Department:

Country:

Tools, REST, Swagger

© Legal Name City 7 April 2020

Internal

Agenda

- What is REST?
- Explore typical problem & solutions
- Popular REST approach
- Deep-dive REST and Pitfalls of REST
- What is Swagger
- Exercise
- References

What is REST?

What is REST?

Representational state transfer

An architectural style for sharing data between applications

Structured architectural properties

Implemented via HTTP

What is REST? – Representational state transfer

Representational state transfer

Representations:
- How resource(e.g get manipulated)
- Part of the resource state transferred between client and server
- JSON or XML

- Example:
  - Resource: Doctor
  - Service: Doctor Profile
  - Representation:
    - Name, Id
    - JSON

GENERALI

What is REST? – Architectural style and properties

An architectural style for sharing data between applications

Structured architectural properties

6 constraints in REST architectural style:

- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (optional)

What is REST? – Architectural style and properties

Uniform Interface

- Define the interface between client and server
- Simplifies and decouple architecture
- Restful design:
- HTTP verbs (e.g Get, Put, Post, Delete)
- URIs (resource name)
- HTTP response (status, body)

Stateless

- Server contains no client state
- Self-descriptive message

What is REST? – Architectural style and properties

Client-Server

- Assume a disconnected system
- Uniform interface to connect client and server

Cacheable

- Server responses are cacheable
  - Implicitly (Client defined)
  - Explicitly (Server defined: e.g max-age)
  - Negotiated

What is REST? – Architectural style and properties

Layered System

- Client can’t assume direct connection to server
- Improves scalability

Code on Demand (Optional)

- Transfer logic to client
  - Client functionality cab be extended by downloading and executing code in the form of applets or scripts

Explore typical problem & solutions

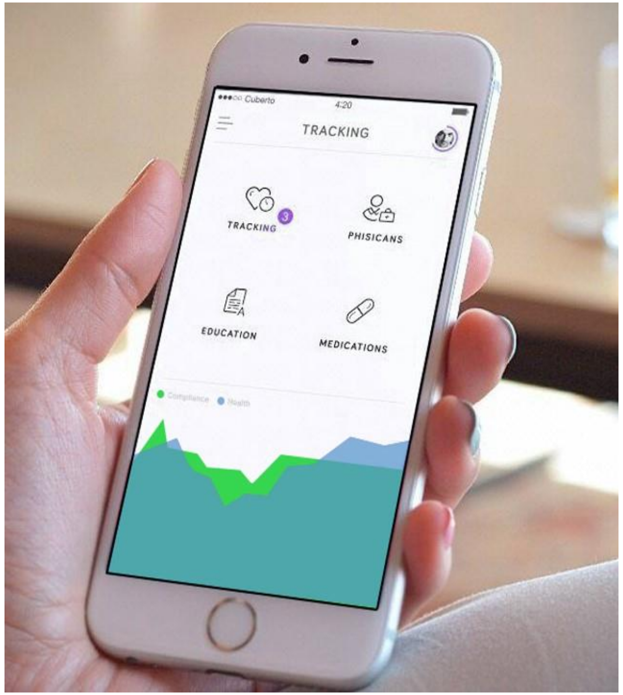
Explore typical problem & solutions

Explore this problem:

You’re a web developer building a healthcare product.

You need to retrieve patient data from the server.

What are some of the typical ways to retrieve data in this scenario?



Typical solutions offer some benefits… but they come with drawbacks

	Custom protocol implementation	Direct database access	SOAP	REST APIs & similar implementations
Benefits	• Full control over implementation • Security can be integrated into existing systems	• SQL is easy & ubiquitously known • Flexibility of a query engine	• Very descriptive API • Utilizes secured authentication	• More detailed discussion to follow
Drawbacks	• Considerable time spend • Potential for security breaches	• Tight coupling b/w client & database • Potential for security breaches	• Tight coupling b/w client & database • Lack of supported data formats	

Let's evaluate REST more closely in this scenario

Supports any hypertext format

Protocol independent – abstraction, authentication, communication and CRUD operations

