# Создание REST API для внешних приложений

# Основные знания

- Ещё раз, что такое REST-подход
- Подходы к созданию REST API
- Стандарт представления JSON API
- Сериализаторы
- Версионирование
- Аутентификация
- Обеспечение безопасности
- CORS

# REST-подход

- Передача состояния представления (представление данных в удобном для клиента формате)
- Клиент-серверное взаимодействие
- Состояние клиента на сервере не сохраняется («здесь и сейчас», «не помню прошлое, не думаю о будущем»)
- Идентификация ресурсов (н-р URI)
- Ресурсы отделены от представления (HTML, JSON, XML)
- Ограниченное число методов (CRUD)
- Пример реализации — HTTP

# REST-подход

CRUD

URI

Клиент

Метод

Метод

Метод

Ресурс

Формат

Формат

# REST-подход


- не помню прошлое, не думаю о будущем

# Прикладной программный интерфейс (API)

# Подходы к созданию REST API

- API внутри
- API — отдельное приложение

# Стандарт представления JSON API

- MIME-тип — application/vnd.api+json
- Корень — ключ data
- Далее идёт массив ресурсных объектов
- Может помимо data встречаться ещё другие ключи, например, included
  - Included — ресурсные объекты, связанные с этим ресурсным объектом

# Ресурсный объект

- type — тип объекта (article, user и т. д.)
- id — идентификатор объекта.
- attributes — совокупность пар ключ-значение.
- relationships — совокупность объектов связей (relationship object), которая описывает связь между текущим ресурсом и другими ресурсами.

# Пример JSON API


{"data":[{"id":"1","type":"users","attributes":{"email":"test@mail.ru"},

"relationships":{"rental-units":{"data":[{"id":"1","type":"rental-units"},{"id":"2","type":"rental-units"}]}}},

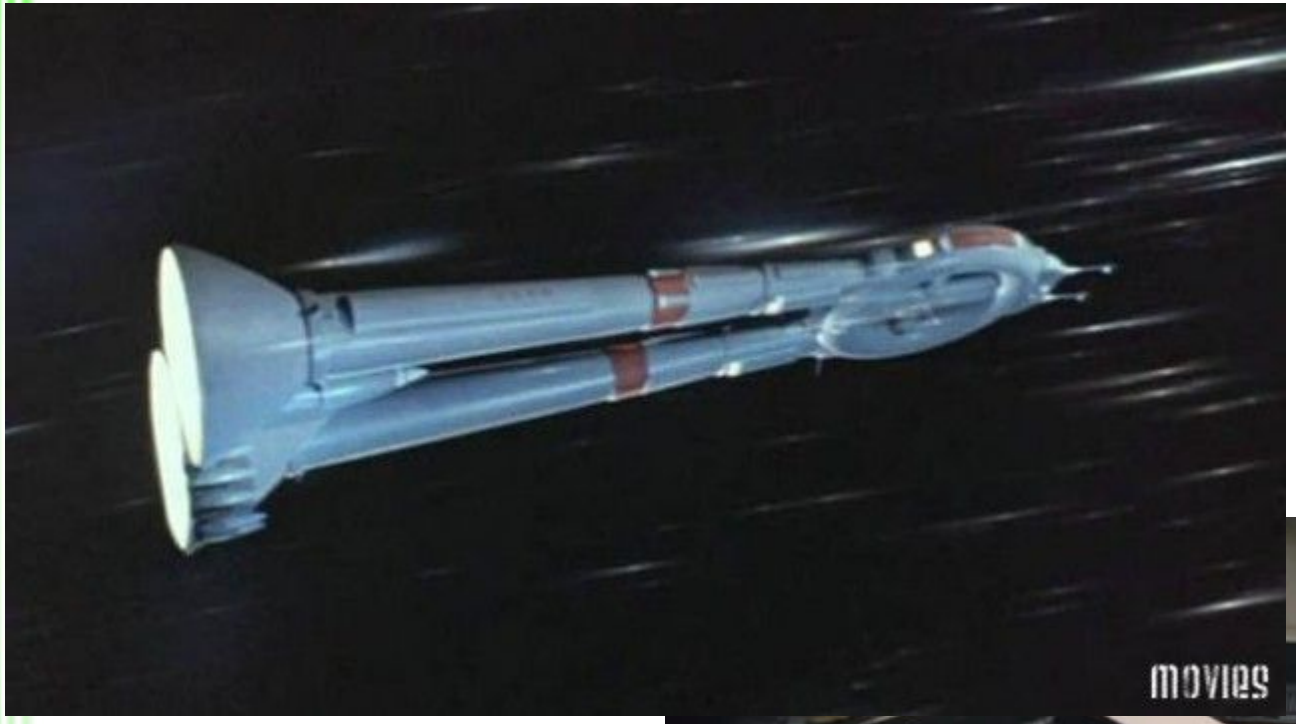
{"id":"2","type":"users","attributes":{"email":"user@mail.ru"},

"relationships":{"rental-units":{"data":[{"id":"3","type":"rental-units"},{"id":"4","type":"rental-units"}]}}}]}

# Сериализаторы

- Сериализаторы — объекты, реализующие сериализацию, т.е. представление информационного ресурса в различных форматах
- Gem
  - **fast_jsonapi**
  - Jbuilder
  - active_model_serializer
  - Jsonapi-rails
  - RABL
  -

# Сериализация




# Сериализация объекта


```
class OrderSerializer
  include FastJsonapi::ObjectSerializer
  attributes :title, :cost

  attribute :phone do |object|
    object.user.phone
  end


  belongs_to :user
  has_many :order_items


end
```

# Прописать маршрутизацию REST API


```
scope module: "api" do
  namespace "v1" do
    resources :competences, only: %I[index]
  end
  namespace "v2" do
    resources :competences, only: %I[index]
  end
end
```

# Версионирование контроллеров

```
|-- api
|   `-- v1
|       |-- api_controller.rb
|       `-- expeditions_controller.rb # наследуемся от api_controller.rb
|   `-- v2
|       |-- api_controller.rb
|       `-- expeditions_controller.rb
|   `-- api_controller.rb
|   `-- expeditions_controller.rb # модуль
|-- application_controller.rb
```


GET /v1/users

# Реализовать контроллер


```ruby
module Api::V1
  class ExpeditionsController < ApiController
    def index
      respond_to do |format|
        format.json do
          render json: ::V1::ExpeditionSerializer.new(Expedition.all).serialized_json
        end
      end
    end

    # ...
```

# Аутентификация

- По логину/паролю
- По токену
- Oauth
- JWT
- …

# Базовая аутентификация

- Кодируем логин:пароль в «Base64»
  - Base64 — представление двоичных данных в виде ASCII-символов
  - Base64.encode64 «login:password»
- Передаём полученную строку в HTTP-Заголовке
  - Authorization: Basic base64_string
- Для обработки используем метод authenticate_with_http_basic (из ActionController::HttpAuthentication::Basic::ControllerMethods)

# Реализовать базовую аутентификацию

```
class Api::ApiController < ApplicationController
  protect_from_forgery with: :null_session, if: Proc.new { |c| c.request.format.json? }
  skip_before_action :authenticate_user!
  before_action :auth_by_password

  include ActionController::HttpAuthentication::Basic::ControllerMethods

  private

  def auth_by_password
    result = authenticate_with_http_basic do |email, password|
      user = User.find_by(email: email)
      user&.valid_password?(password)
    end
    render json: { error: 'Неправильный логин либо пароль!' }, status: 403 unless result
  end

end
```

# Реализовать аутентификацию по токену


## Authorization: Token YYYY

```
include ActionController::HttpAuthentication::Token::ControllerMethods

private

def auth_by_token
  result = authenticate_with_http_token do |token|
    User.find_by(auth_token: token).present?
  end
  render json: { error: 'Неправильный токен!' }, status: 403 unless result
end
```

# Генерировать токен для пользователя


```
class User < ApplicationRecord

  before_create :generate_token
  private
    def generate_token
      self.token = Devise.friendly_token 8
    end
end
```

# Получать данные из тела запроса в контроллере


```
attr_reader :json
before_action :parse_request
private

  def parse_request
    @json = JSON.parse(request.body.read)
  end
end
```

# Обеспечение безопасности

- Защита от большого числа запросов
  - DDoS
  - Throttling
- Создание белых и чёрных списков пользователей по условиям
- Отслеживание запросов по условию
- Gem rack-attack
  - Rack middleware

# Реализовать простейшую защиту


```
# application.rb
config.middleware.use Rack::Attack

# initializers/rack_attack.rb
Rack::Attack.blocklist('block localhost') do |request|
  ['localhost', '127.0.0.1', '0.0.0.0'].include? request.ip
end
```

# Создать тест получения экспедиций через REST API


```
test 'should get index' do
  user = create(:cosmonaut)
  count = 10
  create_list(:expedition, count)
  get v1_expeditions_path(format: :json),
          headers: { 'Authorization' => "Token #{user.auth_token}" }
  assert_response :success
  assert_equal count, JSON.parse(response.body)["data"].length
end
```

# Создать тест проверки аутентификации


```
test 'should get forbidden' do
  get v1_expeditions_path(format: :json)
  assert_response :forbidden
end
```

# Заглянем под капот

# Базовая HTTP-аутентификация

```
def authenticate_with_http_basic(&login_procedure)
        HttpAuthentication::Basic.authenticate(request,
&login_procedure)
end

def authenticate(request, &login_procedure)
  if has_basic_credentials?(request)

login_procedure.call(*user_name_and_password(request)
)
  end
end

def has_basic_credentials?(request)
                request.authorization.present?          &&
(auth_scheme(request).downcase == "basic")
end
```

# Базовая HTTP-аутентификация

```
def authorization
  get_header("HTTP_AUTHORIZATION")   ||
  get_header("X-HTTP_AUTHORIZATION") ||
  get_header("X_HTTP_AUTHORIZATION") ||
  get_header("REDIRECT_X_HTTP_AUTHORIZATION")
End


def auth_scheme(request)
  request.authorization.to_s.split(" ", 2).first
end
```

# Алгоритм действий Rack::Attack

```
def call(env)
                              env['PATH_INFO']                    =
PathNormalizer.normalize_path(env['PATH_INFO'])
  req = Rack::Attack::Request.new(env)
  if safelisted?(req)
    @app.call(env)
  elsif blocklisted?(req)
    self.class.blocklisted_response.call(env)
  elsif throttled?(req)
    self.class.throttled_response.call(env)
  else
    tracked?(req)
    @app.call(env)
  end
end
```

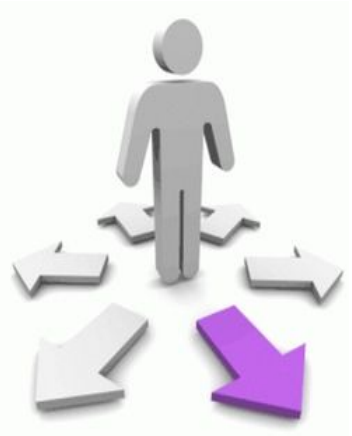

Ты молод, креативен, талантлив? Амбициозен, уверен в себе, полон свежих идей? А делать хоть что-нибудь умеешь?!
Atkritka.com

# Умения


- Версионировать контроллеры
- Версионировать сериализаторы
- Вывести информацию о экспедициях и экспедиции
- Аутентифицировать по токену
- Создать экспедицию
- Реализовать простейшую защиту с помощью чёрного списка
- Создать тесты для проверки работоспособности REST API

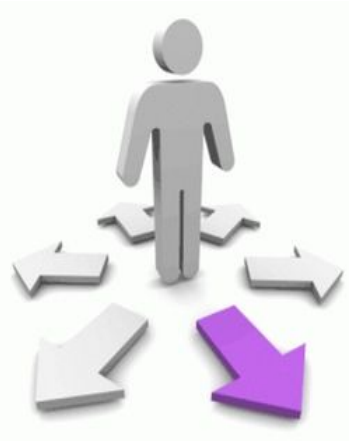

ТУДА
100м
СЮДА
200м
ВЕЧНАЯ НЕОПРЕДЕЛЕННОСТЬ!
КАК ЖИТЬ?
DEMOTIVATORS.RU

# Неопределённости



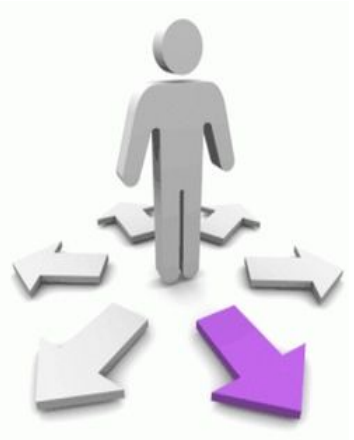
- Можно ли версионировать модель? Нет.
- Разница между included и relationship?
  - Relationships — просто идшники связанных объектов,
  - included — более подробная информация
- Есть ли готовый gem для аутентификации? Да, например, https://github.com/baschtl/devise-token_authenticatable

# Неопределённости



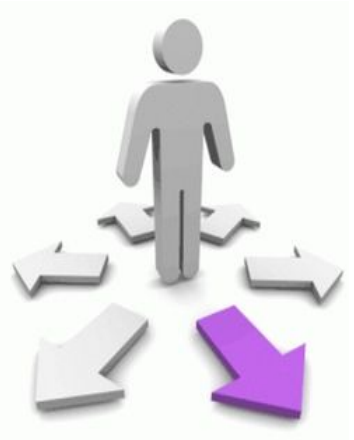
- Как избежать дублирования кода в контроллерах при версионировании?
  - Наследование
  - Вынести общий код в модуль
  - Использовать гемы, например, api-versions
  - Так надо :) (редко когда при изменении версии получается полное дублирование, также предыдущую версию мы можем позже удалить)
  - …

# Самостоятельно



- Базовая аутентификация
- Вложенные атрибуты (accepts_nested_attributes_for)
- Вложенные ресурсы
- JSONP
- CORS
- …

# Дополнительное чтени



- Альтернативы REST подходу — RPC, SOAP
- Аутентификация на основе JWT
- Вопросы безопасности для REST API
- …

# Меж-доменные запросы (CORS, Cross-Origin Resource Sharing)

- Улучшение JSONP
- Позволяет безопасным образом делать AJAX-запросы с одних доменов на другие
- Реализуются простые и сложные запросы

# Простой запрос

- Метод
  - HEAD
  - GET
  - POST
- Заголовки
  - Accept
  - Content-Type, но только со значениями:
    - application/x-www-form-urlencoded
    - multipart/form-data
    - text/plain
  - Движок добавить Origin

# Простой запрос-ответ

- POST /foo/bar HTTP/1.1
- Origin: http://friend.ru
- Host: profport.ru

- 200 OK HTTP/1.1
- Access-Control-Allow-Origin: http://friend.ru
- Content-Type: text/html; charset=utf-8

- <h1>Поехали!</h1>

# • Реализовать простой запрос


```
let xhr = new XMLHttpRequest();
xhr.open("GET",
"http://localhost:3000/v1/competences.json");
xhr.addEventListener('readystatechange', function() {
    if (xhr.readyState === 4 &&
xhr.status === 200)
        alert(xhr.responseText);
});
xhr.send(null);
```

```
config.middleware.insert_before
0, Rack::Cors do
  allow do
    origins '*'
    resource '*', :headers => :any,
:methods => [:get, :post,
:options]
  end
end
```

# Результат


Это же так просто!
Я это сделаю!
Я могу это сделать!
Я попытаюсь...
Как мне это сделать?
Я хочу это сделать!
Я не могу это сделать (не умею, не получится)
Я не буду этого делать.

# Результат

- Познакомились с REST API
- Научились версионировать REST API
- Аутентифицировали пользователя по токену
- Обеспечили простейшую защиту
- Создали тесты, чтобы удостовериться, что всё работает
- В итоге создали REST API для внешних приложений

# Список источников

- Основное
- Building JSON API with Rails 5
- Building the Perfect Rails 5 API Only App
- Дополнительное
- What is REST
- JSON API specification
- Что такое CORS