Ruby on Rails

Урок 6

Авторизация

Аутентификация и авторизация. Гем Devise. Регистрация пользователей. Локализация. OAuth 2.0.

Аутентификация и авторизация

Аутентификация − это процедура, которая определяет, является ли пользователь тем, за кого себя выдает. Для нее используется проверка фактов, которые известны исключительно пользователю: пара логин-пароль, уникальный код, отправляемый на сотовый телефон, и аккаунт социальной сети.

Авторизация − это процедура, которая определяет, имеет ли пользователь право на выполнение тех или иных действий. Разные пользователи имеют разные права в приложении. Например, в разрабатываемом нами блоге пользователи, помеченные флагом creator, имеют право создавать сообщения. А пользователи, отмеченные флагом moderator, могут управлять сообщениями других пользователей: скрывать, редактировать и удалять.

Гем Devise

Для реализации аутентификации используется несколько гемов. Наиболее популярный из них − Devise. Он содержит 10 модулей:

1. **Database Authenticatable** шифрует и хранит пароль в базе данных для аутентификации пользователя. Вход может быть осуществлен как через POST-запрос, так и с помощью HTTP Basic Authentication.
2. **Omniathable** добавляет поддержку OAuth.
3. **Confirmable** отправляет письмо с инструкцией для подтверждения регистрации на только что зарегистрированный почтовый ящик.
4. **Recoverable** отправляет письмо на почтовый ящик для восстановления забытого пароля.
5. **Registrable** позволяет пользователю редактировать и удалять свой аккаунт.
6. **Rememberable** позволяет запоминать пользователей с помощью cookies. Управляет созданием и удалением токенов.
7. **Trackable** сохраняет информацию о количестве и времени входов пользователя, а также об IP-адресе.
8. **Timeoutable** отвечает за продолжительность сессии.
9. **Validatable** предоставляет инструменты для валидации e-mail и пароля.
10. **Lockable** блокирует аккаунт после определенного количества неудачных попыток аутентификации и отправляет письмо на почту с инструкцией по разблокировке.

Добавим поддержку этого гема в наш проект.

Установка

Добавим в Gemfile объявление гема devise и установим этот гем с помощью bundle install.

Gemfile

```
...
gem 'devise'
...
```

Если вызовем команду rails generate без параметров, обнаружим, что в списке генераторов появилась новая секция гема Devise.

```
rails generate
```

```
…
Devise:
  devise
  devise:controllers
  devise:install
  devise:views
…
```

Чтобы добавить в проект поддержку гема, необходимо выполнить команду **rails generate devise:install**. Команда создаст два файла и выдаст инструкции по дальнейшей настройке гема.

```
rails generate devise:install
 create  config/initializers/devise.rb
 create  config/locales/devise.en.yml
```

Файл config/initializers/devise.rb содержит настройки гема Devise, например: почтовый адрес, с которого будут отправляться сообщения; минимальную и максимальную длину пароля; регулярное выражение для электронной почты и т.д.

config/initializers/devise.rb

```
Devise.setup do |config|
  ...
  config.mailer_sender = 'info@geekbrains.ru'
  config.password_length = 6..128
  config.email_regexp = /\A[^@\s]+@[^@\s]+\z/
  ...
end
```

Файл config/locales/devise.en.yml содержит все фразы для английской локали, которые используются Devise. Создав аналогичный файл для русской локали config/locales/devise.en.yml, можно перевести все сообщения гема Devise на русский язык. Готовый перевод здесь.

Одна из необходимых настроек гема – установка URL по умолчанию. Для development-окружения добавим в файл `config/environments/development.rb` значение config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }. Для production-окружения нужно указать действительный url приложения.

config/environments/development.rb

```
Rails.application.configure do
  ...
  config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
end
```

Привязка Devise к модели User

Теперь необходимо сконфигурировать модель и базу данных для поддержки Devise. У нас уже есть таблица для хранения пользователей. Используем ее и добавим необходимые для гема Devise поля

при помощи команды **rails generate devise User**. Если бы модели User не было, команда создала бы новую модель.

```
rails generate devise User
  invoke  active_record
  create    db/migrate/20180114160350_add_devise_to_users.rb
  insert    app/models/user.rb
   route  devise_for :users
```

В файле config/routes.rb добавилась строка devise: users. Посмотреть все доступные роуты можно командой rails routes, отфильтровав роуты Devise при помощи флага -g:

```
rails routes -g devise
...
             Prefix Verb   URI Pattern                    Controller#Action
    new_user_session GET /users/sign_in(.:format)       devise/sessions#new
        user_session POST   /users/sign_in(.:format)    devise/sessions#create
destroy_user_session DELETE /users/sign_out(.:format)   devise/sessions#destroy
   new_user_password GET /users/password/new(.:format)  devise/passwords#new
  edit_user_password GET /users/password/edit(.:format) devise/passwords#edit
       user_password PATCH  /users/password(.:format)   devise/passwords#update
                     PUT /users/password(.:format)      devise/passwords#update
                     POST   /users/password(.:format)   devise/passwords#create
cancel_user_registration GET   /users/cancel(.:format)
devise/registrations#cancel
   new_user_registration GET   /users/sign_up(.:format)
devise/registrations#new
  edit_user_registration GET   /users/edit(.:format)
devise/registrations#edit
      user_registration PATCH  /users(.:format)
devise/registrations#update
                        PUT    /users(.:format)
devise/registrations#update
                        DELETE /users(.:format)
devise/registrations#destroy
                        POST   /users(.:format)
devise/registrations#create
```

В файле app/models/user.rb добавились строки, настраивающие гем Devise:

app/models/user.rb

```
class User < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
      :recoverable, :rememberable, :trackable, :validatable
  ...
end
```

С помощью метода devise можно добавлять или отключать модули гема, перечисленные ранее. Подключенные модули будут переданы в качестве аргументов метода devise, отключенные – перечислены в комментарии перед ним.

Также будет создана миграция, которая внесет в таблицу users дополнительные поля.

db/migrate/<timestamp>_add_devise_to_users.rb

```
class AddDeviseToUsers < ActiveRecord::Migration[5.1]
  def self.up
    change_table :users do |t|
      ## Database authenticatable
      t.string :email,              null: false, default: ""
      t.string :encrypted_password, null: false, default: ""

      ## Recoverable
      t.string   :reset_password_token
      t.datetime :reset_password_sent_at

      ## Rememberable
      t.datetime :remember_created_at

      ## Trackable
      t.integer  :sign_in_count, default: 0, null: false
      t.datetime :current_sign_in_at
      t.datetime :last_sign_in_at
      t.inet     :current_sign_in_ip
      t.inet     :last_sign_in_ip

      ## Confirmable
      # t.string   :confirmation_token
      # t.datetime :confirmed_at
      # t.datetime :confirmation_sent_at
      # t.string   :unconfirmed_email # Only if using reconfirmable

      ## Lockable
      # t.integer  :failed_attempts, default: 0, null: false # Only if lock
strategy is :failed_attempts
      # t.string   :unlock_token # Only if unlock strategy is :email or :both
      # t.datetime :locked_at

      # Uncomment below if timestamps were not included in your original
model.
      # t.timestamps null: false
    end

    add_index :users, :email,              unique: true
    add_index :users, :reset_password_token, unique: true
    # add_index :users, :confirmation_token,   unique: true
    # add_index :users, :unlock_token,         unique: true
  end

  def self.down
```

```
    # By default, we don't want to make any assumption about how to roll back
a migration when your
    # model already existed. Please edit below which fields you would like to
remove in this migration.
    raise ActiveRecord::IrreversibleMigration
  end
end
```

Перед каждым набором новых полей указывается название модуля, которому эти поля нужны. Если подключается модуль, важно, чтобы соответствующие поля в миграции были раскомментированы. Если модуль отключается, лишние поля можно закомментировать.

Удалим из миграции ненужные поля. В конечном счете миграция будет иметь следующий вид:

db/migrate/<timestamp>_add_devise_to_users.rb

```
class AddDeviseToUsers < ActiveRecord::Migration[5.1]
  def self.up
      change_table :users do |t|
      ## Database authenticatable
      t.string :encrypted_password, null: false, default: ""
      ## Recoverable
      t.string   :reset_password_token
      t.datetime :reset_password_sent_at
      ## Rememberable
      t.datetime :remember_created_at
      ## Trackable
      t.integer  :sign_in_count, default: 0, null: false
      t.datetime :current_sign_in_at
      t.datetime :last_sign_in_at
      t.inet      :current_sign_in_ip
      t.inet      :last_sign_in_ip
    end
  end

  def self.down
    raise ActiveRecord::IrreversibleMigration
  end
end
```

Теперь можно применить миграцию к базе данных, выполнив команду **rails db:migrate**.

В таблице появилось новое поле password. Для его корректного заполнения необходимо изменить сиды. Чтобы упростить задачу аутентификации, назначим в качестве начального пароля электронный адрес пользователя.

db/seeds.rb

```
...
hash_users = 10.times.map do
  email = FFaker::Internet.safe_email
```

```
  {
    name: FFaker::Internet.user_name[0...16],
    email: email,
    password: email
  }
end
users = User.create! hash_users
users.first(7).each { |u| u.update creator: true }
users.first(2).each { |u| u.update moderator: true }
...
```

Хелперы Devise

Devise предоставляет хелпер-методы, которые облегчают работу с данными пользователя:

- **user_signed_in?** возвращает булевое значение, определяющее аутентифицирован ли текущий пользователь.
- **current_user** возвращает экземпляр модели User для текущего пользователя.
- **user_session** предоставляет доступ к сессии пользователя.

Слово «user» в названии хелпера взято из названия модели. Если бы модель, к которой подключается Devise, называлась Employee, хелперы носили бы имена: employee_signed_id?, current_employee, employee_session.

Создадим меню в шапке сайта, а также добавим приветствие и ссылку на завершение пользовательской сессии для аутентифицированного пользователя. Если пользователь не аутентифицирован − предоставим ссылки на регистрацию и панель аутентификации.

Создадим partial следующего содержания, который разместим в общей папке shared:

app/views/shared/_header.html.slim

```
header
  ul
    - if user_signed_in?
      li
        span Hello, #{current_user.email}
      li
        = link_to 'Log out', destroy_user_session_path, method: :delete
    - else
      li
        = link_to 'Log in', new_user_session_path
      li
        = link_to 'Register', new_user_registration_path
```

В лейауте views/layouts/application.html.slim подключим partial-файл _header.html.slim при помощи метода render.

app/views/layouts/application.html.slim

```
doctype html
```

```
html
  head
    title Blog
    = csrf_meta_tags
    = stylesheet_link_tag      'application', media: 'all'
    = javascript_include_tag 'application'
  body
    = render 'shared/header'
    = yield
```

Переопределение представлений Devise

Если сейчас попробовать на странице регистрации создать нового пользователя, попытка будет безуспешной. Причина в том, что в модели пользователя определено поле с его именем, а в форме, сгенерированной Devise, имени нет. Чтобы получить возможность модифицировать код представлений, следует переопределить их в приложении. Для этого следует воспользоваться генератором **rails generate devise:views**.

```
rails generate devise:views
      invoke  Devise::Generators::SharedViewsGenerator
      create  app/views/devise/shared
      create  app/views/devise/shared/_links.html.erb
      invoke  form_for
      create  app/views/devise/confirmations
      create  app/views/devise/confirmations/new.html.erb
      create  app/views/devise/passwords
      create  app/views/devise/passwords/edit.html.erb
      create  app/views/devise/passwords/new.html.erb
      create  app/views/devise/registrations
      create  app/views/devise/registrations/edit.html.erb
      create  app/views/devise/registrations/new.html.erb
      create  app/views/devise/sessions
      create  app/views/devise/sessions/new.html.erb
      create  app/views/devise/unlocks
      create  app/views/devise/unlocks/new.html.erb
      invoke  erb
      create  app/views/devise/mailer
      create  app/views/devise/mailer/confirmation_instructions.html.erb
      create  app/views/devise/mailer/email_changed.html.erb
      create  app/views/devise/mailer/password_change.html.erb
      create  app/views/devise/mailer/reset_password_instructions.html.erb
      create  app/views/devise/mailer/unlock_instructions.html.erb
```

К сожалению, Devise не создает представлений в slim-формате, поэтому воспользуемся ранее установленным гемом html2slim для преобразования erb-представлений в slim.

```
erb2slim . -d
```

Добавим имя пользователя в форму регистрации. Для этого изменим файл app/views/devise/registrations/new.html.slim.

app/views/devise/registrations/new.html.slim

```
...
= form_for(resource, as: resource_name, url: registration_path(resource_name))
do |f|
  = devise_error_messages!
  .field
    = f.label :name
    br
    = f.text_field :name, autofocus: true
  .field
    = f.label :email
    br
    = f.email_field :email, autocomplete: "email"
...
```

Также необходимо изменить допустимые параметры для регистрации на стороне контроллера. Для этого в app/controllers/application_controller.rb нужно добавить следующие строки:

app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
  before_action :configure_permitted_parameters, if: :devise_controller?

  private

  def configure_permitted_parameters
    devise_parameter_sanitizer.permit(:sign_up, keys: [:name])
  end
end
```

Теперь новый пользователь сможет зарегистрироваться.

Требование аутентификации

Иногда требуется ограничить доступ пользователя к целым разделам сайта, например к личному кабинету или системе администрирования. В этом случае используется **authenticate_user!**. При помощи колбека before_action его можно подключить таким образом, чтобы он разрешал использовать экшены контроллера только аутентифицированным пользователям.

Например, чтобы ограничить доступ не аутентифицированным пользователям к экшенам контроллера PostsController, достаточно добавить в начало класса вызов метода **before_action :authenticate_user!**

app/controllers/posts_controller.rb

```
class PostsController < ApplicationController
  before_action :authenticate_user!
  ...
end
```

Теперь попытка получить доступ к страницам, которые обслуживаются контроллером PostsController, приведут к редиректу на страницу аутентификации.

Локализация

Ruby on Rails по умолчанию подключает в качестве зависимости гем I18n, который позволяет локализовать приложение, например, для использования русского языка. Перед его использованием рекомендуется установить русскую локаль в конфигурационном файле config/application.rb

app/config/application.rb

```
...
module Blog
  class Application < Rails::Application
    config.i18n.default_locale = :ru
    config.time_zone = 'Moscow'
    ...
  end
end
```

Возможно, что необходимы несколько локалей, например когда сайт должен поддерживать несколько языков: русский и английский. Тогда в файл app/config/application.rb следует добавить параметр config.i18n.available_locales со списком доступных локалей config.i18n.available_locales = [:en, :ru].

Файлы локализации сосредотачиваются в папке config/locales. В настоящий момент в нем должен находиться общий файл для английской локали en.yml и файл devise.en.yml для гема Devise. Чтобы создать локаль для русского языка следует создать два файла ru.yml и devise.ru.yml. Готовый перевод для гема Devise на русский язык здесь.

Переведем какую-нибудь фразу на русский язык. Форма входа views/devise/sessions/new.html.slim имеет заголовок ‘Log in’. Создадим в файле ru.yml секцию label с ключом log_in, которому назначим русский перевод «Вход».

app/config/locales/ru.yml

```
ru:
  label:
    log_in: Вход
```

Для перевода фраз используется метод **I18n.t**. Внутри представлений «I18n» можно опускать и просто использовать метод **t**.

app/views/devise/sessions/new.html.slim

```
h2 = t('label.log_in')
...
```

Название локали, en или ru, не указывается. После метода следует путь к ключу label.log_in. В зависимости от текущей локали подставляется фраза либо из файла ru.yml, либо из файла en.yml. Так как по умолчанию установлена русская консоль, фраза будет выведена на русском языке. Файл ru.yml выглядит следующим образом:

config/locales/ru.yml

```
ru:
  label:
    log_in: Вход
    log_out: Выход
    hello: Привет, %s
    registration: Регистрация
  activerecord:
    errors:
      messages:
        blank: обязательно для заполнения
        too_short: слишком коротко

        models:
        user:      Пользователи
        post:      Сообщения

        attributes:
          user:
            password:                Пароль
            password_confirmation: Подтверждение
            email:                   Email
            name:                    Имя пользователя
```

Окончательный вариант partial меню выглядит следующим образом:

app/views/shared/_header.html.slim

```
header
  ul
    - if user_signed_in?
      li
        span = t('label.hello') % current_user.name
      li
        = link_to t('label.logout'), destroy_user_session_path, method: :delete
    - else
      li
        = link_to t('label.login'), new_user_session_path
      li
        = link_to t('label.registration'), new_user_registration_path
```

Отладка отправки почтовых сообщений

Чтобы иметь возможность отлаживать процесс отправки почтовых сообщений, необходима эмуляция этого процесса без реальной отправки сообщений адресату. В этом поможет гем letter_opener, который открывает содержимое почтового отправления в новой вкладке браузера.

Гем следует подключить в development- и тестовом окружениях в файле Gemfile.

Gemfile

```
…
group :development, :test do
  ...
  gem 'letter_opener'
end
...
```

После этого отправка почтового сообщения приведет к тому, что содержимое письма будет открываться в браузере.

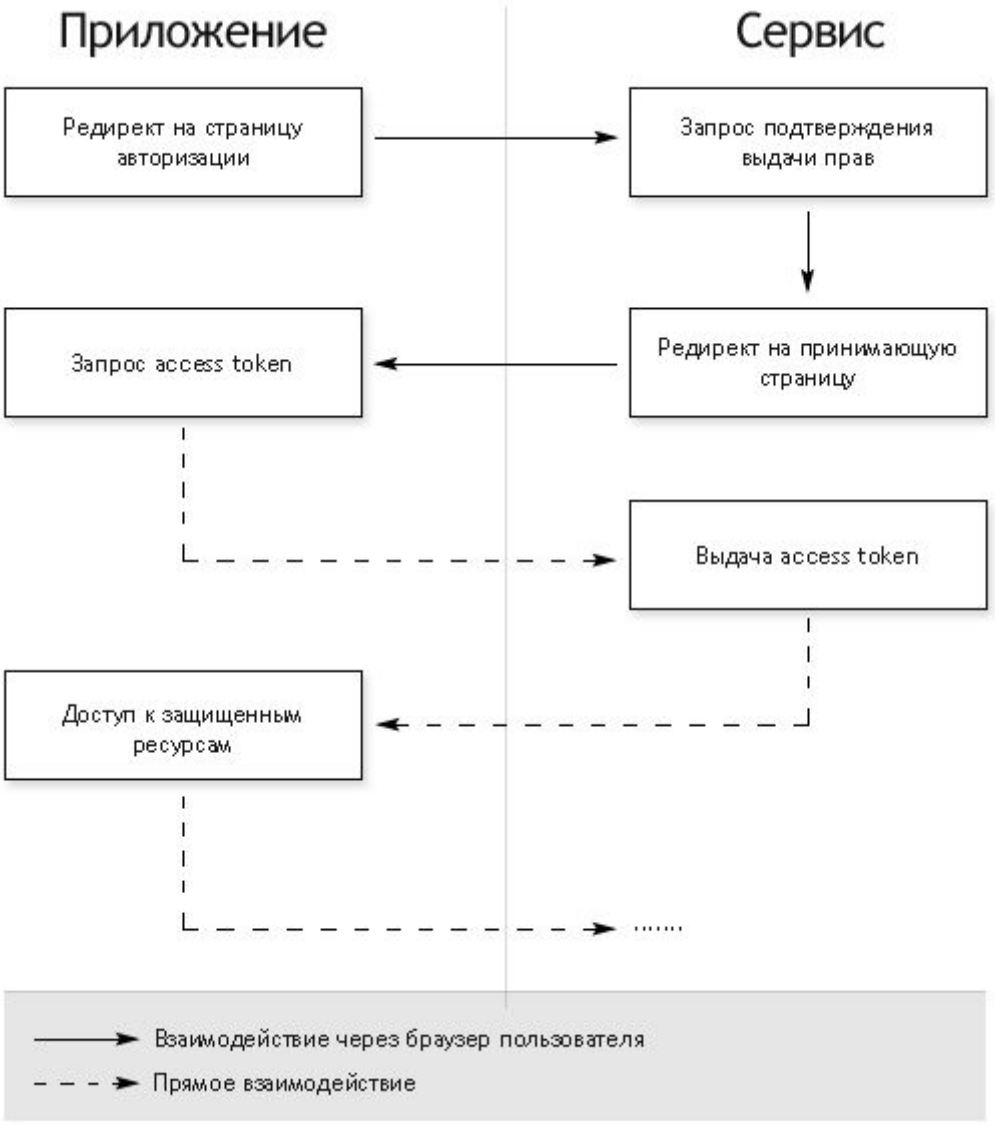
OAuth 2.0

Многие сайты позволяют регистрироваться при помощи сторонних сервисов, например, vk.com, google.com, github.com и т.п. Благодаря этому пользователям не приходится вводить в очередной раз одни и те же данные. Протокол, дающий возможность реализовать такую регистрацию, называется **OmniAuth (OAuth)**. Мы будем говорить об OAuth 2.0.

Oauth 2.0 – это протокол авторизации. Он позволяет выдать одному сервису права на доступ к ресурсам пользователя на другом сервисе. Протокол выдает ограниченный набор прав на ресурсы и избавляет пользователя от необходимости доверять свой логин и пароль стороннему сервису.

Алгоритм работает по следующей схеме:

1. Клиент видит ссылку, предлагающую ему пройти аутентификацию с помощью одного из сервисов, например google. Нажимая на нее, он отправляет запрос к нашему приложению на аутентификацию.
2. Наше приложение перенаправляет пользователя на страницу аутентификации google. Там клиент должен авторизоваться, если еще не авторизован, и дать согласие на доступ к некоторым персональным ресурсам, в которых «заинтересовано» наше приложение.
3. Получив согласия пользователя, google перенаправляет его обратно к нашему приложению на заранее указанный адрес, передав в запросе специальный ключ.
4. С помощью ключа наше приложение запрашивает от google токен для доступа к персональным данным и может работать с ними.


Приложение
Сервис
Редирект на страницу авторизации
Запрос подтверждения выдачи прав
Редирект на принимающую страницу
Запрос access token
Выдача access token
Доступ к защищенным ресурсам
.....
Взаимодействие через браузер пользователя
Прямое взаимодействие

Домашнее задание

1. Внести исправления в представления Devise так, чтобы они удовлетворяли вашему дизайну.
2. *Добавить пользователю возможность редактировать свой профиль.
3. *Добавить поддержу OAuth.

   «*» задание повышенной сложности (по желанию)

Дополнительные материалы

1. Devise.
2. Перевод гема Devise на русский язык.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Гем Devise.
2. Гем letter_opener.
3. Ричард Э. Смит. Аутентификация: от паролей до открытых ключей.
4. Майкл Хартл. Ruby on Rails для начинающих. Изучаем разработку веб-приложений на основе Rails.
5. Ruby on Rails Guides.
6. Obie Fernandez. The Rails 5 Way.
7. Локализация Ruby on Rails приложений.