дополнительными параметрами для атрибутов модели данных:[3]

```
$ rails generate scaffold User name:string email:string
      invoke  active_record
      create    db/migrate/20140821011110_create_users.rb
      create    app/models/user.rb
      invoke    test_unit
      create      test/models/user_test.rb
      create      test/fixtures/users.yml
      invoke  resource_route
       route    resources :users
      invoke  scaffold_controller
      create    app/controllers/users_controller.rb
      invoke    erb
      create      app/views/users
      create      app/views/users/index.html.erb
      create      app/views/users/edit.html.erb
      create      app/views/users/show.html.erb
      create      app/views/users/new.html.erb
      create      app/views/users/_form.html.erb
      invoke    test_unit
      create      test/controllers/users_controller_test.rb
      invoke    helper
      create      app/helpers/users_helper.rb
```

[3]Именование scaffold следует соглашению об именовании *моделей*, которые используются в единственном числе, в отличие от ресурсов и контроллеров, которые используются во множественном числе. Таким образом, получаем `User` вместо `Users`.

```
      invoke      test_unit
      create        test/helpers/users_helper_test.rb
      invoke    jbuilder
      create      app/views/users/index.json.jbuilder
      create      app/views/users/show.json.jbuilder
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/users.js.coffee
      invoke    scss
      create      app/assets/stylesheets/users.css.scss
      invoke  scss
      create    app/assets/stylesheets/scaffolds.css.scss
```

Добавив **`name:string`** и **`email:string`**, мы добились того, что модель User приобрела форму, соответствующую Рисунку 2.2. (Обратите внимание на то, что нет надобности включать параметр **`id`** --- он создается Rails автоматически для использования в качестве *первичного ключа* в базе данных.)

Для того, чтобы продолжить с мини-приложением, сначала необходимо *migrate (мигрировать, переместить)* базу данных, используя *Rake* (Блок 2.1):

```
$ bundle exec rake db:migrate
==  CreateUsers: migrating ====================================================
-- create_table(:users)
   -> 0.0017s
==  CreateUsers: migrated (0.0018s) ===========================================
```

Таким образом мы просто обновляем базу данных новой моделью **`users`**. (Мы узнаем больше о миграциях баз данных в Разделе 6.1.1.) Обратите внимание, что для обеспечения использования командами версии Rake, соответствующей **`Gemfile`**, необходимо запускать **`rake`** при помощи **`bundle exec`**. На многих системах, включая облачную IDE, можно опустить **`bundle exec`**, но на некоторых он необходим, поэтому я буду добавлять его для полноты картины.

Теперь мы можем запустить локальный веб-сервер в отдельной вкладке (Рисунок 1.7):[4]

```
$ rails server -b $IP -p $PORT # Просто `rails server`, если запускаете локально
```

Сейчас мини-приложение должно быть доступно на локальном сервере, как описано в Разделе 1.3.2. (Если работаете в облачной IDE, обязательно откройте полученный сервер в новой вкладке *браузера*, а не внутри самой IDE.)

Блок 2.1. Rake

В традициях Unix, утилита *make* играет важную роль в сборке исполняемых программ из исходного кода; множество компьютерных хакеров зафиксировали в мышечной памяти строку

[4] Скрипт **`rails`** построен таким образом, что не нуждается в **`bundle exec`**.

URL	Действие	Предназначение
/users	index	страница для отображения списка всех пользователей
/users/1	show	страница для отображения пользователя с id 1
/users/new	new	страница для создания нового пользователя
/users/1/edit	edit	страница для редактирования пользователя с id 1

Таблица 2.1: Соответствие между страницами и URL для ресурса Users.

```
$ ./configure && make && sudo make install
```

обычно используемую для компиляции кода на Unix-системах (включая Linux и Mac OS X).

Rake --- это *Ruby make*, make-подобный язык, написанный на Ruby. Rails использует Rake очень широко, особенно для решения бесчисленных мелких административных задач, необходимых при разработке веб-приложений, опирающихся на базы данных. Команда `rake db:migrate`, вероятно, наиболее распространена, но есть и многие другие; можно увидеть список задач базы данных при помощи `-T db`:

```
$ bundle exec rake -T db
```

Чтобы увидеть все доступные Rake задачи, запустите

```
$ bundle exec rake -T
```

Список, вероятно, будет потрясающим, но не волнуйтесь, вы не должны знать все (или даже большую часть) этих команд. К концу *Rails Tutorial* вы будете знать все самые важные.

2.2.1. Обзор пользователя

При посещении корневого URL / (читается``слэш", как указано в Разделе 1.3.4) мы увидим ту же самую страницу Rails, созданную по умолчанию, что и на Рисунке 1.9; но кроме неё мы создали большое количество страниц для управления пользователями при генерировании ресурса Users методом scaffold. Например, страницу для отображения списка всех пользователей на /users, и страницу для создания нового пользователя на /users/new. Остальная часть этого раздела будет посвящена беглому обзору всех этих страниц. В процессе для вас может оказаться полезным иногда поглядывать в Таблицу 2.1, которая показывает соответствие между страницами и их URL.

Мы начнем со страницы, показывающей всех пользователей в нашем приложении, она называется `index`; как вы могли ожидать, изначально на ней нет никаких пользователей (Рисунок 2.4).

Для того, чтобы создать нового пользователя, посетим страницу `new`, она показана на Рисунке 2.5. (Поскольку http://0.0.0.0:3000 или часть адреса облачной IDE подразумевается, пока мы разрабатываем локально, я буду опускать ее с этого момента.) В Главе 7 она станет страницей регистрации пользователя. Мы можем создать пользователя, введя значения имени и адреса электронной почты в текстовые поля, и

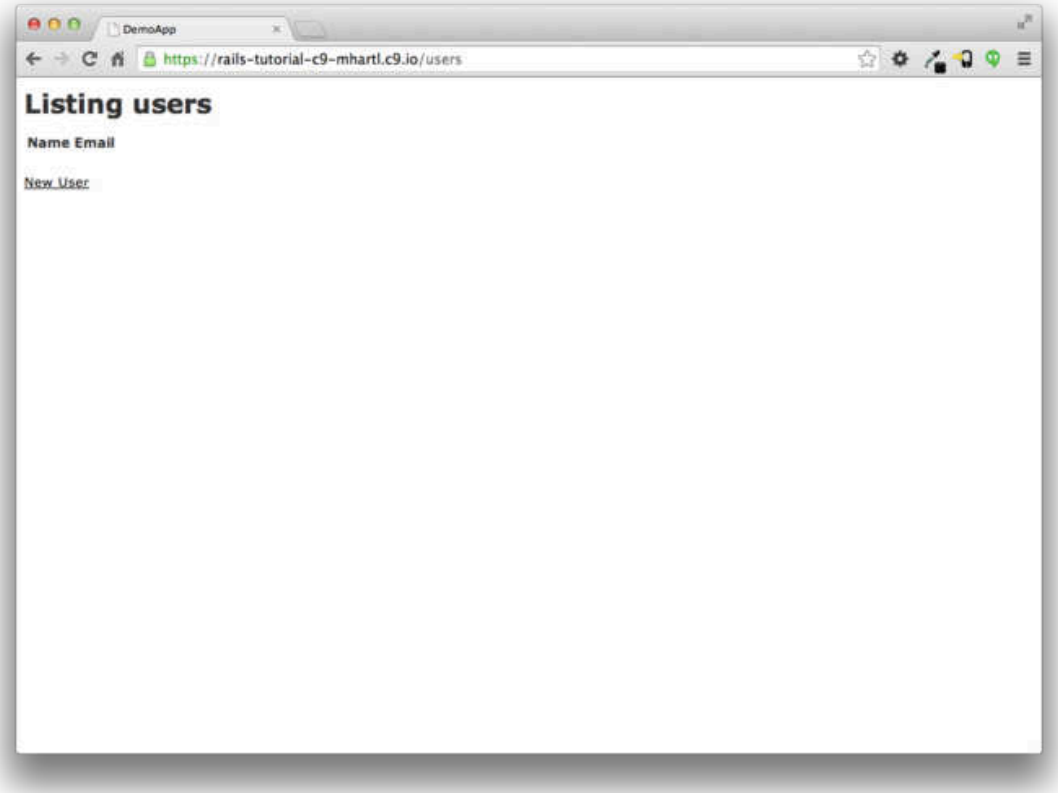


Рис. 2.4: Начальная index-страница для ресурса Users (/users).