APIs are independent of client

Requires fewer resources in transport as well as description

Caching utilizes less network resources

Popular REST approach

API path naming convention

- **Verbs are bad**
- **Nouns are good**
- **Plurals are better**
- **For complex variations, put them after “?”**

e.g. /dogs?color=brown&state=running&location=park

REST API

Using HTTP Methods for RESTful Services

- **POST** - used to create a new entity, but it can also be used to update an entity
- **GET** - used to read (or retrieve) a representation of a resource
- **PUT** - update an existing entry
- **DELETE** - delete an entry; however, the resource does not have to be removed immediately. It could be an asynchronous or long-running request

HTTP status codes

- 1XX - informational
- 2XX - success
- 3XX - redirection
- 4XX - client error
- 5XX - server error

Samples

- The first one is for a collection

  e.g. /dogs
- The second is for an element

  e.g. /dogs/1234

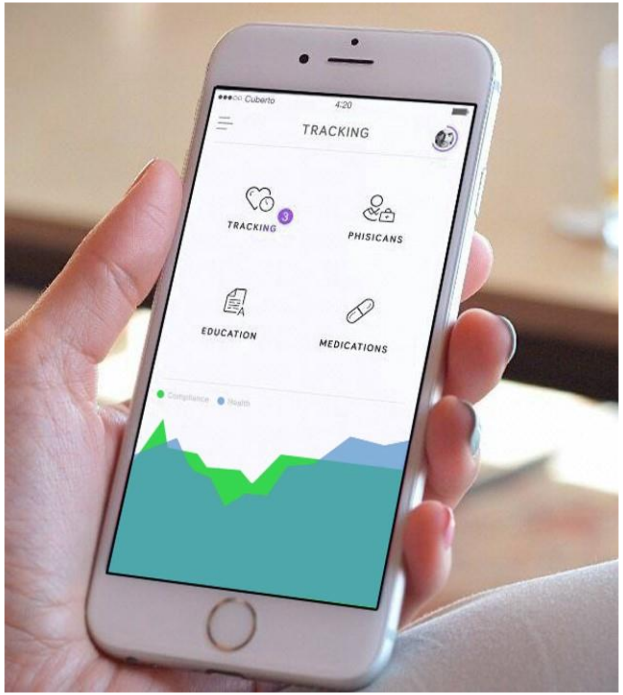
Resource	**POST** create	**GET** read	**PUT** update	**DELETE** delete
/dogs	create a new dog	list dogs	bulk update dogs	delete all dogs
/dogs/foo	error	show Foo	if exists, update Foo; if not, error	delete Foo

Explore typical problem & solutions

Let's add a dimension to our problem:

We've chosen to utilize REST APIs to retrieve the data for a patient and the doctors s/he may've visited.

What would our REST API solution look like?



Explore typical problem & solutions

We would need a data model w/ 3 entities - Patients, Visits, and Doctors…
…and we would write conforming code as our solution

Patients

- ID
- Name

Visits

- Patient ID
- Doctor ID
- Completed

Doctors

- ID
- Name

Simplified REST API

```
var express = require("express");
var app = express();
var router = express.Router();

router.route("/patients/:id").get(function(req, res) {
  var patient = db.getPatient(req.body.id);
  res.json(patient);
});

router.route("/doctors/:id").get(function(req, res) {
  var doctor = db.getDoctor(req.body.id);
  res.json(doctor);
});

router.route(“/visits/:patientId”).get(function(req, res) {
  var visits = db.getVisits(req.body.patientId);
  res.json(visits);
});
```

…but where might our solution struggle?

Deep-dive REST and Pitfalls of REST

Pitfalls of REST

REST comes with pitfalls to be aware of…

- REST APIs are synchronous & hence resource intensive
- REST offers fewer verbs to operate on (e.g. no upsert/merge functionality)

- Loses relationships between entities

- Often misused losing scalability

Pitfalls of REST

Even the popular REST implementations have pitfalls to be aware of…

Popular REST API

```
router.route("/patientsWithDoctors/:patientId").get(function(req,
res) {
  var result = {
    patient: {},
    doctors: []
  };
  result[‘patient’] = db.getPatient(req.body.patientId);
  var visits = db.getVisits(req.body.patientId);
  visits.forEach(function(visit) {
    result.doctors.push(db.getDoctor(visit.doctor_id));
  });
  res.json(result);
});
```

Discuss the downside of this approach

- Does not conform to standards
- Increases implementation complexity
- Endpoints tailored to specific use case – restricting reuse
- Any new feature requires editing an endpoint, resulting into potential breaking changes

What is SWAGGER?

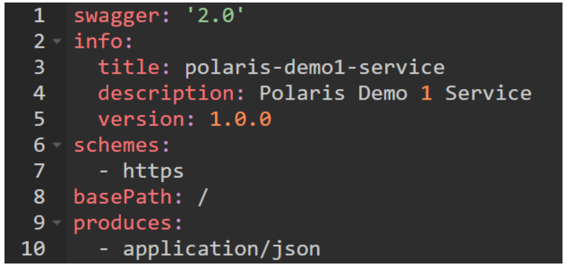
SWAGGER

Swagger is a framework for describing your API using a common language that everyone can understand. Think of it as a blueprint for a house. You can use whatever building materials you like, but you can't step outside the parameters of the blueprint.

- It's comprehensible for developers and non-developers.
- It's human readable and machine readable.
- It's easily adjustable.

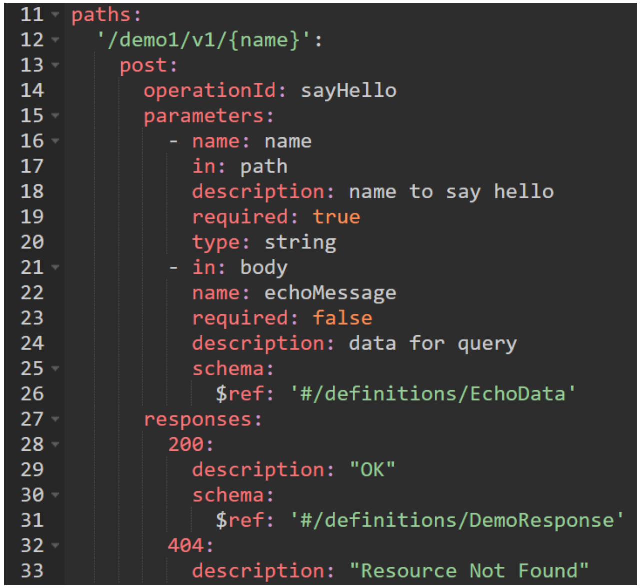
Sample SWAGGER File

Header



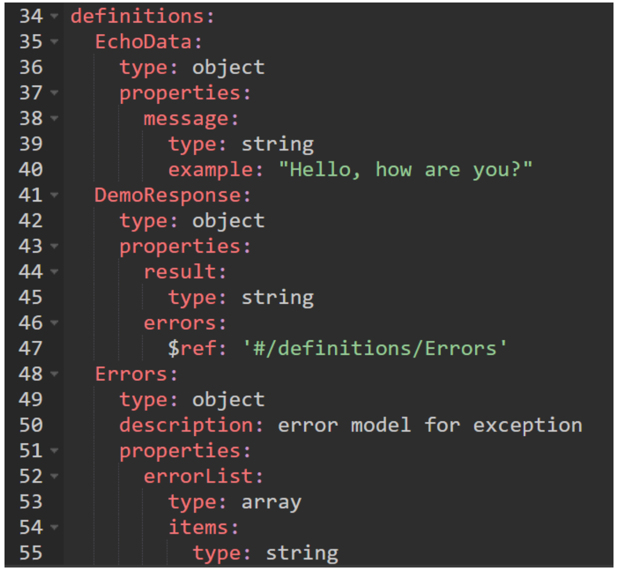
```yaml
swagger: '2.0'
info:
  title: polaris-demo1-service
  description: Polaris Demo 1 Service
  version: 1.0.0
schemes:
  - https
basePath: /
produces:
  - application/json
```

API Paths



```yaml
paths:
  '/demo1/v1/{name}':
    post:
      operationId: sayHello
      parameters:
        - name: name
          in: path
          description: name to say hello
          required: true
          type: string
        - in: body
          name: echoMessage
          required: false
          description: data for query
          schema:
            $ref: '#/definitions/EchoData'
      responses:
        200:
          description: "OK"
          schema:
            $ref: '#/definitions/DemoResponse'
        404:
          description: "Resource Not Found"
```

Definitions

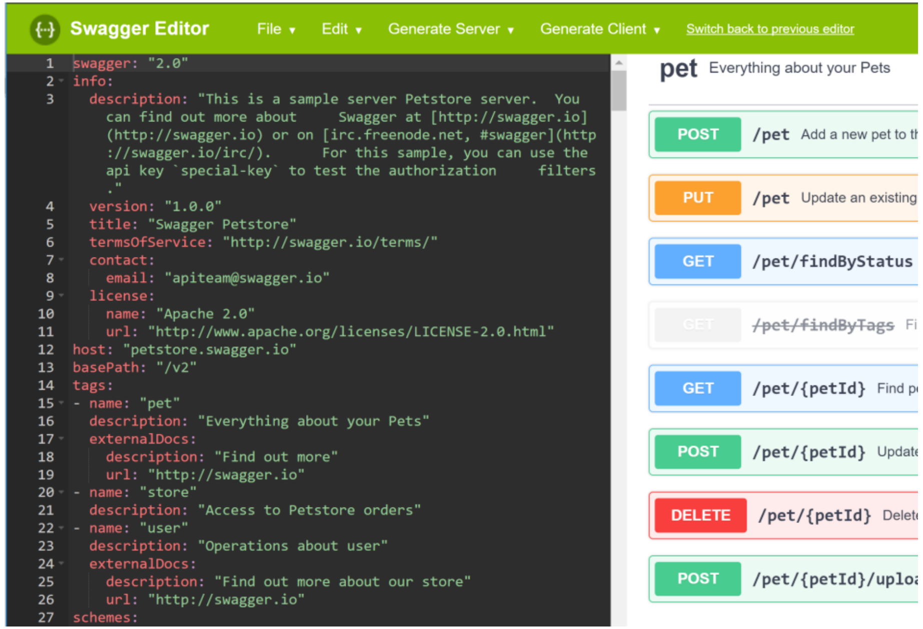


```yaml
definitions:
  EchoData:
    type: object
    properties:
      message:
        type: string
        example: "Hello, how are you?"
  DemoResponse:
    type: object
    properties:
      result:
        type: string
      errors:
        $ref: '#/definitions/Errors'
  Errors:
    type: object
    description: error model for exception
    properties:
      errorList:
        type: array
        items:
          type: string
```

SWAGGER Tools

Swagger Editor

The Swagger Editor is great for quickly getting started with the Swagger specification. It’s clean, efficient, and armed with a number of features to help you design and document your RESTful interfaces, straight out of the box.

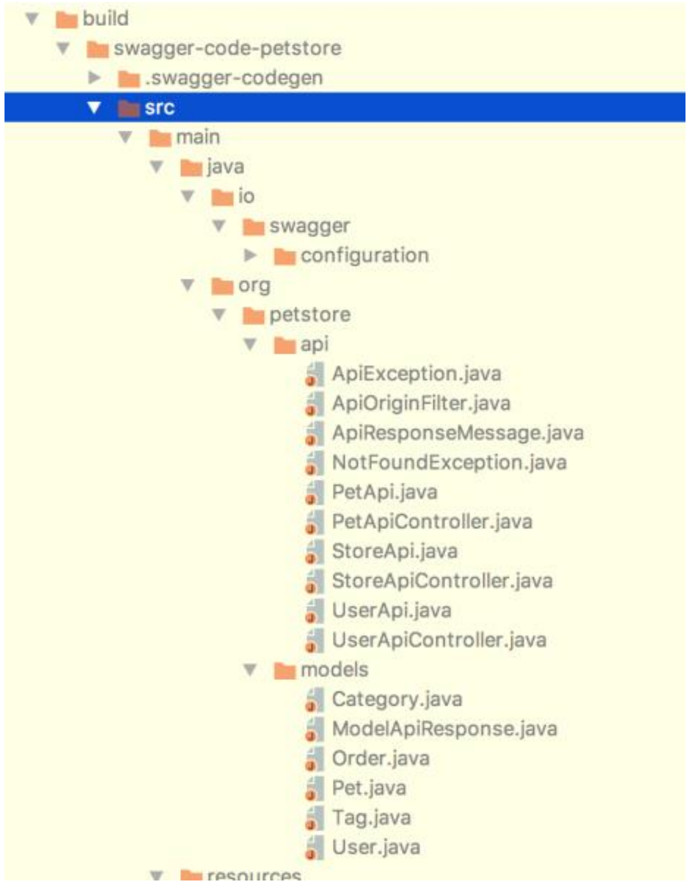

Swagger Editor
File ▾
Edit ▾
Generate Server ▾
Generate Client ▾
Switch back to previous editor
pet Everything about your Pets
POST /pet Add a new pet to th
PUT /pet Update an existing
GET /pet/findByStatus
~~GET /pet/findByTags~~ Fi
GET /pet/{petId} Find p
POST /pet/{petId} Update
DELETE /pet/{petId} Delete
POST /pet/{petId}/uploa

```
swagger: "2.0"
info:
  description: "This is a sample server Petstore server.  You
    can find out more about     Swagger at [http://swagger.io]
    (http://swagger.io) or on [irc.freenode.net, #swagger](http
    ://swagger.io/irc/).      For this sample, you can use the
    api key `special-key` to test the authorization     filters
    ."
  version: "1.0.0"
  title: "Swagger Petstore"
  termsOfService: "http://swagger.io/terms/"
  contact:
    email: "apiteam@swagger.io"
  license:
    name: "Apache 2.0"
    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
host: "petstore.swagger.io"
basePath: "/v2"
tags:
- name: "pet"
  description: "Everything about your Pets"
  externalDocs:
    description: "Find out more"
    url: "http://swagger.io"
- name: "store"
  description: "Access to Petstore orders"
- name: "user"
  description: "Operations about user"
  externalDocs:
    description: "Find out more about our store"
    url: "http://swagger.io"
schemes:
```

SWAGGER Tools

Swagger Codegen

It can simplify your build process by generating server stubs and client SDKs from your OpenAPI specification, so your team can focus better on your API’s implementation and adoption.

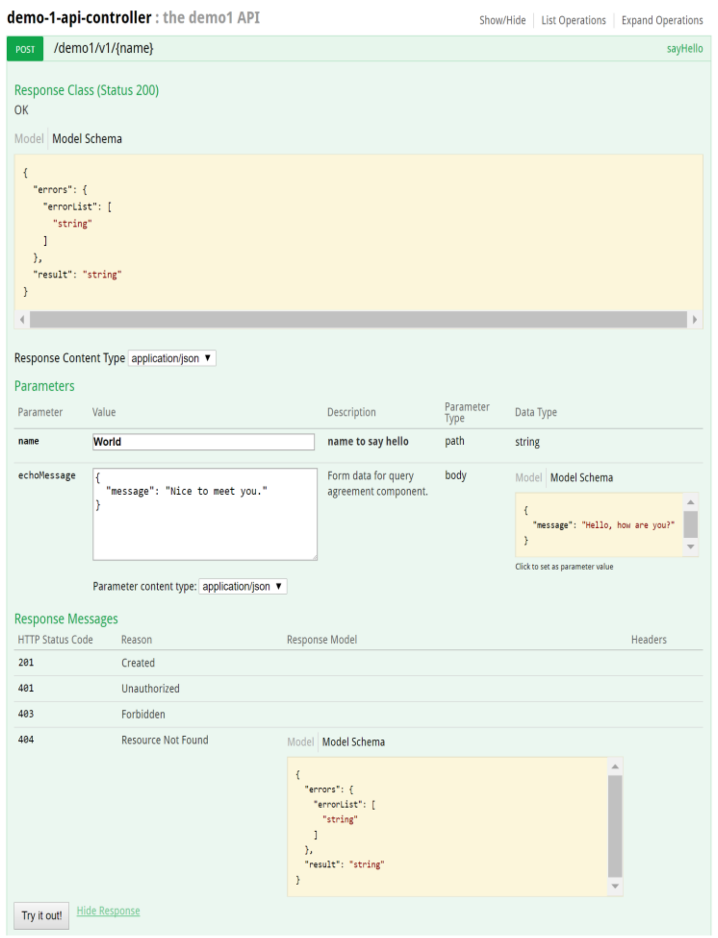


```
▼ build
  ▼ swagger-code-petstore
    ▶ .swagger-codegen
    ▼ src
      ▼ main
        ▼ java
          ▼ io
            ▼ swagger
              ▶ configuration
          ▼ org
            ▼ petstore
              ▼ api
                  ApiException.java
                  ApiOriginFilter.java
                  ApiResponseMessage.java
                  NotFoundException.java
                  PetApi.java
                  PetApiController.java
                  StoreApi.java
                  StoreApiController.java
                  UserApi.java
                  UserApiController.java
              ▼ models
                  Category.java
                  ModelApiResponse.java
                  Order.java
                  Pet.java
                  Tag.java
                  User.java
        ▼ resources
```

SWAGGER Tools

Swagger UI

It allows anyone to visualize and interact with the API’s resources without having any of the implementation logic in place. It’s automatically generated from your Swagger specification, with the visual documentation making it easy for back end implementation and client side consumption.



demo-1-api-controller : the demo1 API — Show/Hide | List Operations | Expand Operations

POST /demo1/v1/{name} — sayHello

Response Class (Status 200)
OK

Model | Model Schema

```
{
  "errors": {
    "errorList": [
      "string"
    ]
  },
  "result": "string"
}
```

Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
name	World	name to say hello	path	string
echoMessage	`{ "message": "Nice to meet you." }`	Form data for query agreement component.	body	Model | Model Schema `{ "message": "Hello, how are you?" }` Click to set as parameter value

Parameter content type: application/json

Response Messages

HTTP Status Code	Reason	Response Model	Headers
201	Created		
401	Unauthorized		
403	Forbidden		
404	Resource Not Found	Model | Model Schema	

```
{
  "errors": {
    "errorList": [
      "string"
    ]
  },
  "result": "string"
}
```

Try it out! Hide Response

App Exercise 1

Generate Spring service with Swagger codegen

ATLASSIAN Bitbucket

REST API

{JSON}

SWAGGER SMARTBEAR

App Exercise 1

Generate Spring service with Swagger codegen

1. Download “2a-swagger-codegen.zip” and extract to a folder (e.g. C:\Temp\2a-swagger-codegen)

2. Verify the files in folder “2a-swagger-codegen\swagger”.
- config.json – specify the target folders
- swagger-demo-service.yml – definition of your service
- generate-api.txt – script to generate the code
- swagger-codegen-cli-2.3.1.jar – codegen client

3. Generate the service in “swagger" folder by

```
java -jar swagger-codegen-cli-2.3.1.jar generate -i swagger-demo-service.yml -c config.json -l spring
```

App Exercise 1

Generate Spring service with Swagger codegen

4. Verify the generated directory structure

```
C:.
|   .swagger-codegen-ignore
|   config.json
|   generate-api.txt
|   output.doc
|   pom.xml
|   README.md
|   swagger-codegen-cli-2.3.1.jar
|   swagger-demo-service.yml|
\---src
    \---main
        \---java
        |   \---com
        |       \---swagger
        |           \---demo
        |               |   RFC3339DateFormat.java
        |               |   Swagger2SpringBoot.java
        |               |
        |               \---api
        |               |       ApiException.java
        |               |       ApiOriginFilter.java
        |               |       ApiResponseMessage.java
        |               |       DemoApi.java
        |               |       DemoApiController.java
        |               |       DemoApiDelegate.java
        |               |       NotFoundException.java
        |               |
        |               \---config
        |               |       CustomInstantDeserializer.java
        |               |       HomeController.java
        |               |       JacksonConfiguration.java
        |               |       SwaggerDocumentationConfig.java
        |               |
        |               \---model
        |                       GetNameResponse.java
        |
        \---resources
                application.properties
```

Summary

We covered

- Rest is a common standard to develop API
- We learned how to use Swagger to define Rest API specification
- We learned how to generate a microservice stub by using the Swagger specification

References

- Architectural Styles and the Design of Network-based Software Architectures

  - Roy Fielding

- REST APIs must be hypertext-driven

  - Roy Fielding

GENERALI

GENERALI

Thank You.

Contacts: