«Введение в REST»,

рассказанное *Михаилом Шохиревым, Sr.*
на собрании Клуба программистов «***Caiman***»
весной 2016 года в городе Шадринске
:-(а не на Карибских островах :-)

Что вы узнаете?

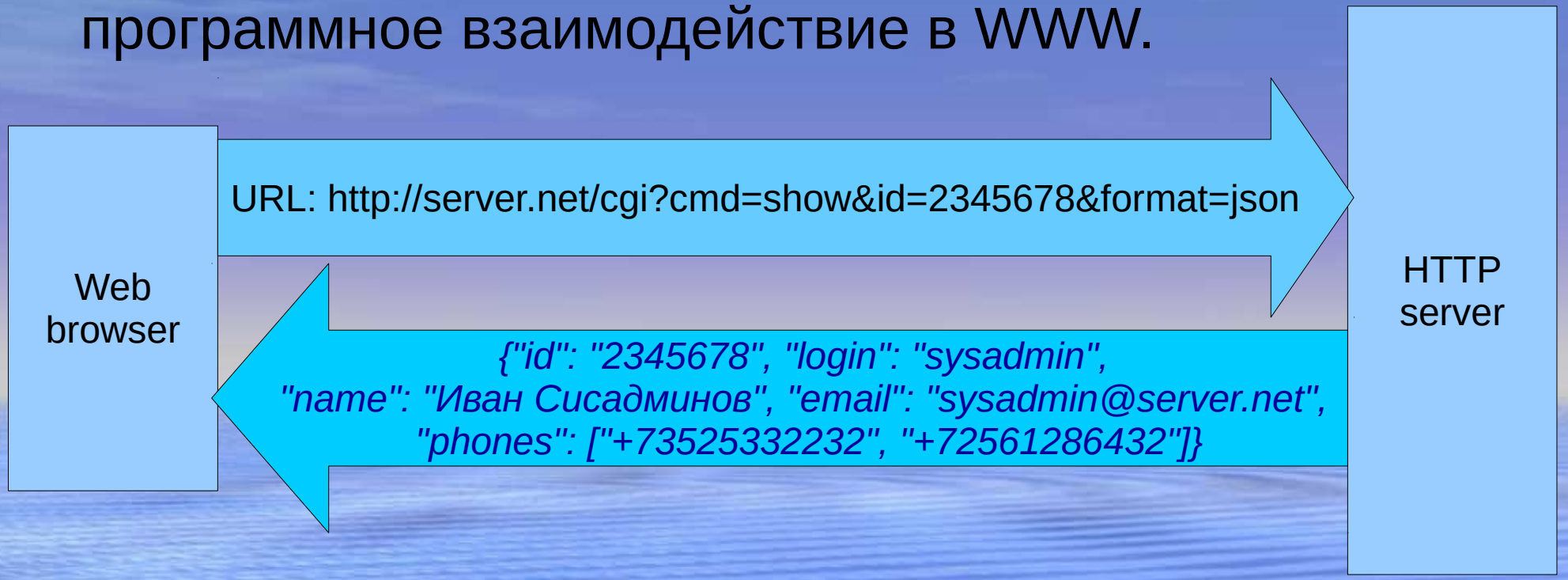
- Чем полезен отдых (*англ.* **REST** ['rest]).
- Кем дано новое описание работы WWW.
- Как программно использовать Web.
- Когда несостоятельно ООП.
- Почему простота лучше Web-сервисов.
- Есть ли аналог SQL для WWW.
- Зачем и где применять **REST**.
- Кто такие **REST**афарианцы.
- ... и многое другое...

Хакнуть WWW?

Вскоре после появления в 1991 году
«Всемирной паутины»
у программистов появилось желание
применять её для создания
распределённых информационных систем:
использовать её не только
для пассивного просмотра
информации человеком,
но и для активного
программного манипулирования ресурсами
(use the Web programmatically).

CGI и т. п.

CGI (Common Gateway Interface) — 1-й стандарт на программное взаимодействие в WWW.



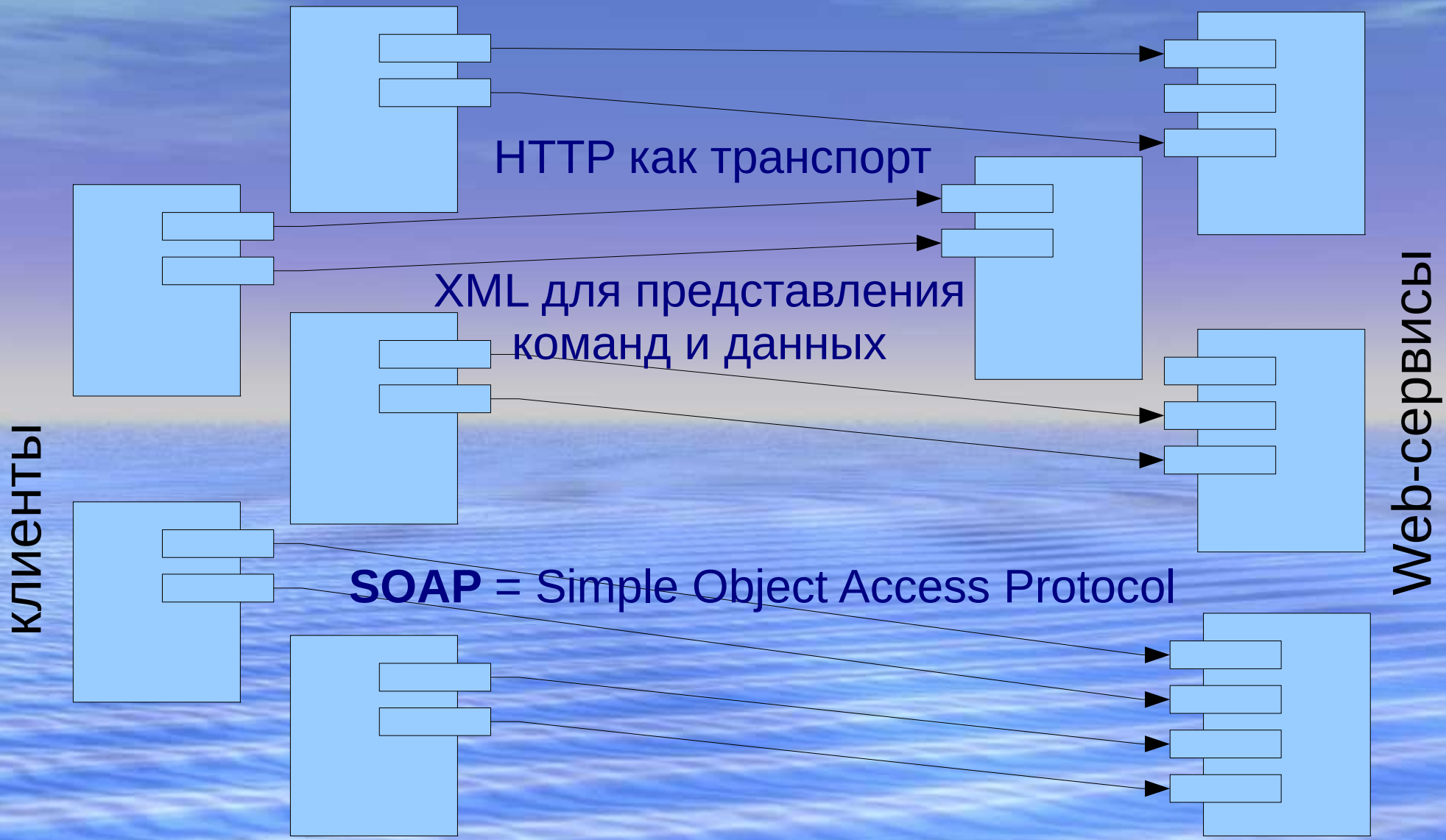
В стандарте на **URL** (Uniform Resource Locator) описан унифицированный способ кодирования параметров:
<схема>://<логин>:<пароль>@<хост>:<порт>/<путь>
?<параметр_1=значение_1&параметр_2=значение_2>
#<якорь>

Традиционная разработка

Объектно-ориентированное
программирование (ООП)
предполагает традиционный подход
к разработке распределённых программных систем
(distributed systems),
основанных на удалённом вызове процедур (RPC):

- Выявить **классы** в предметной области.
- Описать **свойства** для хранения состояния.
- Определить для них **интерфейсные** методы.
- Реализовать **методы** изменения состояния...
- … с определённым **набором параметров** требуемых типов.

Service-Oriented Architecture

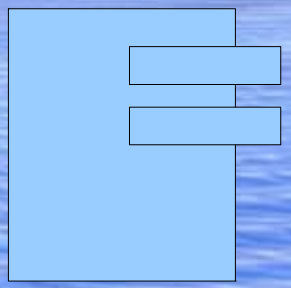


Web-сервисы: XML-RPC, SOAP

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getResourceDetailsResponse xmlns="http://server.net/ws">
      <getResourceDetailsResult>
        <resourceID>2345678</resourceID>
        <name>Иван Сисадминов</name>
        <email>sysadmin@server.net</email>
        <login>sysadmin</login>
        <phones>
          <number>+73525332232</number>
          <number>+72561286432</number>
        </phones>
      </getResourceDetailsResult>
    </getResourceDetailsResponse>
  </soap:Body>
</soap:Envelope>
```

Ответ сервера: объект данных

клиенты



Web-сервисы

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getResourceDetails xmlns="http://server.net/ws">
      <resourceID>2345678</resourceID>
    </getResourceDetails>
  </soap:Body>
</soap:Envelope>
```

Запрос клиента: метод, параметры

Распределённые системы: RPC

RPC, SOAP: REST† in peace

- Слишком много уникальных классов.
- Чересчур много разных интерфейсов.
- Разнообразие параметров: типы, порядок.
- Многообразие протоколов, стандартов.
- Множество способов идентификации.
- Необходимость системы поиска объектов.
- Целое семейство сопутствующих технологий.

Как побороть необоснованную сложность?

† *«Покойтесь с миром»*

Контроль сложности в RDB

В реляционных базах данных (RDB) сложность держится под контролем за счёт применения унификации:

- **единого представления данных** в виде таблиц с первичными ключами (rows of columns in tables / views with PK);
- **стандартного способа доступа** в виде структурированного языка запросов (SQL);
- **ограниченного набора типов данных** (valid SQL data types);

«Краеугольный камень» RDB

Идентификация данных

Первичные ключи (PK)
уникальные для строк в каждой из таблиц

Операции

команды SQL
например:
SELECT

Представление данных

типы данных SQL
например:
INTEGER

Хорошо бы: Web как БД...

SELECT * FROM www.shgpi.edu.ru/teachers;

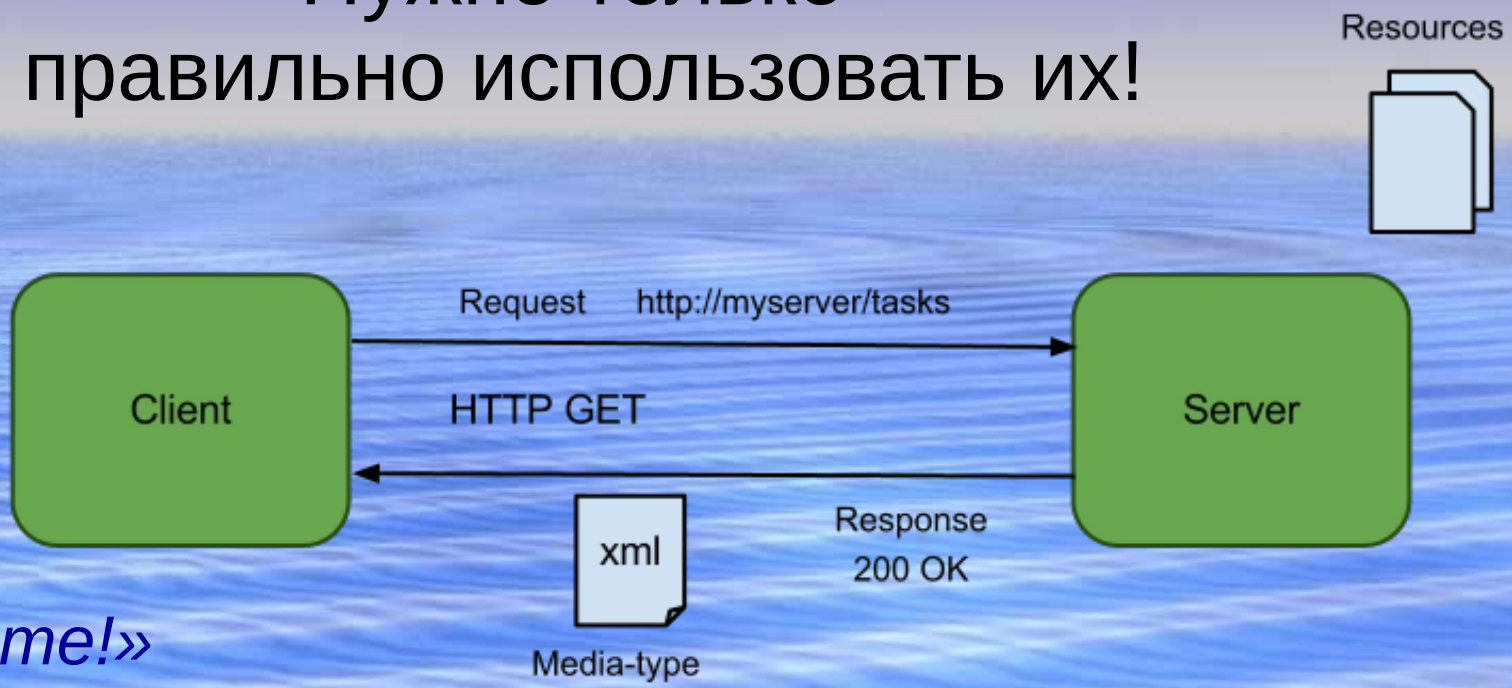
SELECT * FROM www.shgpi.edu.ru/teachers
WHERE id = *1234567*;

UPDATE www.shgpi.edu.ru/teachers
SET email = "*pirogov@shadrinsk.net*"
WHERE id = *1234567*;

DELETE FROM www.shgpi.edu.ru/teachers
WHERE email LIKE *"kourov%"*;

Всё уже придумано: REST*!

На самом деле все средства программного манипулирования данными уже реализованы в стандартах WWW почти с самого начала.
Нужно только правильно использовать их!



** «Отдыхайте!»*

What the Web RESTs* on?

Работа с информационными мультимедийными ресурсами в WWW основана на общеизвестных стандартах (RFC), описывающих:

- адресацию — именование по **URL**;
- доступ — протокол **HTTP**.
- типы данных — форматы **MIME**.

** «На чём же базируется WWW?»*

Кто придумал REST?

REST = Representational State Transfer

Этот термин ввёл **Roy T. Fielding**
в своей докторской диссертации
"Architectural Styles and the Design of Network-based Software Architectures"
(2000).

Рой Т. Филдинг описал концепцию построения распределённого приложения (на основе стандартных технологий WWW), где каждый запрос клиента к серверу содержит в себе исчерпывающую информацию для его обработки, а также о желаемом представлении состояния данных в ответе сервера, и сервер не обязан сохранять информацию о состоянии клиента («клиентской сессии»).

Рой Томас Филдинг

Активный участник разработки:

- **HTTP** (RFC 1945, RFC 2068, RFC 2145, RFC 2616)
- **URL** (RFC 1808), **URI** (RFC 2396)
- **Apache** Web-сервер (httpd)

Один из основателей **Apache Software Foundation** (ASF)

Автор **REST** — «архитектурного стиля» для распределённых вычислительных гипермедиа-систем.

REST как архитектура

REST — архитектурный стиль*
для создания
распределённых программных систем:
ориентированный на ресурсы,
отлично масштабируемый,
гибко адаптируемый,
простой и понятный,
легко реализуемый.

**Architectural style* — *ограничения и соглашения при подходе к проектированию программ.*

REST: как перевести?

Смысл такой: «мы можем переводить ресурс из одного состояния в другое*, при этом получать его в виде требуемого представления».

REST = Representational State Transfer

Варианты перевода с английского:

- «переходы между состояниями [ресурса], доступными как представления [ресурса]»;
- «изменение состояния на основе представлений»;
- «передача представлений состояния»;
- «передача репрезентативного состояния»;
- «манипуляция ресурсами через представление»;
- ???

** WWW-ресурс как «конечный автомат» (state machine).*

Принципы REST

- **resources** — ресурсы, как единый взгляд на данные: они идентифицируются в запросах и отделены от представлений;
- **addressability** — универсальная идентификация ресурсов по URI;
- **uniform interface** — унифицированный интерфейс доступа к ресурсам по HTTP;
- **representations** — множество представлений одного и того же ресурса (Internet Media Types, MIME);
- **client-server** — взаимодействие по запросам;
- **statelessness** — сервер хранит состояние данных, клиент хранит состояние приложения;

Треугольник REST

Существительные

Ресурсы

идентификаторы URI

например:

https://ru.wikipedia.org/wiki/REST

Глаголы

Операции

методы HTTP

например:

GET

Типы данных

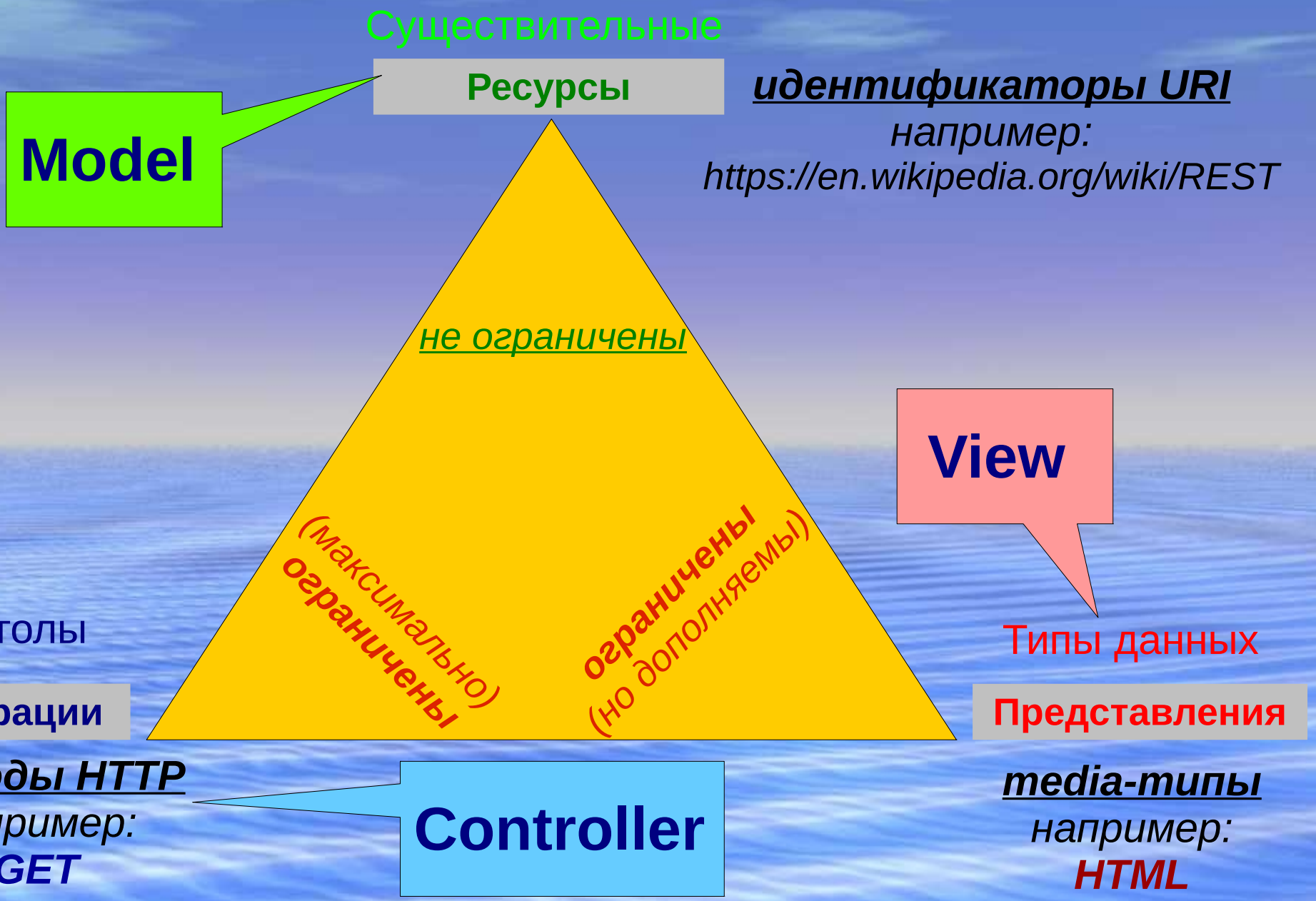
Представления

media-типы

например:

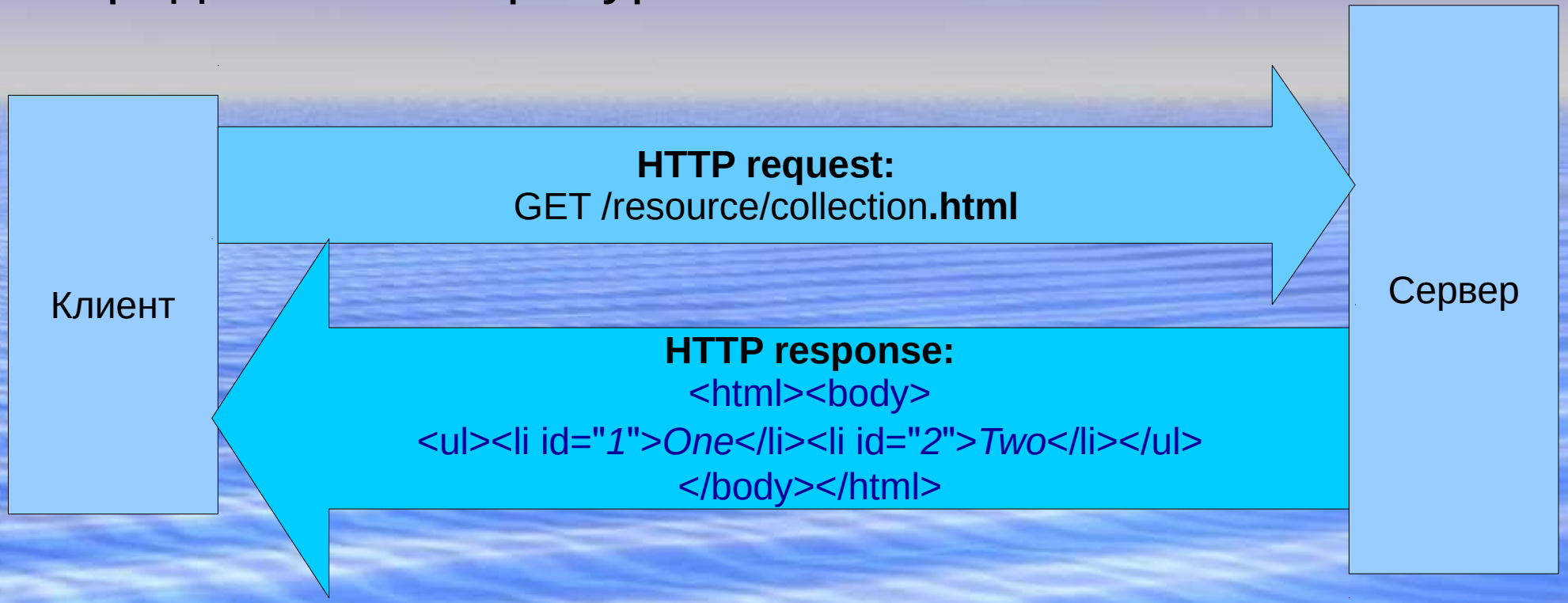
HTML

REST и MVC


Существительные
Ресурсы
идентификаторы URI
например:
https://en.wikipedia.org/wiki/REST
Model
не ограничены
(максимально)
ограничены
ограничены
(но дополняемы)
View
Глаголы
Типы данных
Операции
Представления
методы HTTP
например:
GET
Controller
media-типы
например:
HTML

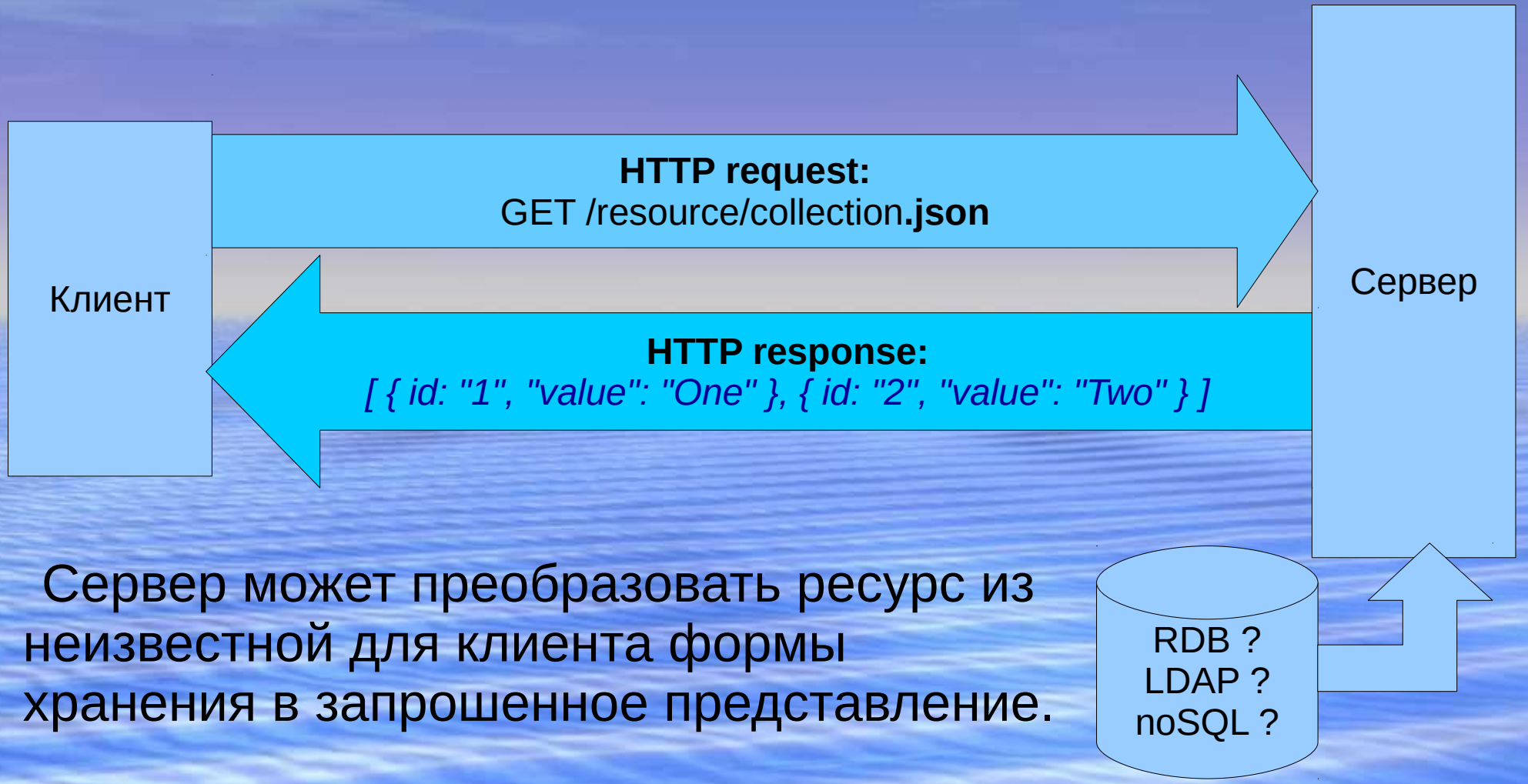
RESTful-взаимодействие

- Ресурс: уникально идентифицируется по URI.
- REST-запрос: HTTP GET / PUT / POST / DELETE.
- Данные для обновления: в параметрах запроса.
- Желаемое представление: в параметрах запроса.
- Представление ресурса — в HTTP-ответе.



RESTful-взаимодействие

Клиент может запросить тот же самый ресурс в другом представлении:



Сервер может преобразовать ресурс из неизвестной для клиента формы хранения в запрошенное представление.

Ресурсы: URI

URI (Uniform Resource Identifier) — уникальный идентификатор ресурса в виде URL:

- server.net/resource/collection — коллекция;
- server.net/resource/collection/*filter* — часть коллекции;
- server.net/resource/collection/*id* — ресурс;
- application.info/class/ — список объектов класса;
- application.info/class/*id* — объект из списка;
- application.info/class/subclass — объекты подкласса;
- localhost/path/to/dir/ — список файлов в каталоге;
- localhost/path/to/dir/*file*.txt — файл;

Ресурсы подразделяются на 2 вида: **единичные** ресурсы (объекты) и **коллекции** (списки, наборы, массивы) ресурсов. Но доступ к любому из них производится однотипно.

URI = URL + URN

В стандарте URI, помимо широко известных URL, описываются правила единообразного именования ресурсов. **URN** (Uniform Resource Name) — постоянная последовательность символов, идентифицирующая абстрактный или физический ресурс:

urn:<NID=Namespace Identifier>:<NSS=Namespace Specific String>

Примеры:

- urn:isbn:5-9556-0078-7
- urn:ietf:rfc:1737
- urn:iso:std:iso:26300:-2010:ru
- urn:oid:2.16.643
- urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66
- urn:imei:86087200015525O
- urn:mac:B4:B5:2F:AD:D4:D4
- urn:snmp:mib:system.sysUpTime.0
- urn:caiman.org:presentation:rest_intro:version:2016-03-31:pdf

Действия: HTTP-методы

В протоколе HTTP (HyperText Transfer Protocol) предусмотрен весь стандартный набор операций для манипулирования данными (**CRUD** = **C**reate, **R**ead, **U**pdate, **D**elete):

- **POST** = *SQL CREATE* ~ создать, добавить
- **GET** = *SQL SELECT* ~ выбрать, получить
- **PUT** = *SQL UPDATE* ~ изменить, обновить
- **DELETE** = *SQL DELETE* ~ удалить, стереть

REST API

Методы HTTP для манипуляций с ресурсами:

GET /resource/collection	запросить коллекцию ресурсов
POST /resource/collection	добавить единичный ресурс в коллекцию
GET /resource/collection/*id*	запросить единичный ресурс
PUT /resource/collection/*id*	изменить ресурс (или коллекцию)
DELETE /resource/collection/*id*	удалить ресурс (или коллекцию)

Сочетание URI и метода HTTP определяет, какое действие с ресурсом требуется выполнить.

Представление коллекции обычно содержит идентификаторы (URI) одиночных ресурсов.

Представления: MIME

Multipurpose Internet Mail Extensions (MIME)

Стандартизированные (RFC) представления данных, в т. ч. мультимедийных, были разработаны ещё для вложений электронной почты и дополнены новыми типами после появления WWW.

Клиент может запросить у сервера требуемое представление ресурса

- как часть URI (выглядит, как суффикс файла):
  - resource.**HTML**
  - resource.**XML**
  - resource.**JSON**
  - resource.**PDF**
- в заголовке *Accept* запроса HTTP:
  - Accept: **text/html**
  - Accept: **application/xml**
  - Accept: **application/json**
  - Accept: **application/pdf**

Сервер сообщает о представлении передаваемого ресурса в заголовке *Content-Type* ответа HTTP.

MIME / Internet Media Types

MIME = Multipurpose Internet Mail Extensions

Стандартизированные (RFC) базовые типы: application, audio, example, image, message, model, multipart, text, video.

Примеры:

- application/json - JavaScript Object Notation, JSON (RFC 4627);
- application/octet-stream - двоичный файл (RFC 2046);
- application/pdf - Portable Document Format, PDF (RFC 3778);
- application/xml - eXtensible Markup Language, XML;
- audio/mpeg - MPEG-аудио, включая MP3 (RFC 3003);
- image/png - Portable Network Graphics, PNG (RFC 2083);
- message/rfc822 - сообщения E-mail (RFC 2045, 2046);
- model/vrml - файлы WRL и VRML (RFC 2077);
- text/csv - Comma-Separated Values, CSV (RFC 4180);
- text/html - HyperText Markup Language, HTML (RFC 2854);
- text/plain - текстовые данные (RFC 2046, 3676);
- video/mpeg - MPEG-1 (RFC 2045, 2046);

Нестандартные типы (x): application/x-font-ttf, text/x-jquery-tmpl, ...

Типы вендоров (vnd): application/vnd.ms-excel, audio/vnd.wave, ...

Обратная связь?

Коды состояния HTTP *(избранные)*:

- **1xx: Informational (информационные)**:
  - 102 Processing («идёт обработка»).
- **2xx: Success (успешно):**
  - 200 OK («выполнено»).
  - 201 Created («создано»).
- **3xx: Redirection (перенаправление):**
  - 301 Moved Permanently («перемещено навсегда»): изменён URI.
  - 304 Not Modified («не изменялось»): можно брать из кэша.
- **4xx: Client Error (ошибка клиента):**
  - 400 Bad Request («неверный запрос»).
  - 401 Unauthorized («не авторизован»).
  - 403 Forbidden («запрещено»).
  - 404 Not Found («не найдено»).
  - 410 Gone («удалено»).
- **5xx: Server Error (ошибка сервера):**
  - 500 Internal Server Error («внутренняя ошибка сервера»).
  - 501 Not Implemented («не реализовано»).
  - 503 Service Unavailable («сервис недоступен»).
  - 507 Insufficient Storage («переполнение хранилища»).

Диалог в стиле REST

- **Запрос** (HTTP Request):
  GET /collection/resource/*2345678* HTTP/1.1
  Host: server.net
  Accept: application/json
  Connection: close
- **Ответ** (HTTP Response):
  HTTP/1.1 200 OK
  Content-Language: ru
  Content-Type: application/json; charset=utf-8
  Content-Length: 155
  Connection: close

{"id": "2345678", "login": "sysadmin", "name": "Иван Сисадминов",
"phones": ["+73525332232", "+72561286432"],
"email": "sysadmin@server.net"}

Преимущества REST

- **Открытость**: это не стандарт, а архитектурный стиль, дающий методологию и оставляющий реализацию на усмотрение разработчика.
- **Унификация**: за счёт применения общепризнанных общедоступных стандартов.
- **Надёжность**: информации о состоянии клиента не хранится на сервере, GET-запросы повторяемы, ...
- **Прозрачность** системы взаимодействия.
- **Простота** архитектуры, интерфейса, протокола.
- **Переносимость**: независимость компонентов от языка, операционной системы, платформы.
- **Сопровождаемость**: лёгкость внесения изменений.
- **Производительность**: за счёт использования кэша.
- **Масштабируемость** и способность эволюционировать, приспосабливаясь к новым требованиям (пример: WWW).

REST — весьма модно, но на самом деле полезно!

Многие современные информационные системы и приложения имеют REST-интерфейс для программного доступа к их ресурсам.

Где применим REST

- И в Internet, и в intranet.
- При разработке новых Web-приложений.
- Для взаимодействия между Web-приложениями.
- Для разработки нового простого API к старому Web-приложению вместо имеющегося сложного API.
- Для создания Web-интерфейса к существующим унаследованным (legacy) информационным системам.
- Для интеграции любых информационных систем на предприятии, в организации, на разных сайтах.
- . . .

REST как идеология

Архитектура программной системы сильно упрощается, если при её проектировании и разработке применять методологию REST:

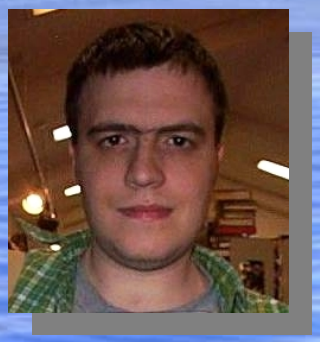
- Данные представляются как **набор логических ресурсов**.
- Для идентификации выбирается **подходящая иерархия имён** ресурсов, которая отображается на хранилище данных.
- **Только 4 стандартные операции** используются для доступа к ресурсам.
- Для API / для пользователей **реализуются свои представления** ресурсов (JSON, YAML, XML / XHTML, PDF, ...).

Кто такие РЕСТафарианцы?

RESTafarians
(по аналогии с *растафарианцами* , также известными, как *растаманы*) —
ярые последователи принципов REST
при разработке приложений
8-)

Tim Bray

David Heinemeier Hansson



Leonard Richardson

Sam Ruby

- Mark Baker
- Benjamin Carlyle
- Duncan Cragg
- Joe Gregorio
- Pete Lacey
- Mark Nottingham
- Paul Prescod
- Stefan Tilkov
- . . .

REST в программировании

Примеры программных каркасов (frameworks) для создания приложений в стиле REST на некоторых языках:

- Java — Restlet <http://www.restlet.org/>
- Perl — Catalyst <http://www.catalystframework.org/>
- Python — Django <http://www.djangoproject.com/>
- Ruby — Rails <http://rubyonrails.org/>

REST на всех языках

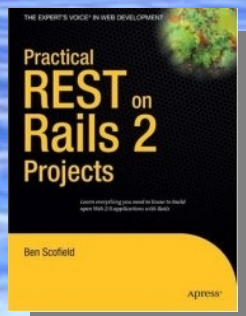
Взаимодействие в стиле REST легко реализовать в любом языке программирования, поскольку для этого есть **стандартные библиотеки** для работы с HTTP, URL, MIME.

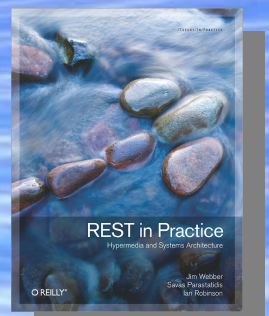
- Android — google-http-client-android.
- Java — HttpClient; Unirest, Retrofit, ...
- JavaScript — AJAX для браузеров.
- Perl — LWP.
- PHP — HttpRequest; Unirest.
- Python — Requests.
- Ruby — Net::HTTP.

Что почитать *«for REST*»*

- *Leonard Richardson, Sam Ruby* **«Resful Web Services»** - O'Reilly Media, 2007.
- Ben Scofield **«Practical REST on Rails 2 Projects»** - Apress, 2008.
- *Jon Flanders* **«RESTful .NET»** - O'Reilly Media, 2008.
- *Samisa Abeysinghe* **«RESTful PHP Web Services» -** Packt Publishing, 2008.
- *J. Webber, S. Parastatidis, I. Robinson* **«REST in Practice»** - O'Reilly Media, 2010.
- *Subbu Allamaraju* **«RESTful Web Services Cookbook»** - O'Reilly Media, 2010.
- *Mark Masse* **«REST API Design Rulebook»** - O'Reilly Media, 2011.
- *Erik Wilde, Cesare Pautasso* **«REST: From Research to Practice»** - Springer Science & Business Media, 2011.

** «для отдыха»*

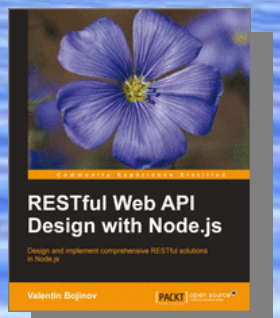


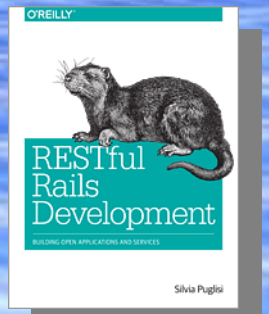


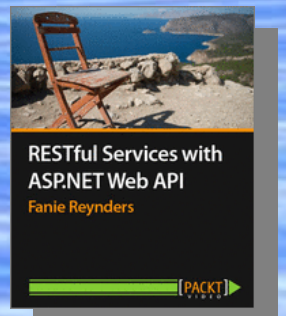
Другие RESTful*-книги

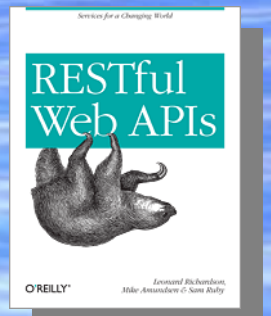
- *Thomas Erl et al.* **«SOA with REST»** - Prentice Hall, 2012.
- *L. Richardson, M. Amundsen, S. Ruby* **«RESTful Web APIs»** - O'Reilly Media, 2013.
- *Fanie Reynders* **«RESTful Services with ASP.NET Web API»** - Packt Publishing, 2014.
- *Silvia Puglisi* **«RESTful Rails Development» -** O'Reilly Media, 2015.
- *Valentin Bojinov* **«RESTful Web API Design with Node.js»** - Packt Publishing, 2015.
- *Jobinesh Purushothaman* **«RESTful Java Web Services»** - Packt Publishing, 2015.
- *Mike Amundsen* **«RESTful Web Clients»** - O'Reilly Media, 2016.

** «отдыхательные»*









And the REST* of the info … is in the Web

(остальная информация)*

Неплохие ресурсы (для начала):

- «Типы HTTP-запросов и философия REST»
- «Web-сервисы RESTful: основы» @ IBM
- «Руководство по созданию RESTful сервиса»
- «RESTful API для сервера – делаем правильно»
- «Программирование в стиле REST на Ruby» @ IBM

А потом:

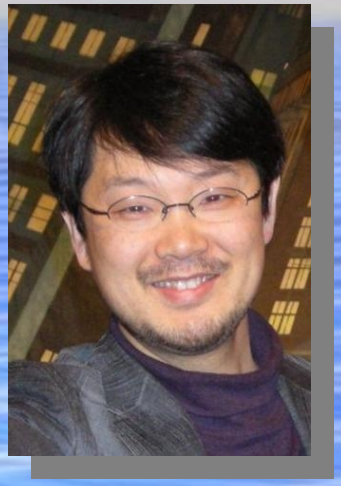
- *«OK, Google!»* ;-)

Благодарности

Рою Томасу Филдингу – за REST

Дэвиду Хайнемейеру Ханссону – за Rails

Мацумото Юкихиро – за Ruby



Винстенту Шталю – за картинку фона

Вопросы?

mailto: *Mikhail* @ *Shokhirev* . com

А теперь — демонстрация!

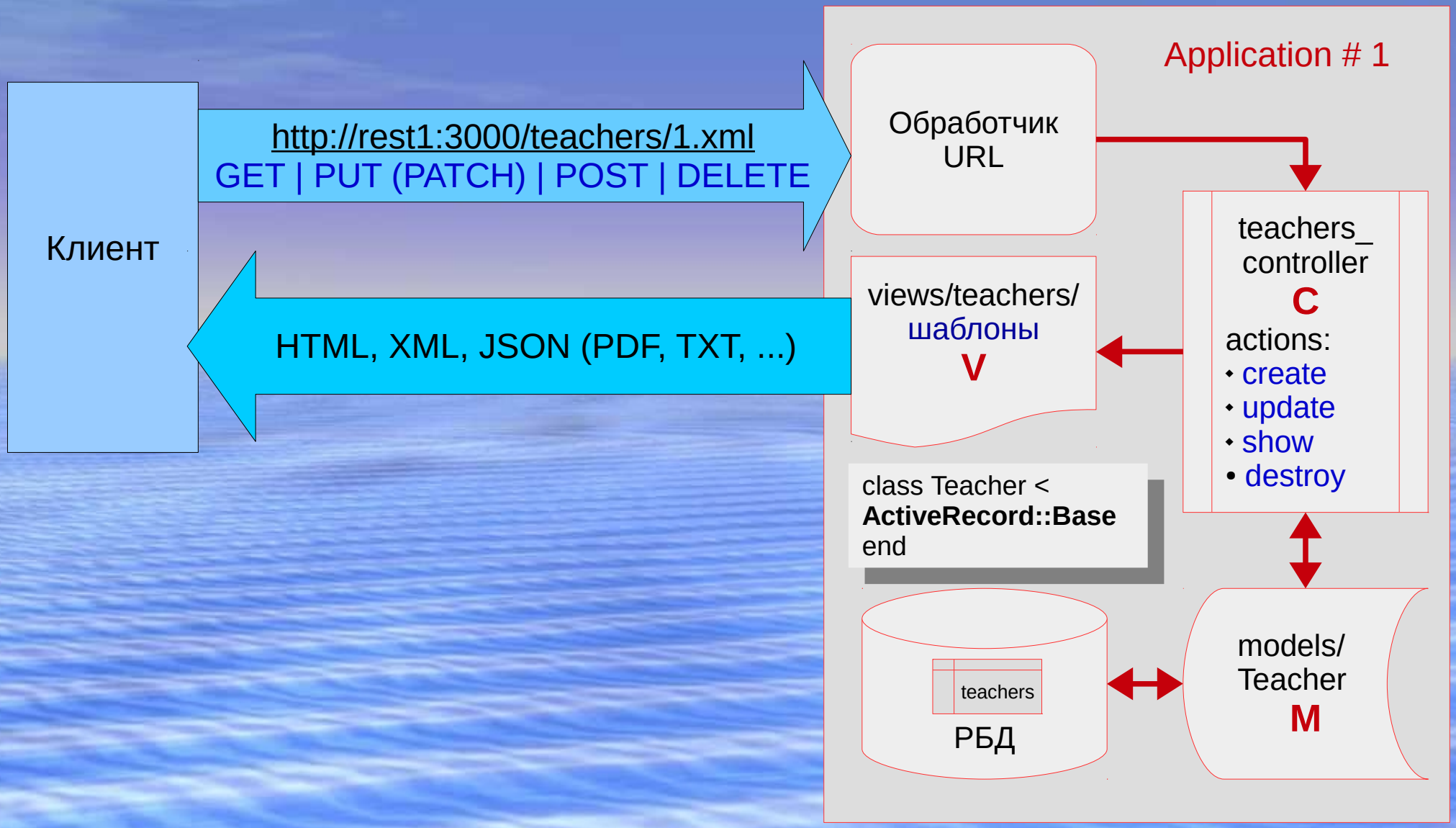
Использованные средства:

Ruby on Rails

(*RubyStack @ BitNami.org*)

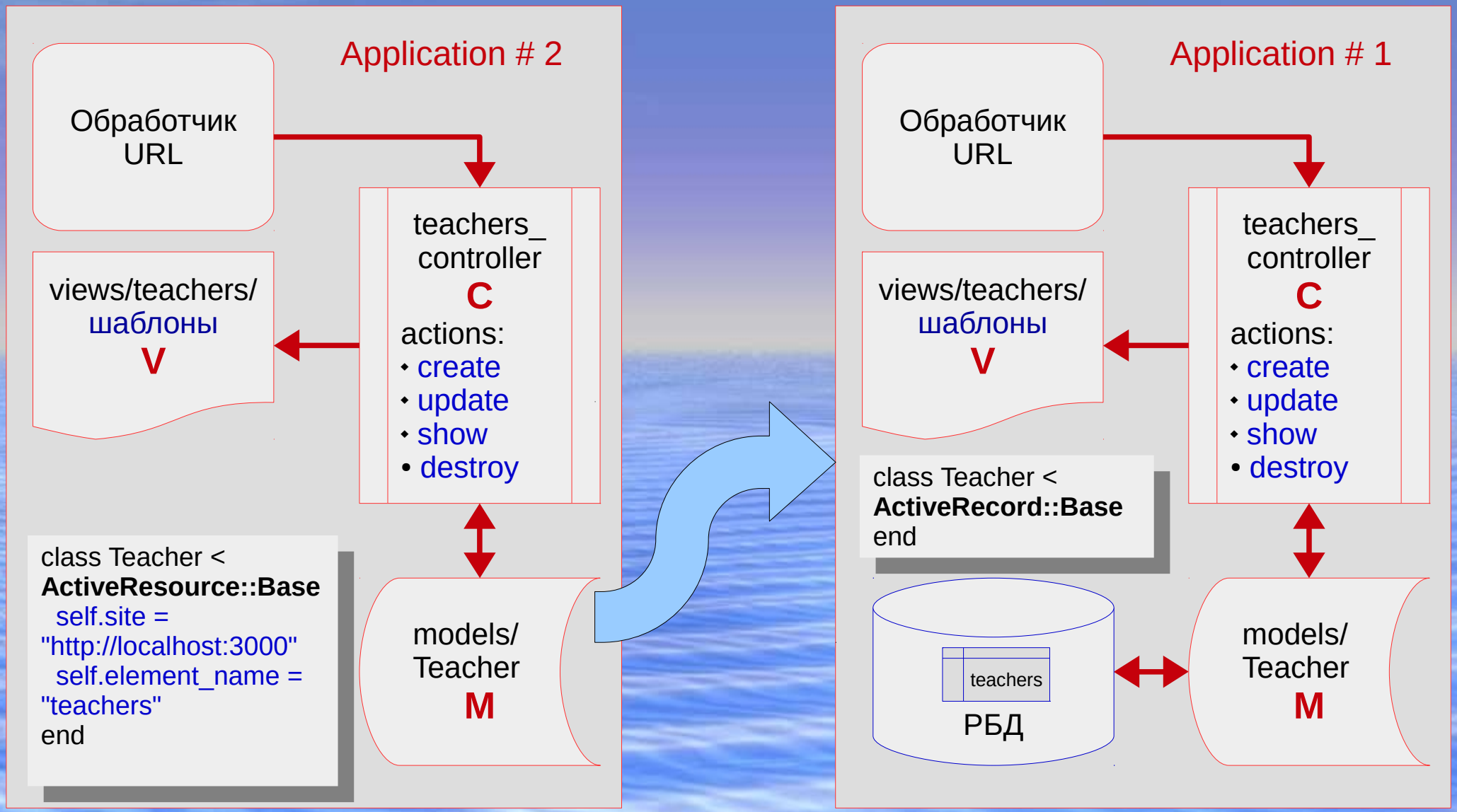
REST в Rails

В основе **Ruby On Rails** — архитектура REST.



ActiveResource в Rails

ActiveResource (#2) обращается через REST к ActiveRecord (#1).



```
/opt/rubystack/rubyconsole
gem install activeresource

# Application # 1 (rest1)
rails new rest1 --database-sqlite3 --skip-bundle
cd rest1

rails generate scaffold teachers name:string email:string telephone:string birthday:date
rake db:migrate
rake routes
      Prefix Verb URI Pattern                  Controller#Action
    teachers GET /teachers(.:format)           teachers#index
             POST /teachers(.:format)          teachers#create
 new_teacher GET /teachers/new(.:format)       teachers#new
edit_teacher GET /teachers/:id/edit(.:format)  teachers#edit
     teacher GET /teachers/:id(.:format)       teachers#show
           PATCH /teachers/:id(.:format)       teachers#update
             PUT /teachers/:id(.:format)       teachers#update
          DELETE /teachers/:id(.:format)       teachers#destroy
rails server

URL: http://localhost:3000/teachers
    logs/development.log
    Started GET "/teachers" for 127.0.0.1 at 2016-03-23 15:18:14 +0500
    Processing by TeachersController#index as HTML

    Started GET "/teachers/1" for 127.0.0.1 at 2016-03-23 15:19:48 +0500
    Processing by TeachersController#show as HTML
    Parameters: {"id"=>"1"}

# разрешить обмен данными с другими приложениями по REST в формате JSON
    app/controllers/application_controller.rb
    protect_from_forgery unless: -> { request.format.json? }
```

 # Application # 1 (rest1) – продолжение


```
# добавить обработчики форматов: JSON, XML
    app/controllers/teachers_controller.rb
    def show
      respond_to do |format|
        format.html { render :show }
        format.json { render json: @teacher }
        format.xml  { render xml: @teacher }
      end
    end
```

URL: http://localhost:3000/teachers/1.html

```
<html><head><title>Rest1</title></head><body>
<p><strong>Name:</strong>Пирогов Владислав Юрьевич</p>
<p><strong>Email:</strong>pirogov@shadrinsk.net</p>
<p><strong>Telephone:</strong>+79630092234</p>
<p><strong>Birthday:</strong>2016-12-13</p>
<a href="/teachers/1/edit">Edit</a> |
<a href="/teachers">Back</a>
</body></html>
```

URL: http://localhost:3000/teachers/1.json

```
{"id":1, "name":"Пирогов Владислав Юрьевич", "email":"", "telephone":"+79630092234",
"birthday":"2016-12-13", "created_at":"2016-03-23T06:42:20.732Z", "updated_at":"2016-03-
23T09:17:08.587Z"}
```

URL: http://localhost:3000/teachers/1.xml

```
<teacher><id type="integer">1</id><name>Пирогов Владислав Юрьевич</name><email>
pirogov@shadrinsk.net</email><telephone>+79630092234</telephone>
<birthday type="date">2016-12-13</birthday><created-at type="dateTime">2016-03-
23T06:42:20Z</created-at><updated-at type="dateTime">2016-03-23T09:17:08Z</updated-at>
</teacher>
```

Application # 2 (rest2)


```
rails new rest2 --skip-bundle
cd rest2
    Gemfile
    gem 'activeresource', '>= 4.0.0'

    app/models/teacher.rb
    require 'active_resource'
    class Teacher < ActiveResource::Base
      self.site = "http://localhost:3000"
      self.element_name = "teachers"
    end
rails console
t0 = Teacher.get 1
=> {"id"=>1, "name"=>"Пирогов Владислав Юрьевич", "email"=>"pirogov@shadrinsk.net",
"telephone"=>"", "birthday"=>"2016-12-13", "created_at"=>"2016-03-23T06:42:20.732Z",
"updated_at"=>"2016-03-23T06:42:20.732Z"}
t1 = Teacher.find 1
=> #<Teacher:0x0055b2c2139078 @attributes={"id"=>1, "name"=>"Пирогов Владислав Юрьевич",
"email"=>"pirogov@shadrinsk.net", "telephone"=>"", "birthday"=>"2016-12-13",
"created_at"=>"2016-03-23T06:42:20.732Z", "updated_at"=>"2016-03-23T06:42:20.732Z"},
@prefix_options={}, @persisted=true>
t1.telephone="+79630092234"
t1.save
    logs/development.log
    Started PUT "/teachers/1.json" for 127.0.0.1 at 2016-03-23 14:00:46 +0500
    Processing by TeachersController#update as JSON
    Parameters: {"id"=>"1", "name"=>"Пирогов Владислав Юрьевич",
"email"=>"pirogov@shadrinsk.net", "telephone"=>"+79630092234", "birthday"=>"2016-12-13",
"created_at"=>"2016-03-23T06:42:20.732Z", "updated_at"=>"2016-03-23T06:42:20.732Z",
"teacher"=>{"id"=>"1", "name"=>"Пирогов Владислав Юрьевич",
"email"=>"pirogov@shadrinsk.net", "telephone"=>"+79630092234", "birthday"=>"2016-12-13",
"created_at"=>"2016-03-23T06:42:20.732Z", "updated_at"=>"2016-03-23T06:42:20.732Z"}}
```

Application # 2 (rest2) – продолжение

```
t2 = Teacher.find 2
=> #<Teacher:0x0055de0100b380 @attributes={"id"=>2, "name"=>"Самозванец", "email"=>"",
"telephone"=>"", "birthday"=>"2016-03-23", "created_at"=>"2016-03-23T06:43:31.670Z",
"updated_at"=>"2016-03-23T06:43:31.670Z"}, @prefix_options={}, @persisted=true>
t2.destroy
=> #<Net::HTTPNoContent 204 No Content readbody=true>
    logs/development.log
    Started DELETE "/teachers/2.json" for 127.0.0.1 at 2016-03-23 14:18:07 +0500
    Processing by TeachersController#destroy as JSON
    Parameters: {"id"=>"2"}

t3 = Teacher.create name: "Слинкин Дмитрий Анатольевич", email: "sda@shgpi.edu.ru"
=> #<Teacher:0x0055de010f1268 @attributes={"name"=>"Слинкин Дмитрий Анатольевич",
"email"=>"sda@shgpi.edu.ru", "id"=>3, "telephone"=>nil, "birthday"=>nil,
"created_at"=>"2016-03-23T09:35:00.185Z", "updated_at"=>"2016-03-23T09:35:00.185Z"},
@prefix_options={}, @persisted=true, @remote_errors=nil, @validation_context=nil,
@errors=#<ActiveResource::Errors:0x0055de010f0818 @base=#<Teacher:0x0055de010f1268 ...>,
@messages={}>>
    logs/development.log
    Started POST "/teachers.json" for 127.0.0.1 at 2016-03-23 14:35:00 +0500
    Processing by TeachersController#create as JSON
    Parameters: {"name"=>"Слинкин Дмитрий Анатольевич", "email"=>"sda@shgpi.edu.ru",
"teacher"=>{"name"=>"Слинкин Дмитрий анатольевич", "email"=>"sda@shgpi.edu.ru"}}

Tutors = Teacher.all
=> #<ActiveResource::Collection:0x0055de017ec7c0 @elements=[#<Teacher:0x0055de017ec6a8
@attributes={"id"=>1, "name"=>"Пирогов Владислав Юрьевич", ... >] ...>
    logs/development.log
    Started GET "/teachers.json" for 127.0.0.1 at 2016-03-23 14:52:04 +0500
    Processing by TeachersController#index as JSON
```

Application # 2 (rest2) – продолжение


```
# скопировать описание маршрутов, контроллер, шаблоны из rest1 в rest2
cp ../rest1/config/routes.rb                          app/config
cp ../rest1/app/controllers/teachers_controller.rb    app/controllers
cp -r ../rest1/app/views/teachers                     app/views
```

views/teachers/_form.html.erb

```
<div class="field"> <%= f.label :name %><br>
  <%= text_field @teacher, :name, id: "teacher_name", name: "teacher[name]" %>
</div>
<div class="field"> <%= f.label :email %><br>
  <%= text_field @teacher, :email, id: "teacher_email", name: "teacher[email]" %>
</div>
<div class="field"> <%= f.label :telephone %><br>
  <%= text_field @teacher, :telephone, id: "teacher_telephone", name: "teacher[telephone]" %>
</div>
<div class="field"> <%= f.label :birthday %><br>
  <%= text_field @teacher, :birthday, id: "teacher_birthday", name: "teacher[birthday]" %>
</div>
```

```
rails server -p 3001
```

URL: http://localhost:3001/teachers/5

{"id":5,"name":"Шохирев Михаил", "email":"mikhail@shokhirev.com", "telephone":"+73525332232", "birthday":"1958-11-12", "created_at":"2016-03-30T08:16:29.774Z", "updated_at":"2016-03-30T08:16:29.774Z"}

logs/development.log

```
Started GET "/teachers/5" for 127.0.0.1 at 2016-03-30 13:25:26 +0500
Processing by TeachersController#show as HTML
Parameters: {"id"=>"5"}
Rendered teachers/show.html.erb within layouts/application (1.0ms)
Completed 200 OK in 11ms (Views: 4.1ms | ActiveRecord: 0.0ms)
```