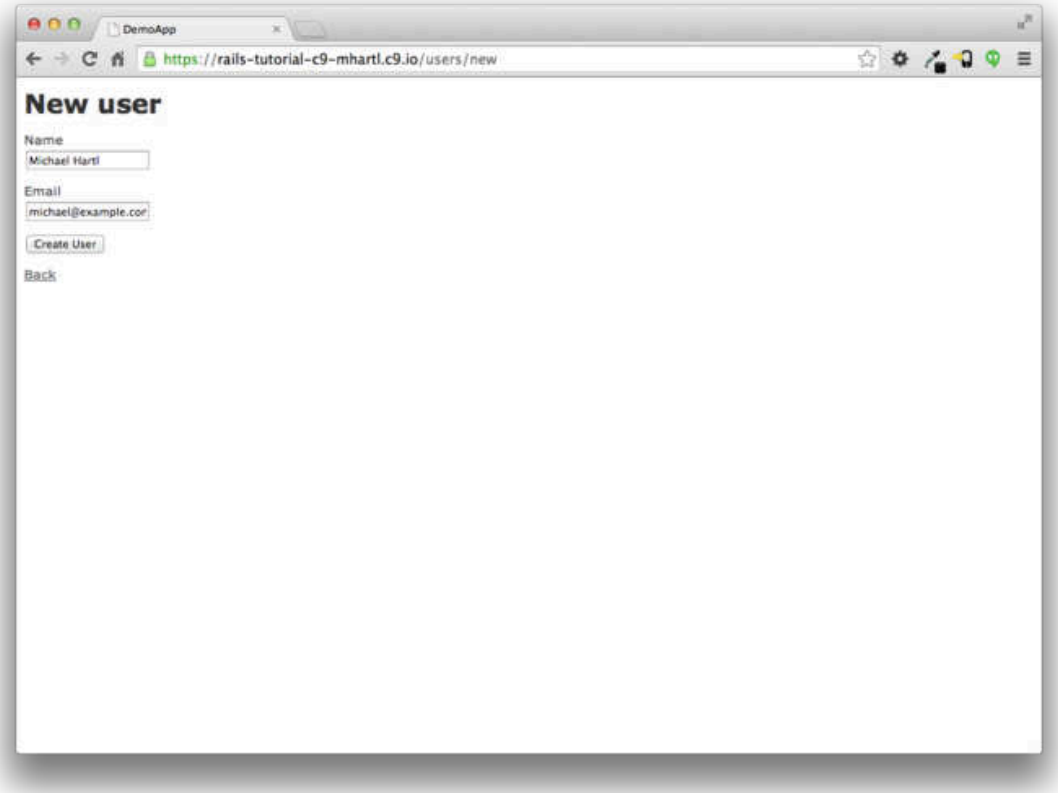


Рис. 2.5: Страница создания нового пользователя (/users/new).

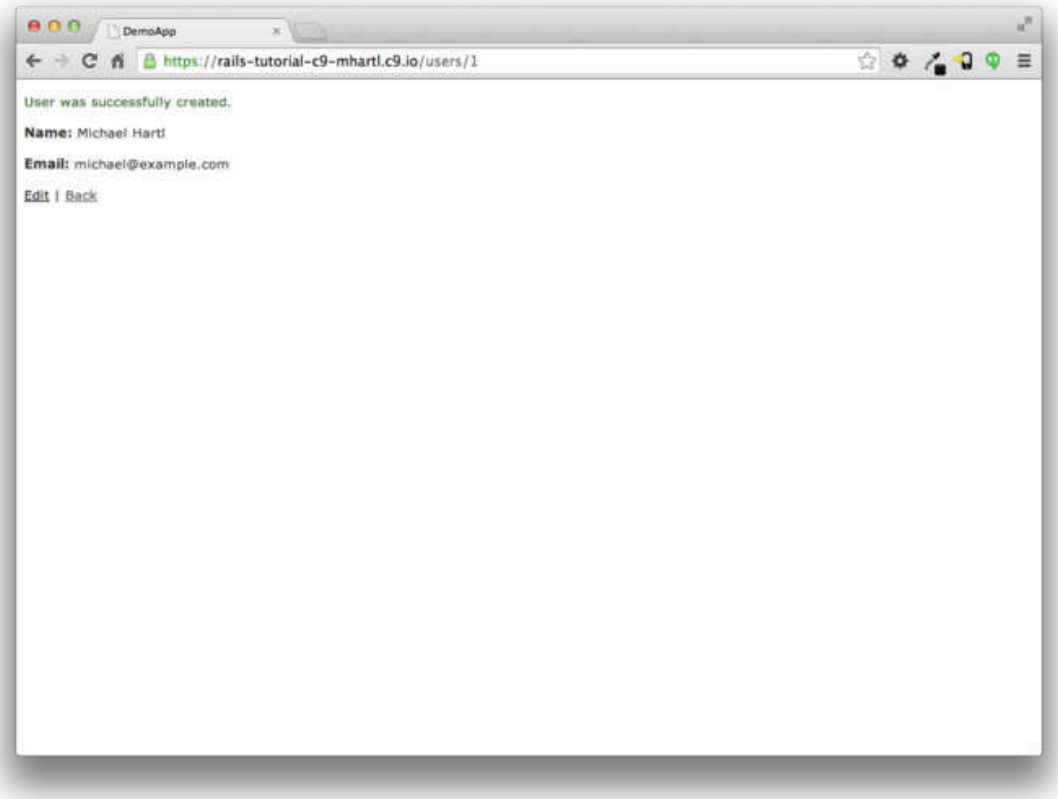


Рис. 2.6: Страница, показывающая пользователя (/users/1).

нажав затем кнопку Create User. Результат --- страница пользователя `show`, как показано на Рисунке 2.6. (Зеленое приветственное сообщение выполняется с использованием *флэш*, о котором мы узнаем в Разделе 7.4.2.) Обратите внимание на URL --- /users/1; как вы можете предположить, цифра `1` --- это просто атрибут `id` из Рисунка 2.2. В Разделе 7.1 эта страница станет профилем пользователя.

Для того, чтобы отредактировать данные пользователя, мы посетим страницу `edit` (Рисунок 2.7). Изменяя данные и нажимая кнопку Update User, мы изменяем информацию о пользователе в мини-приложении (Рисунок 2.8). (В Главе 6 мы детально разберём, что эти данные пользователя хранятся в базе данных.) Мы добавим функции edit/update (редактировать/обновить) пользователя к учебному приложению в Разделе 9.1.

Теперь мы создадим второго пользователя, повторно посетив страницу `new` и отправив второй набор данных; получившаяся страница `index` показана на Рисунке 2.9. Раздел 7.1 преобразует этот список в более изысканную страницу демонстрации всех пользователей.

Показав, как создавать, отображать и редактировать пользователей, мы переходим, наконец, к их уда-

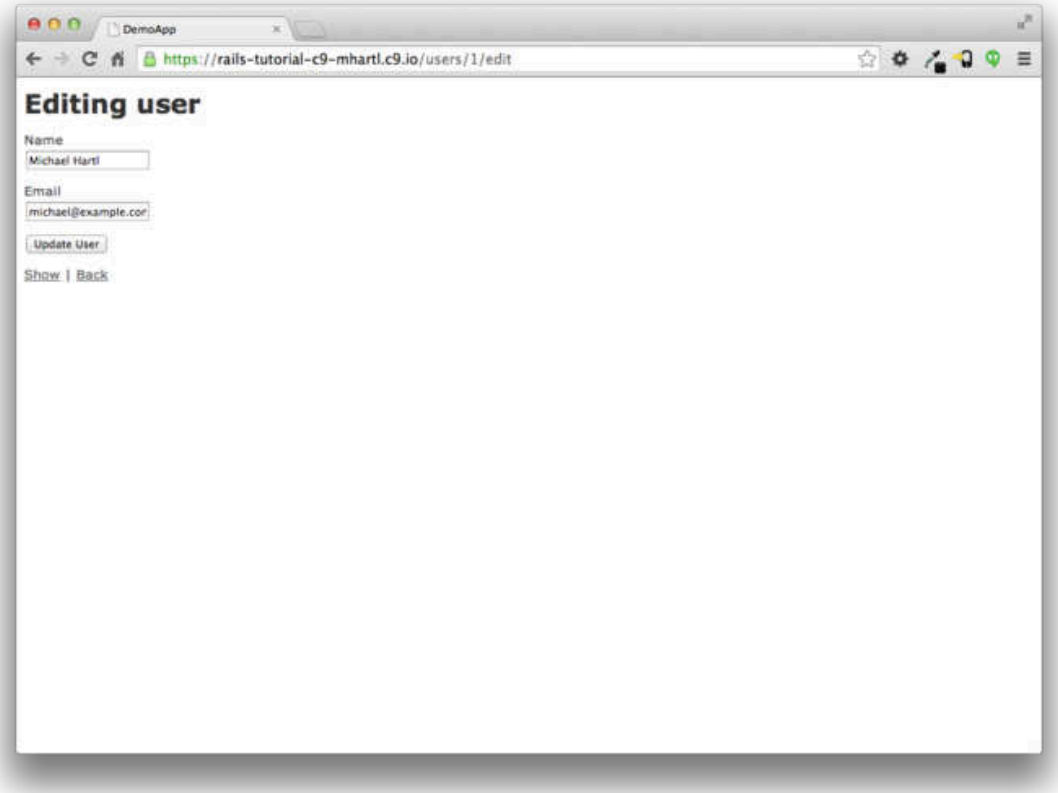


Рис. 2.7: Страница редактирования пользователя (/users/1/edit).

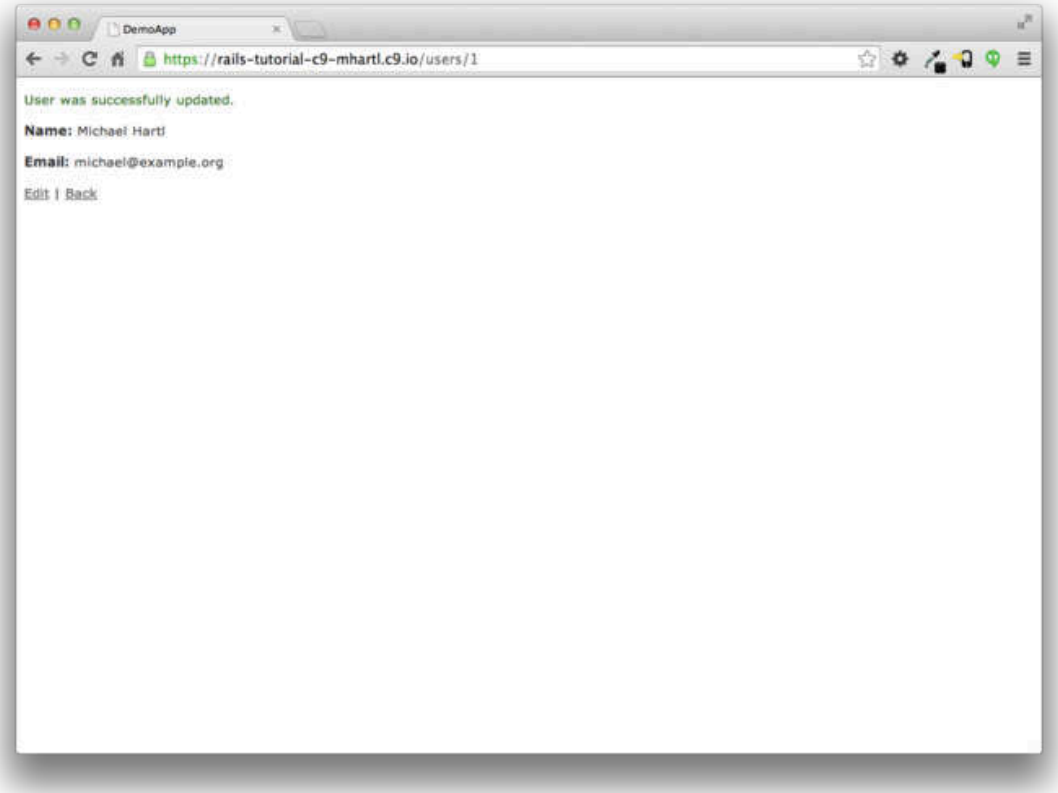


Рис. 2.8: Пользователь с обновленной информацией.

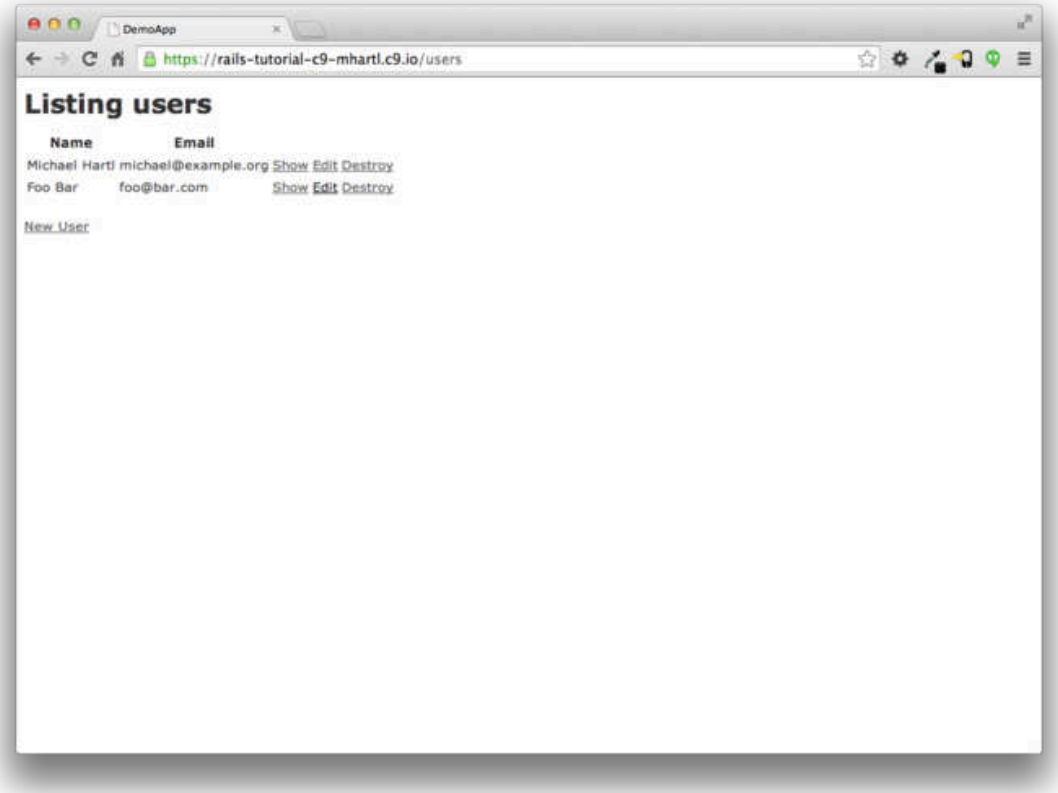


Рис. 2.9: Страница-список (/users) со вторым пользователем.

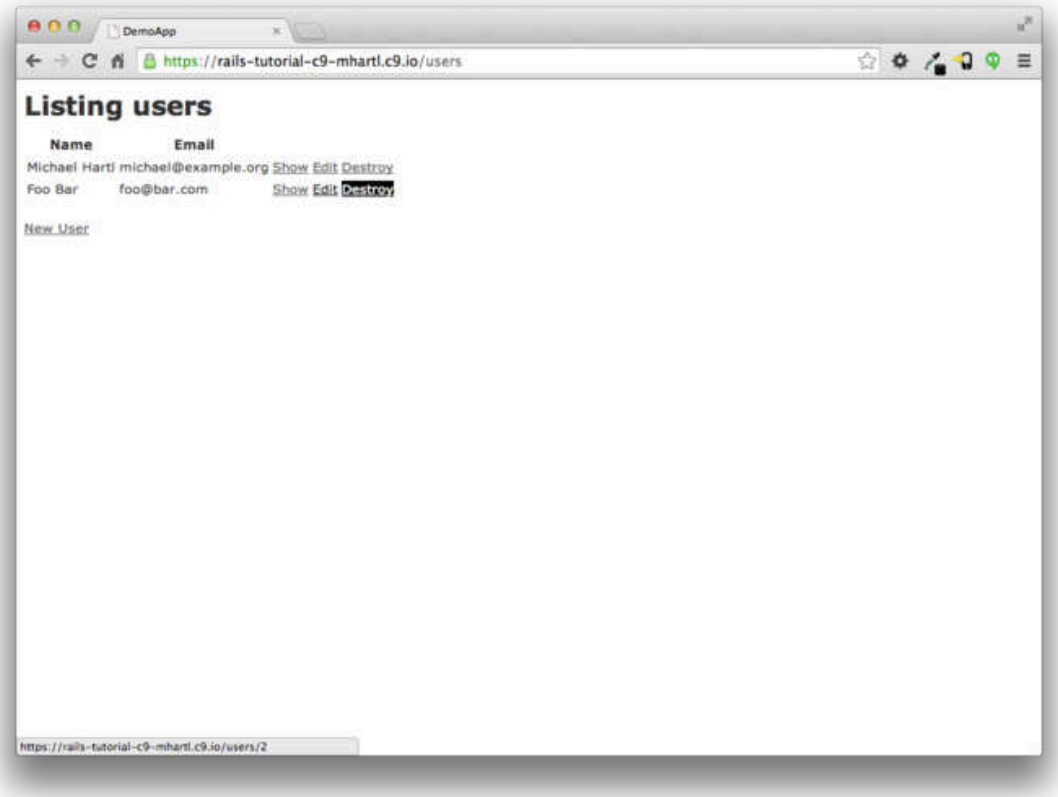


Рис. 2.10: Уничтожение пользователя.

лению (Рисунок 2.10). Следует проверить, что клик по ссылке, показанной на Рисунке 2.10, уничтожает второго пользователя, приводя к index-странице с одним пользователем. (Если это не работает, убедитесь, что JavaScript включен в вашем браузере; Rails использует JavaScript, чтобы выдать запрос, необходимый для уничтожения пользователя.) Раздел 9.4 добавляет удаление пользователя к учебному приложению, заботясь об ограничении его использования специальным классом административных пользователей.

2.2.2. MVC в действии

Теперь, когда мы завершили быстрый обзор ресурса Пользователи (Users), давайте исследуем отдельную его часть в контексте схемы Модель-Представление-Контроллер (MVC), представленной в Разделе 1.3.3. Наша стратегия будет состоять в том, чтобы описать результаты типичного запроса браузера --- посещение index-страницы /users --- в терминах MVC (Рисунок 2.11).

На Рисунке 2.11 показано резюме шагов:

1. Браузер выдает запрос на URL /users.

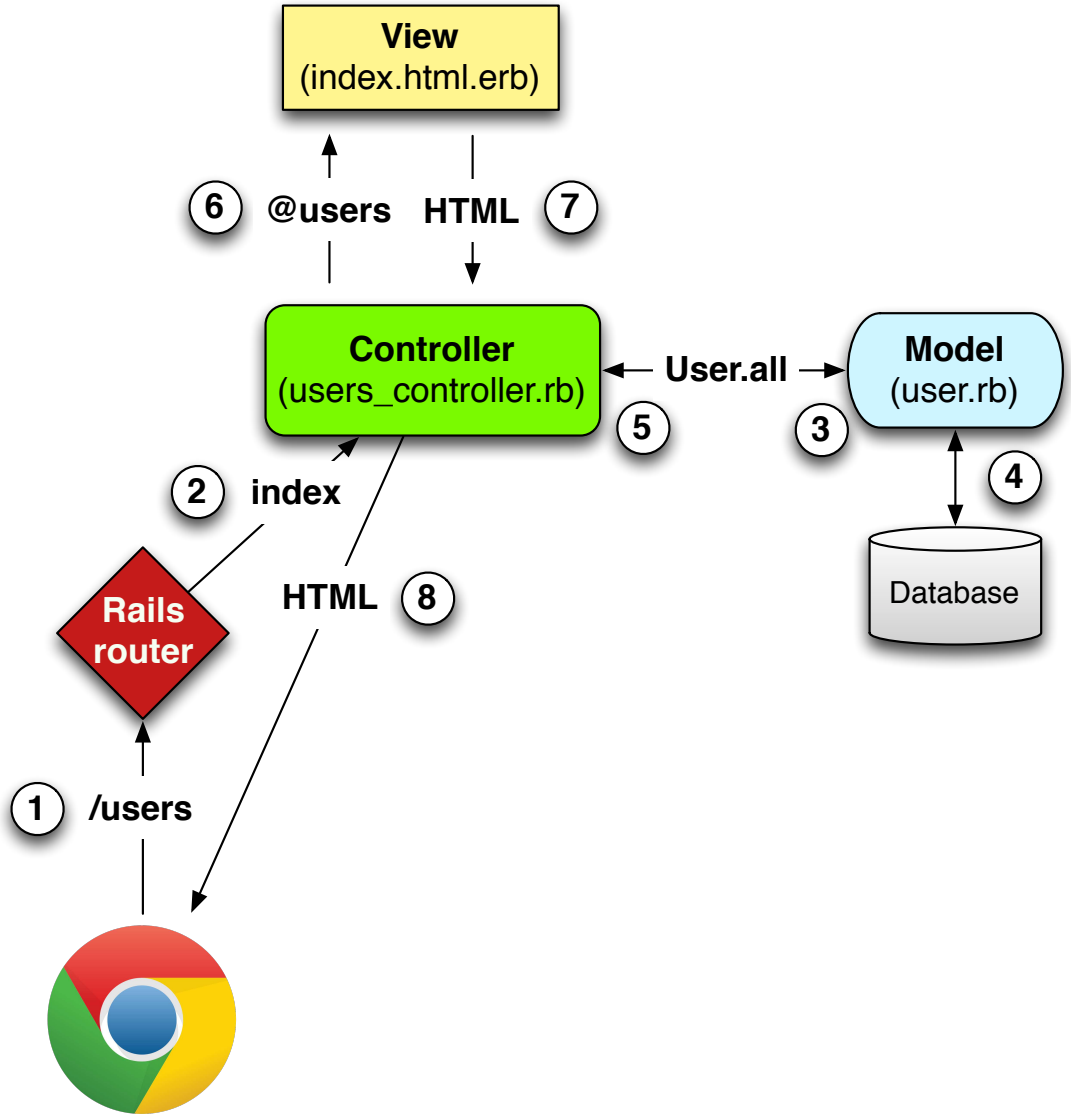


Рис. 2.11: Подробная схема MVC в Rails.

2. Rails направляет /users к действию **`index`** в контроллере Users.
3. Действие **`index`** запрашивает у модели User получение всех пользователей (**`User.all`**).
4. Модель User вытягивает всех пользователей из базы данных.
5. Модель User возвращает список пользователей в контроллер.
6. Контроллер получает пользователей в переменной **`@users`**, которую он передаёт представлению **`index`**.
7. Представление использует Embedded (Встроенный) Ruby, чтобы визуализировать страницу в виде HTML.
8. Контроллер возвращает HTML в браузер.[5]

Мы начинаем с запроса, выданного браузером --- то есть, с результата ввода URL в адресной строке или клика по ссылке (Шаг 1 на Рисунке 2.11). Этот запрос вызывает *Rails-маршрутизатор* (Шаг 2), который направляет к соответствующему *действию контроллера*, опираясь на URL (и, как мы увидим в Блоке 3.2, на тип запроса). Код, создающий перенаправление URL пользователя к действиям контроллера для ресурса Users, представлен в Листинге 2.2; этот код фактически связывает пары URL/Действие, которые мы видели в Таблице 2.1. (Странная запись **`:users`** --- это *символ*, о которых мы узнаем в Разделе 4.3.3.)

Листинг 2.2: Маршруты Rails, с правилом для ресурса Users.
config/routes.rb

```
Rails.application.routes.draw do
  resources :users
  .
  .
  .
end
```

Пока мы находимся в файле маршрутов, давайте воспользуемся моментом и свяжем корневой маршрут со страницей отображения пользователей, так, чтобы ``слэш'' приводил к /users. Вспомним Листинг 1.10,

```
# root 'welcome#index'
```

который мы изменили на

[5] Некоторые источники утверждают, что представление возвращает HTML непосредственно браузеру (через веб-сервер, такой как Apache или Nginx). Независимо от деталей реализации, я предпочитаю думать о контроллере как о центральном хабе, через который проходят информационные потоки всего приложения

```
root 'application#hello'
```

чтобы корневой маршрут приводил к действию **hello** в Application-контроллере. Теперь нам понадобится действие **index** в контроллере Users, и мы можем добиться этого с помощью когда из Листинга 2.3. (Здесь я рекомендую удалить действие **hello** из Application-контроллера, если вы добавили его в начале раздела.)

Листинг 2.3: Добавление корневого маршрута для users.
config/routes.rb

```
Rails.application.routes.draw do
  resources :users
  root 'users#index'
  .
  .
  .
end
```

Страницы из обзора в Разделе 2.2.1 соответствуют *действиям* в *контроллере* Users, который по сути является набором связанных действий. Контроллер, сгенерированный методом scaffold, схематично показан в Листинге 2.4. Обратите внимание на запись **class UsersController < ApplicationController**, которая является примером Ruby-*класса* с *наследованием*. (Мы вкратце обсудим наследование в Разделе 2.3.4 и раскроем обе темы более подробно в Разделе 4.4.)

Листинг 2.4: Контроллер Users в схематичной форме.
app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  .
  .
  .
  def index
    .
    .
    .
  end

  def show
    .
    .
    .
  end

  def new
    .
    .
    .
  end

  def edit
    .
    .
```

HTTP-запрос	URL	Действие	Предназначение
GET	/users	**index**	страница со списком всех пользователей
GET	/users/1	**show**	страница для отображения пользователя с id **1**
GET	/users/new	**new**	страница для создания нового пользователя
POST	/users	**create**	создает нового пользователя
GET	/users/1/edit	**edit**	страница для редактирования пользователя с id **1**
PATCH	/users/1	**update**	обновляет данные пользователя с id **1**
DELETE	/users/1	**destroy**	удаляет пользователя с id **1**

Таблица 2.2: RESTful маршруты, обеспеченные ресурсом Users в Листинге 2.2.

```
    .
  end

  def create
    .
    .
    .
  end

  def update
    .
    .
    .
  end

  def destroy
    .
    .
    .
  end
end
```

Вы могли заметить, что действий больше, чем страниц. Действия **index**, **show**, **new** и **edit** --- все соответствуют страницам из Раздела 2.2.1, но есть и дополнительные: **create** (создать), **update** (обновить) и **destroy** (уничтожить). Эти последние обычно не визуализируют страниц (хотя иногда могут); вместо этого, их основная цель состоит в том, чтобы изменять информацию о пользователях в базе данных. Этот полный комплект действий контроллера, сведенный в Таблицу 2.2, представляет собой реализацию архитектуры REST в Rails (Блок 2.2), которая опирается на идею *передачи состояния представления*, сформулированную учёным в области информатики Роем Филдингом.[6] Обратите внимание в Таблице 2.2 на некоторое наложение в URL; например, оба действия **show** и **update** соответствуют URL /users/1. Различие между ними в методах запроса HTTP, на которые они отвечают. Мы узнаем больше о методах запроса HTTP в Разделе 3.3.

Блок 2.2. REpresentational State Transfer (REST)

[6] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

Если вы много читали о веб-разработке на Ruby on Rails, вы видели много ссылок на ``REST'', что является сокращением от REpresentational State Transfer («передача состояния представления». REST --- архитектурный стиль для разработки распределенных сетевых систем и приложений. Хотя теория REST довольно абстрактна, в контексте Rails-приложений REST означает, что большинство компонентов приложения (таких как пользователи и микрособщения) моделируются как *ресурсы*, которые могут быть созданы (Created), прочитаны (Read), обновлены (Updated) и удалены (Deleted) --- эти операции соответствуют и CRUD-операциям в реляционных базах данных, и четырём основным методам запросов HTTP: `POST`, `GET`, `PATCH` и `DELETE`.[7] (Мы узнаем больше о запросах HTTP в Разделе 3.3 и особенно в Блоке 3.2.)

RESTful стиль разработки помогает вам, как разработчику Rails-приложений, определиться --- какие контроллеры и действия писать: вы просто структурируете приложение, используя ресурсы, которые создаются, читаются, обновляются, и удаляются. В случае с пользователями и микрособщениями все пространство, так как это --- натуральные ресурсы по определению. В Главе 12 мы увидим пример, где принципы REST позволяют нам моделировать более тонкую проблему, ``Следование за пользователями '', естественным и удобным способом.

Чтобы исследовать отношения между контроллером Users и моделью User, давайте сосредоточимся на упрощенной версии действия **`index`**, показанной в Листинге 2.5. (Scaffold-код уродливый и запутанный, так что я опустил большую его часть.)

Листинг 2.5: Упрощённое действие **`index`** для мини-приложения.
app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  .
  .
  .
  def index
    @users = User.all
  end
  .
  .
  .
end
```

В действии **`index`** находится строка **`@users = User.all`** (Шаг 3 на Рисунке 2.11), которая запрашивает у модели User всех пользователей из базы данных (Шаг 4), и затем помещает их в переменную **`@users`** (читается``at-users'') (Шаг 5). Сама модель User показана в Листинге 2.6; хотя она довольно проста, к ней прилагается большое количество функций из-за наследования (Раздел 2.3.4 и Раздел 4.4). В частности, за счет использования Rails-библиотеки *Active Record*, код в Листинге 2.6 заставляет **`User.all`** возвратить всех пользователей из базы данных.

Листинг 2.6: Модель User для мини-приложения.
app/models/user.rb

```
class User < ActiveRecord::Base
end
```

Как только переменная **@users** определена, контроллер вызывает *представление* (Шаг 6), показанное в Листинге 2.7. Переменные, которые начинаются со знака **@**, называются *переменные экземпляра*, они автоматически доступны в представлении; в данном случае, представление **index.html.erb** перебирает список **@users** и выводит строку HTML для каждого. (Помните, вы не должны пытаться понять этот код прямо сейчас. Он показан только для иллюстрации.)

Листинг 2.7: Представление index для пользователей.
app/views/users/index.html.erb

```
<h1>Listing users</h1>

<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Email</th>
      <th colspan="3"></th>
    </tr>
  </thead>

<% @users.each do |user| %>
  <tr>
    <td><%= user.name %></td>
    <td><%= user.email %></td>
    <td><%= link_to 'Show', user %></td>
    <td><%= link_to 'Edit', edit_user_path(user) %></td>
    <td><%= link_to 'Destroy', user, method: :delete,
                                     data: { confirm: 'Are you sure?' } %></td>
  </tr>
<% end %>
</table>

<br>

<%= link_to 'New User', new_user_path %>
```

Представление преобразует свое содержимое в HTML (Шаг 7), который затем возвращается контроллером в браузер для отображения (Шаг 8).

2.2.3. Недостатки данного Users-ресурса

Несмотря на полезность для общего обзора Rails, scaffold-генерированный ресурс Users имеет много серьезных недостатков.

- **Нет валидации данных.** Наша модель User безропотно принимает такие данные, как пустые имена и недопустимые адреса электронной почты.
- **Нет аутентификации (подтверждения подлинности.** У нас нет понятия регистрации, и нет способа воспрепятствовать тому, чтобы любой пользователь выполнил любую операцию.
- **Нет тестов.** Что, технически, не вполне верно, --- scaffolding включает рудиментарные тесты --- но в них нет тестов для валидации данных, аутентификации, или каких-либо других пользовательских потребностей.
- **Нет стилей или макета.** Нет сколько-нибудь связанного стиля сайта или навигации.
- **Нет реального понимания.** Если вы понимаете scaffold-код, вы, вероятно, не стали бы читать эту книгу.

2.3. Ресурс Microposts

Сгенерировав и исследовав ресурс Users, мы обратимся теперь к связанному с ним ресурсу Microposts. На протяжении этого раздела я рекомендую сравнивать между собой элементы этих ресурсов; вы должны увидеть, что они во многом параллельны друг другу. RESTful структура Rails-приложений лучше всего усваивается посредством наблюдения за подобными повторяющимися формами --- действительно, наблюдение параллельности структур Users и Microposts, даже на этой ранней стадии, является одной из причин написания этой главы.

2.3.1. Микрообзор микрособщений

Как и с ресурсом Users, сгенерируем код scaffold для ресурса Microposts, используя **`rails generate scaffold`**, в этом случае реализуя модель данных из Рисунка 2.3:[8]

```
$ rails generate scaffold Micropost content:text user_id:integer
      invoke  active_record
      create    db/migrate/20140821012832_create_microposts.rb
      create    app/models/micropost.rb
      invoke    test_unit
      create      test/models/micropost_test.rb
      create      test/fixtures/microposts.yml
      invoke  resource_route
       route    resources :microposts
      invoke  scaffold_controller
      create    app/controllers/microposts_controller.rb
      invoke    erb
      create      app/views/microposts
      create      app/views/microposts/index.html.erb
      create      app/views/microposts/edit.html.erb
```

[8] Как и в случае с User scaffold, генератор scaffold для микрособщений следует соглашению для именования в единственном числе для моделей Rails; таким образом, мы получим **`generate Micropost`**.

```
      create      app/views/microposts/show.html.erb
      create      app/views/microposts/new.html.erb
      create      app/views/microposts/_form.html.erb
      invoke    test_unit
      create      test/controllers/microposts_controller_test.rb
      invoke    helper
      create      app/helpers/microposts_helper.rb
      invoke      test_unit
      create        test/helpers/microposts_helper_test.rb
      invoke    jbuilder
      create      app/views/microposts/index.json.jbuilder
      create      app/views/microposts/show.json.jbuilder
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/microposts.js.coffee
      invoke    scss
      create      app/assets/stylesheets/microposts.css.scss
      invoke  scss
   identical    app/assets/stylesheets/scaffolds.css.scss
```

(Если вы получите ошибку, связанную со Spring, просто запустите команду снова.) Чтобы обновить базу данных новой моделью данных, мы должны запустить миграцию, как в Разделе 2.2:

```
$ bundle exec rake db:migrate
==  CreateMicroposts: migrating ===============================================
-- create_table(:microposts)
   -> 0.0023s
==  CreateMicroposts: migrated (0.0026s) ======================================
```

Теперь мы можем создавать микросообщения так же, как создавали пользователей в Разделе 2.2.1. Как вы можете догадаться, генератор scaffold добавил в файл маршрутов Rails правило для ресурса Microposts, Листинг 2.8.[9] Так же, как и для пользователей, маршрутное правило **`resources :microposts`** направляет URL'ы микросообщений к действиям контроллера Microposts, как видно в Таблице 2.3.

Листинг 2.8: Маршруты Rails с новым правилом для ресурса Microposts.
config/routes.rb

```
Rails.application.routes.draw do
  resources :microposts
  resources :users
  .
  .
  .
end
```

Сам Microposts-контроллер в схематичной форме представлен в Листинге 2.9. Обратите внимание, он полностью *идентичен* Листингу 2.4, кроме **`MicropostsController`** вместо **`UsersController`**. Это отражение REST-архитектуры, характерной для обоих ресурсов.

[9] Код scaffold мог создать дополнительные символы новой строки по сравнению с Листингом 2.8. Это не повод для беспокойства, поскольку Ruby игнорирует их.

HTTP-запрос	URL	Действие	Предназначение
GET	/microposts	**index**	страница со списком микрособщений
GET	/microposts/1	**show**	страница для отображения микрособщения с id **1**
GET	/microposts/new	**new**	страница для создания нового микрособщения
POST	/microposts	**create**	создает новое микрособщение
GET	/microposts/1/edit	**edit**	страница для редактирования микрособщения с id **1**
PATCH	/microposts/1	**update**	обновляет данные микрособщения с id **1**
DELETE	/microposts/1	**destroy**	удаляет микрособщение с id **1**

Таблица 2.3: RESTful маршруты, обеспеченные ресурсом Microposts в Листинге 2.8.

Листинг 2.9: Контроллер Microposts в схематичной форме.
app/controllers/microposts_controller.rb

```
class MicropostsController < ApplicationController
  .
  .
  .
  def index
    .
    .
    .
  end

  def show
    .
    .
    .
  end

  def new
    .
    .
    .
  end

  def edit
    .
    .
    .
  end

  def create
    .
    .
    .
  end

  def update
    .
    .
    .
  end

  def destroy
    .
    .
```

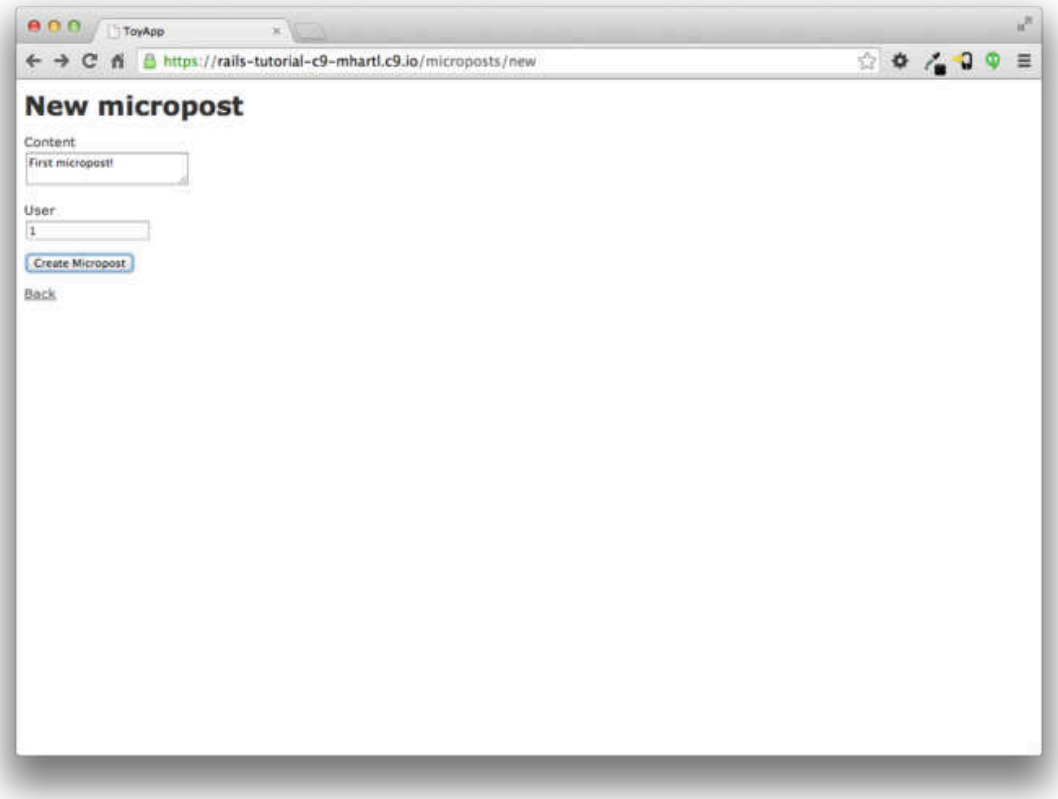


Рис. 2.12: Страница для создания микрособщения (/microposts/new).

```
    .
  end
end
```

Для того, чтобы сделать несколько настоящих микрособщений, мы вводим информацию на странице нового микрособщения, /microposts/new, см. Рисунок 2.12.

Теперь создайте одно-два микрособщения, проверив, чтобы по крайней мере у одного из них был `user_id 1` для сопоставления с id первого пользователя, созданного в Разделе 2.2.1. Результат должен быть похож на Рисунок 2.13.

2.3.2. Помещение *микро* в микрособщения

Каждое *микро*сообщение, достойное своего имени, должно иметь средства предписания длины сообщения. Реализация этого ограничения в Rails легко достигается с помощью *валидаций (проверок допустимости)*; чтобы принять микрособщения длиной не более 140 символов (а-ля Twitter), мы используем валидацию

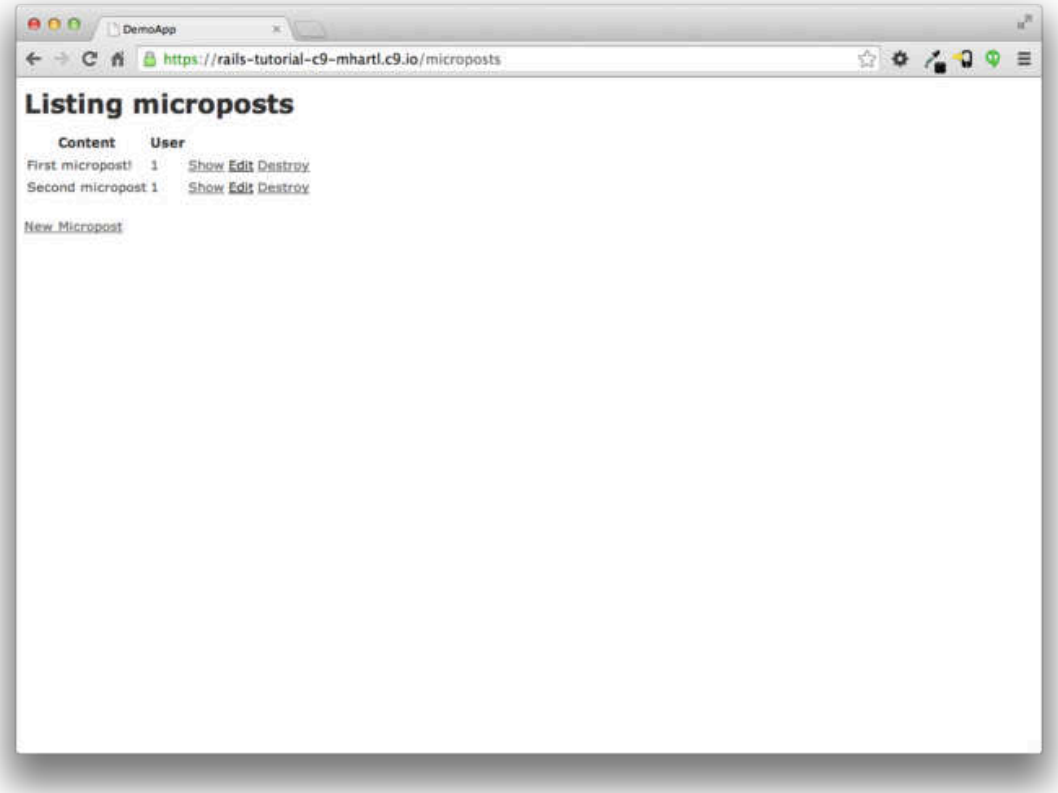


Рис. 2.13: Страница-список микрособщений (/microposts).

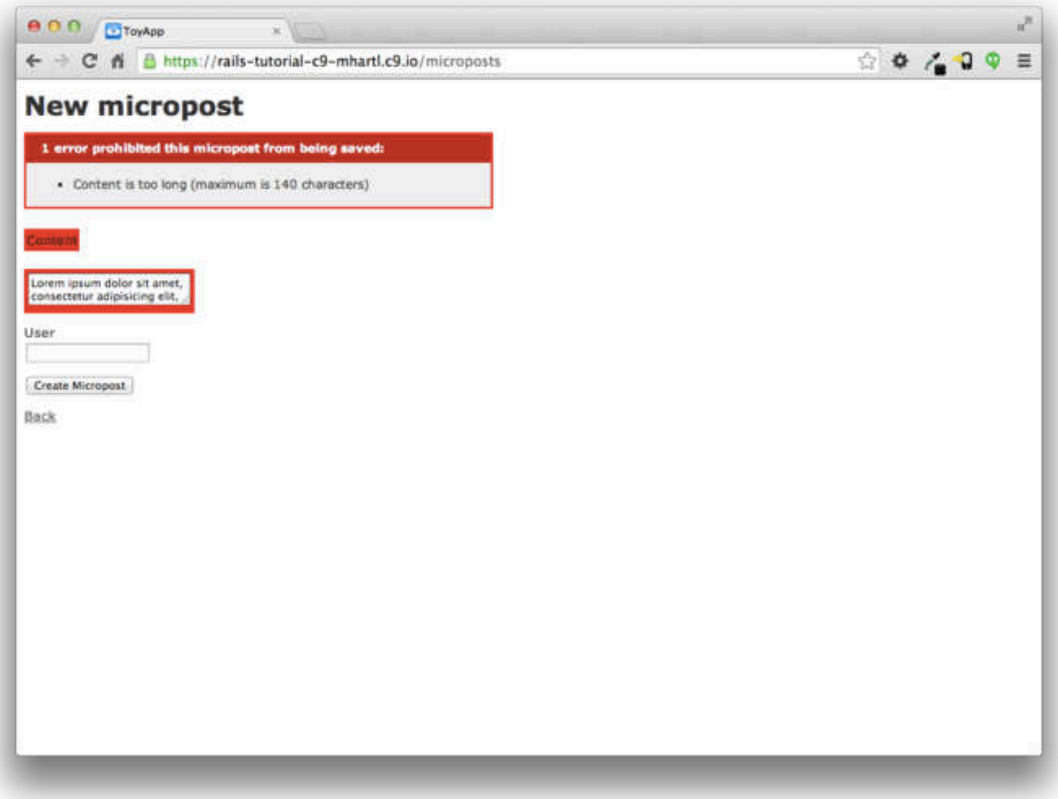


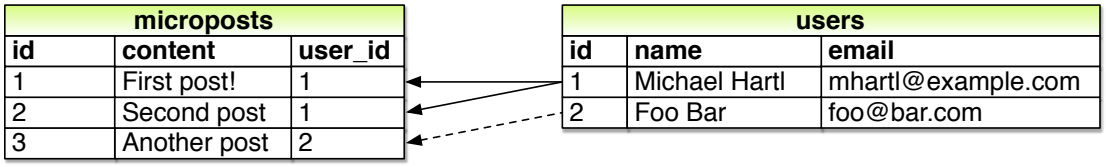
Рис. 2.14: Сообщение об ошибке для неудачно созданного микрособщения.

длины. Теперь нужно открыть файл **`app/models/micropost.rb`** в текстовом редакторе или IDE и заполнить его содержимым Листинга 2.10.

Листинг 2.10: Ограничение длины микрособщений максимумом в 140 знаков.
`app/models/micropost.rb`

```
class Micropost < ActiveRecord::Base
  validates :content, length: { maximum: 140 }
end
```

Код в Листинге 2.10 может выглядеть довольно таинственно --- мы рассмотрим валидацию более тщательно в Разделе 6.2 --- но его эффект станет вполне очевиден, если мы перейдем на страницу создания микрособщений и введем больше 140 символов в поле. Как видно на Рисунке 2.14, Rails выдаёт *сообщение об ошибке*, указывающее на то, что содержимое микрособщения является слишком длинным. (Мы узнаем больше о сообщениях об ошибках в Разделе 7.3.3.)



microposts		
id	content	user_id
1	First post!	1
2	Second post	1
3	Another post	2

users		
id	name	email
1	Michael Hartl	mhartl@example.com
2	Foo Bar	foo@bar.com

Рис. 2.15: Связь между микрособщениями и пользователями.

2.3.3. У пользователя `has_many` (много) микрособщений

Одна из наиболее мощных функций Rails --- это возможность формировать *связи* между различными моделями данных. В случае нашей модели User, у каждого пользователя потенциально есть много микрособщений. Мы можем выразить это в коде, обновив модели User и Micropost, как это показано в Листинге 2.11 и Листинге 2.12.

Листинг 2.11: У пользователя много микрособщений (user has many microposts).
app/models/user.rb

```
class User < ActiveRecord::Base
  has_many :microposts
end
```

Листинг 2.12: Микрособщения принадлежат пользователю (micropost belongs to user).
app/models/micropost.rb

```
class Micropost < ActiveRecord::Base
  belongs_to :user
  validates :content, length: { maximum: 140 }
end
```

Результат применения этой связи виден на Рисунке 2.15. За счёт столбца **`user_id`** в таблице **`microposts`**, Rails (через Active Record) может вывести микрособщения, связанные с каждым пользователем.

В Главе 11 и Главе 12 мы будем использовать такую же связь для того, чтобы вывести на экран все микрособщения пользователя, и для того, чтобы создать Twitter-подобную ленту микрособщений. А пока мы можем выяснить смысл связи ``микрособщения-пользователь" с помощью *консоли*, которая является полезным инструментом для взаимодействия с Rails-приложениями. Сначала мы вызовем консоль командой **`rails console`** в командной строке, а затем вытянем первого пользователя из базы данных, используя **`User.first`** (поместив результаты в переменную **`first_user`**):[10]

[10]Ваше приглашение командной строки, вероятно, будет чем-то вроде `2.1.1 :001 >`, но я буду использовать >>, поскольку версии Ruby могут отличаться.

```
$ rails console
>> first_user = User.first
=> #<User id: 1, name: "Michael Hartl", email: "michael@example.org",
created_at: "2014-07-21 02:01:31", updated_at: "2014-07-21 02:01:31">
>> first_user.microposts
=> [#<Micropost id: 1, content: "First micropost!", user_id: 1, created_at:
"2014-07-21 02:37:37", updated_at: "2014-07-21 02:37:37">, #<Micropost id: 2,
content: "Second micropost", user_id: 1, created_at: "2014-07-21 02:38:54",
updated_at: "2014-07-21 02:38:54">]
>> micropost = first_user.microposts.first    # Micropost.first так же сработает.
=> #<Micropost id: 1, content: "First micropost!", user_id: 1, created_at:
"2014-07-21 02:37:37", updated_at: "2014-07-21 02:37:37">
>> micropost.user
=> #<User id: 1, name: "Michael Hartl", email: "michael@example.org",
created_at: "2014-07-21 02:01:31", updated_at: "2014-07-21 02:01:31">
>> exit
```

(Я оставил **`exit`** в последней строке только чтобы продемонстрировать, как выйти из консоли. На большинстве систем вы можете использовать Ctrl-D с этой же целью.)[11] Таким образом, мы получили доступ к микросообщениям пользователя, используя код **`first_user.microposts`**. Благодаря ему, Active Record автоматически возвращает все микросообщения с **`user_id`** равным id **`first_user`** (в данном случае, **`1`**). Мы узнаем намного больше о возможностях ассоциаций (связей) Active Record в Главе 11 и Главе 12.

2.3.4. Иерархия наследования

Мы закончим обсуждение мини-приложения кратким описанием иерархии классов контроллеров и моделей в Rails. В этом будет смысл, только если у вас уже был некоторый опыт объектно-ориентированного программирования (ООП); если вы не изучали ООП, не стесняйтесь пропустить этот раздел. В частности, если вы не знакомы с *классами* (обсуждаемыми в Разделе 4.4), я предлагаю вернуться к этому разделу позже.

Мы начнем со структуры наследования для моделей. Сравнивая Листинг 2.13 и Листинг 2.14, мы видим, что и модель User, и модель Micropost наследуются (через левую угловую скобку **`<`**) от **`ActiveRecord::Base`**, который является базовым классом для моделей, предоставляемым библиотекой ActiveRecord; схема, резюмирующая это отношение, представлена на Рисунке 2.16. Благодаря наследованию от **`ActiveRecord::Base`**, наши объекты модели получают возможность связываться с базой данных, обрабатывать столбцы базы данных как Ruby-атрибуты, и так далее.

Листинг 2.13: Класс **`User`**, с наследованием.
`app/models/user.rb`

```
class User < ActiveRecord::Base
  .
  .
  .
end
```

[11] Как и в случае с ``Ctrl-C", заглавная ``D" означает клавишу на клавиатуре, а не заглавную букву, поэтому не нужно зажимать кнопку Shift вместе с Ctrl.

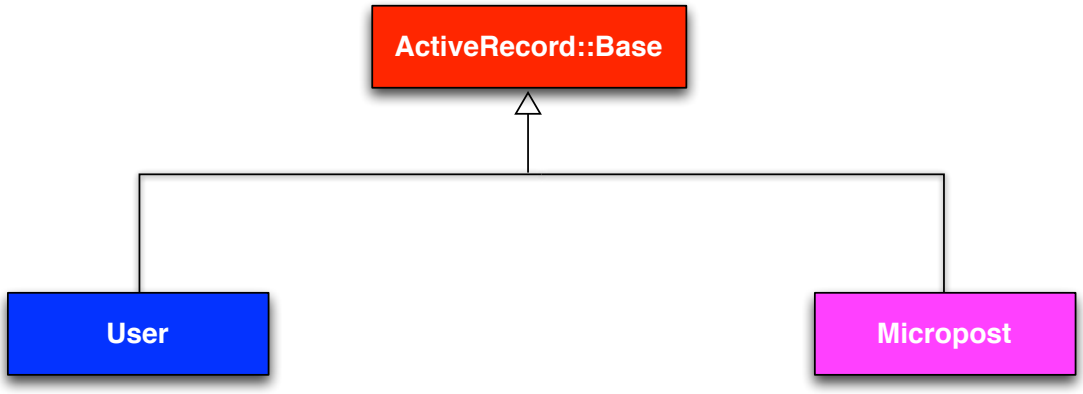


Рис. 2.16: Иерархия наследования для моделей User и Micropost.

Листинг 2.14: Класс `Micropost`, с наследованием.
app/models/micropost.rb

```
class Micropost < ActiveRecord::Base
  .
  .
  .
end
```

Структура наследования для контроллеров лишь немногим более сложная. Сравнивая Листинг 2.15 с Листингом 2.16, мы видим, что и контроллер User, и контроллер Micropost наследуются от Application controller. Исследуя Листинг 2.17, видим, что сам `ApplicationController` наследуется от `ActionController::Base`; это базовый класс для контроллеров, обеспеченных Rails-библиотекой Action Pack. Отношения между этими классами показаны на Рисунке 2.17.

Листинг 2.15: Класс `UsersController`, с наследованием.
app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  .
  .
  .
end
```

Листинг 2.16: Класс `MicropostsController`, с наследованием.
app/controllers/microposts_controller.rb

```
class MicropostsController < ApplicationController
  .
```

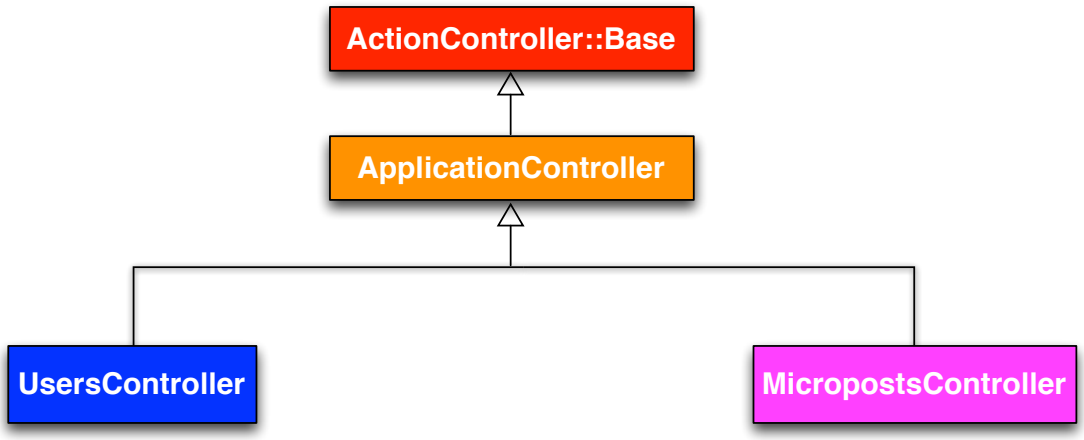


Рис. 2.17: Иерархия наследования для контроллеров Users и Microposts.

```
  .
  .
end
```

Листинг 2.17: Класс `ApplicationController`, с наследованием.
app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  .
  .
  .
end
```

Как и при наследовании моделей, оба контроллера (Users и Microposts) получают большое количество функций за счёт наследования от базового класса (в данном случае, `ActionController::Base`), включая возможность управлять объектами моделей, фильтровать входящие запросы HTTP и визуализировать представления в качестве HTML. Так как все контроллеры Rails наследуются от `ApplicationController`, правила, определенные в нем, автоматически применяются к каждому действию в приложении. Например, в Разделе 8.4 мы увидим, как можно добавить хелперы для осуществления входа и выхода во все контроллеры учебного приложения.

2.3.5. Развёртывание мини-приложения

Закончив ресурс Microposts, мы можем отправить репозиторий на Bitbucket: