BEST PRACTICES FOR DESIGNING SUPERIOR RESTFUL APIs

Rohit Bhardwaj
Principal Cloud Engineer
rbhardwaj@kronos.com
Twitter: rbhardwaj1
DESIGNING RESTFUL APIS

- RESTful APIs
- Hypermedia
- Principles for designing APIs
- SAML and OAuth
- Customer driven contract
- API testing
RESTFUL APIS

- REpresentational State Transfer
- Separation of Client and Server
- Server Requests are Stateless
- Cacheable Requests
- Uniform interface
RESOURCE BASED ARCHITECTURE

Diagram:
- URI
  - has resource
  - may have representation
- resource
  - owns Person or organization
REST API Design

GET /tasks - display all tasks
POST /tasks - create a new task
GET /tasks/{id} - display a task by ID
PUT /tasks/{id} - update a task by ID
DELETE /tasks/{id} - delete a task by ID
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:body pb="http://www.acme.com/phonebook">
    <pb:GetUserDetails>
      <pb:UserID>12345</pb:UserID>
    </pb:GetUserDetails>
  </soap:Body>
</soap:Envelope>

http://www.acme.com/phonebook/UserDetails/12345
HYPERMEDIA
DESIGNING APIS
REST-FUL URIS AND NON-REST-FUL URIS

• /admin/updatebook.jsp?book=5
• /bookview.jsp?book=5
• /bookandreviews.jsp?book=5
NOUNS ARE GREAT! VERBS ARE BAD

<table>
<thead>
<tr>
<th>getCustomers</th>
<th>saveCustomers</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCustomersByName</td>
<td>getCustomersByPhone</td>
</tr>
<tr>
<td>getCustomersByContact</td>
<td>getCustomersUsingPaging</td>
</tr>
<tr>
<td>getNewCustomers</td>
<td>getCurrentCustomers</td>
</tr>
<tr>
<td>createNewCustomer</td>
<td>deleteCustomer</td>
</tr>
<tr>
<td>verifyCredit</td>
<td>removeOldInvoice</td>
</tr>
</tbody>
</table>
WHAT IS SOLUTION?

• Prefer Plurals
  
• http://..//api/Customers

• http://..//api/Employees

• http://..//api/States

• http://..//api/Orders
IDENTIFIERS KEY

- http://../api/Customers/978
- http://../api/Employees/emp123
- http://../api/States/MA
- http://../api/Orders/234
WHERE DOES VERBS GO?

<table>
<thead>
<tr>
<th>Resource</th>
<th>GET (read)</th>
<th>POST (create)</th>
<th>PUT (update)</th>
<th>DELETE (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/customers</td>
<td>Get List</td>
<td>New Customer</td>
<td>Update Batch</td>
<td>Error</td>
</tr>
</tbody>
</table>
WHERE DOES VERBS GO?

<table>
<thead>
<tr>
<th>Resource</th>
<th>GET (read)</th>
<th>POST (create)</th>
<th>PUT (update)</th>
<th>DELETE (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/customers</td>
<td>Get List</td>
<td>Create Item</td>
<td>Update Batch</td>
<td>Error</td>
</tr>
<tr>
<td>/customers/123</td>
<td>Get Item</td>
<td>Error</td>
<td>Update Item</td>
<td>Delete Item</td>
</tr>
</tbody>
</table>
**WHAT SHOULD YOU RETURN?**

<table>
<thead>
<tr>
<th>Resource</th>
<th>GET (read)</th>
<th>POST (insert)</th>
<th>PUT (update)</th>
<th>DELETE (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/customers</td>
<td>List</td>
<td>New Item</td>
<td>Status Code Only</td>
<td>Status Code Only*</td>
</tr>
<tr>
<td>/customers/123</td>
<td>Item</td>
<td>Status Code Only*</td>
<td>Updated Item</td>
<td>Status Code Only</td>
</tr>
</tbody>
</table>

*Note: The asterisk (*) indicates a condition or note related to the status code.*
<table>
<thead>
<tr>
<th>Verb</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieve a resource</td>
</tr>
<tr>
<td>HEAD</td>
<td>Get the status code as for GET, without retrieving the resource</td>
</tr>
<tr>
<td>POST</td>
<td>Create a new resource</td>
</tr>
<tr>
<td></td>
<td>Catch all usage in exception cases.</td>
</tr>
<tr>
<td>PUT</td>
<td>Update an existing resource, through complete replacement</td>
</tr>
<tr>
<td>PATCH</td>
<td>Partial update of a resource</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete an existing resource</td>
</tr>
</tbody>
</table>
### Status Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
<td>400</td>
<td>Bad Request</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
<td>401</td>
<td>Not Authorized</td>
</tr>
<tr>
<td>202</td>
<td>Accepted</td>
<td>403</td>
<td>Forbidden</td>
</tr>
<tr>
<td>302</td>
<td>Found</td>
<td>404</td>
<td>Not Found</td>
</tr>
<tr>
<td>304</td>
<td>Not Modified</td>
<td>405</td>
<td>Method Not Allowed</td>
</tr>
<tr>
<td>307</td>
<td>Temp Redirect</td>
<td>409</td>
<td>Conflict</td>
</tr>
<tr>
<td>308</td>
<td>Perm Redirect</td>
<td>500</td>
<td>Internal Error</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Real Story</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
<td>----------------------</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
<td>&quot;It Worked&quot;</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
<td>&quot;You did bad&quot;</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>Internal Error</td>
<td>&quot;We did bad&quot;</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
<td></td>
<td></td>
</tr>
<tr>
<td>304</td>
<td>Not Modified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td></td>
<td></td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HOW TO SATISFY ASSOCIATIONS?

• http://../api/Customers/978/Orders
• http://../api/Employees/emp123/Address
• http://../api/States/MA/Cities
• http://../api/Orders/234/Details
RETURN SAME OBJECT

• http://..//api/States/MA/Cities

• http://..//api/Cities
SAME OBJECT MULTI ASSOCIATIONS

- http://../api/Customers/978/Orders
- http://../api/Customers/978/Shipments
- http://../api/Customers/978/Invoices
WHAT IF YOU HAVE COMPLEX ASSOCIATION?

- http://../api/Customers/state=MA
- http://../api/Customers/state=MA&City=Chelmsford
HOW TO FORMAT RESULTS?

• Use content Negotiation

• Accept header

GET /api/customers/987 HTTP/1.1

Accept: application/json, text/xml

Host: localhost:9876
## MIME Types

<table>
<thead>
<tr>
<th>Type</th>
<th>MIME Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON</td>
<td>application/json</td>
</tr>
<tr>
<td>XML</td>
<td>text/xml</td>
</tr>
<tr>
<td>JSONP*</td>
<td>application/javascript</td>
</tr>
<tr>
<td>RSS</td>
<td>application/xml+rss</td>
</tr>
<tr>
<td>ATOM</td>
<td>application/xml+xml</td>
</tr>
</tbody>
</table>
FORMATING

• http://.../api/Customers?format=json

• http://.../api/Customers.json

• http://.../api/Customers
  format=jsonp&callback=foo
JSON EXAMPLE

```json
{
    "id": 978,
    "firstName": "John",
    "lastName": "Smith",
    "age": 45,
    "address": {
        "streetAddress": "21 Chelmsford Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10293"
    },
    "phoneNumber": [
        {
            "type": "home",
            "number": "987 555-1234"
        },
        {
            "type": "fax",
            "number": "938 555-3733"
        }
    ]
}
```
MEMBER NAMES

- Use camelCasing - Objects
HANDLING ERRORS

- Learn writing code through errors
- Test Driven Development
- Troubleshooting and resolving issues
ERROR HANDLING: APPROACH I

Facebook
HTTP Status Code: 200
{"type" : "OauthException", "message":"(#803) Some of the aliases you requested do not exist: foo.bar"}

Twilio
HTTP Status Code: 401
{"status" : "401", "message":"Authenticate","code":20003, "more info": "http://www.twilio.com/docs/errors/20003"}

SimpleGeo
HTTP Status Code: 401
{"code" : 401, "message": "Authentication Required"}
BE VERBOSE AND USE PLAIN LANGUAGE DESCRIPTIONS

• {"developerMessage" : "Verbose, plain language description of the problem for the app developer with hints about how to fix it.", "userMessage" : "Pass this message on to the app user if needed.", "errorCode" : 12345, "more info" : "http://dev.teachdogrest.com/errors/12345"}
DESIGNING COLLECTIONS

```json
{
  "numberofResults": 6,
  "results": [
    {
      "id": 978,
      "firstName": "John",
      "lastName": "Smith",
      "age": 45,
      "address": {
        "streetAddress": "21 Chelmsford Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10293"
      },
      "phoneNumber": [
        {
          "type": "home",
          "number": "987 555-1234"
        },
        {
          "type": "fax",
          "number": "938 555-3733"
        }
      ]
    }
  ]
}
```
ENTITY TAGS (ETAGS)

• Header to support smart server caching

• Strong and Weak Caching
ENTITY TAGS (ETAGS)

• Header to support smart server caching

• Strong and Weak Caching

HTTP/1.1 200 OK
Last-Modified: Tue, 12 Dec 2015 03:03:59 GMT
ETag: "10c24bc-4ab-457wer1f"
Content-Length: 12195

GET /i/yahoo.gif HTTP/1.1
Host: us.yimg.com
If-Modified-Since: Tue, 12 Dec 2015 03:03:59 GMT
If-None-Match: "10c24bc-4ab-457e1c1f"
HTTP/1.1 304 Not Modified

Use 304 to indicate object not modified
ENTITY TAGS (ETAGS)

• Header to support smart server caching

• Weak Caching: Objects are semantically same

HTTP/1.1 200 OK
Last-Modified: Tue, 12 Dec 2015 03:03:59 GMT
ETag: W/"10c24bc-4ab-457wer1f"
Content-Length: 12195
PAGING TO RETURN COLLECTIONS

• Prevent to return huge amount of data

• Large result sets performance

• Query String parameter to narrow down

• GET /customers?offset=10&limit=5
## Filtering

<table>
<thead>
<tr>
<th>Query Parameter</th>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>where</td>
<td>Filtering criteria</td>
<td><code>where=&quot;from=03/30/2016&quot;</code></td>
</tr>
<tr>
<td>index</td>
<td>Start index</td>
<td><code>index=100</code></td>
</tr>
<tr>
<td>count</td>
<td>Number of items returned</td>
<td><code>count=50</code></td>
</tr>
<tr>
<td>sort</td>
<td>Sorting direction. It can be ascending or descending</td>
<td><code>sort=ascending</code></td>
</tr>
<tr>
<td>by</td>
<td>Sorting element name</td>
<td><code>by=creation_time</code></td>
</tr>
<tr>
<td>{attr}</td>
<td>Short form. Shorthand for <code>where=&quot;%{attr}%=&quot;...&quot;</code></td>
<td><code>from=01/20/2013</code></td>
</tr>
</tbody>
</table>
FILTERING

- GET /customer/v1/user/orders?
  from=01/12/2016&to=01/12/2016&sort=ascending&by=
  …&index=0&count=20

- The equivalent long form is:

- GET /customer/v1/user/orders?where="from=01/12/2016
  and to=01/12/2016"&sort=ascending&by=...
  index=0&count=20
BATCH CRUD OPERATIONS

- multi_create
- multi_read
- multi_update
- multi_delete
Language

- Support ISO 639 Language codes

Currency

- ISO 4217: Currency codes
118NL10N

- Numbers: Support Decimal XML Schema
  - +2, -123.45, 0.34, 0

- Floats and doubles: IEEE 754-1985
  - -1E4, 1234.43233E10, 12.78e-3, 13, -0, 0
I8NL10N

- Date/time: ISO-8601 standard
  - Date: YYYY-MM-DD -> 2011-12-03
  - Date and time: YYYY-MM-DD’T’HH:MM:SS -> 2011-12-03T10:15:30
  - Time: HH:MM:SS -> 10:15:30
Chatty APIs: Bandwidth Aware
Reduce chattiness

9 network calls collapsed into 1. WAN network latency cost paid only once.

Client logic pushed to server. 20+ redundant calls removed.

http://techblog.netflix.com/2013/01/optimizing-netflix-api.html
PAGING

• {
  "totalNumResults": 9999,
  "nextPage": "http://.../api/customers/?page=5",
  "prevPage": "http://.../api/customers/?page=1",
  "results": [...]
}

• }
PARTIAL ITEMS

• Query string to return less

• https://www.googleapis.com/demo/v1?key=YOUR-API-KEY

• https://developers.google.com/drive/v3/web/performance#partial-response
API

- https://www.googleapis.com/demo/v1?key=YOUR-API-KEY

```json
{
  "kind": "demo",
  ...
  "items": [
    {
      "title": "First title",
      "comment": "First comment.",
      "characteristics": {
        "length": "short",
        "accuracy": "high",
        "followers": ["Jo", "Will"],
      },
      "status": "active",
      ...
    },
    {
      "title": "Second title",
      "comment": "Second comment.",
      "characteristics": {
        "length": "long",
        "accuracy": "medium"
      },
      "status": "pending",
      ...
    },
    ...
  ]
}```
PARTIAL ITEMS

- https://www.googleapis.com/demo/v1?key=YOUR-API-KEY&fields=kind,items(title,characteristics/length)

  200 OK

  ```json
  {
    "kind": "demo",
    "items": [
      {
        "title": "First title",
        "characteristics": {
          "length": "short"
        }
      },
      {
        "title": "Second title",
        "characteristics": {
          "length": "long"
        }
      },
      ...
    ]
  }
  ```
NON RESOURCE APIS

• get support only

• Should be functional API

• http://.../api/calculatePrice?state=MA&total=62.95

• http://.../api/pingServer?isNotify=true

• http://.../api/weatherForecast?isSummary=true&state="MA"
SUMMARY

• Do not surprise users
• Follow patterns available
• Caching and use Etags
• Use of Partial items
• Non Resource APIs
VERSIONING REST APIS

1.5.2

Major
Minor
Patch
WHY VERSIONING?

- Publishing an API is not a trivial move
- Users/Customers rely on the API not changing
- But requirements will change
- Need a way to evolve the API without breaking existing clients
- API Versioning isn't Product Versioning Don’t tie them together
• Thy Shall not break existing Customers
EXAMPLES

• Uri Path
  • Tumbler
    • http://api.tumblr.com/v2/user/

• Uri Parameter
  • Netflix
    • http://api.netflix.com/catalog/titles/series/70023522?v=1.5

• Content Negotiation
  • GitHub API
    • ContentType: application/vnd.github.1.param+json

• Request Header
  • Azure
    • x-ms-version: 2011-08-18
VERSIONING IN THE URI PATH

• Using Part of Your Path to Version
• Allows you to drastically change the API
• Everything below the version is open to change

• Twitter
  • http://api.twitter.com/1/users/show/noradio.xml

• Atlassian
  • http://host:port/context/rest/upm/1/plugin/a-plugin-key
<table>
<thead>
<tr>
<th>Service</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digg</td>
<td><a href="http://services.digg.com/2.0/comment.bury">http://services.digg.com/2.0/comment.bury</a></td>
</tr>
<tr>
<td>Delicious</td>
<td><a href="https://api.del.icio.us/v1/posts/update">https://api.del.icio.us/v1/posts/update</a></td>
</tr>
<tr>
<td>Last FM</td>
<td><a href="http://ws.audioscrobbler.com/2.0/">http://ws.audioscrobbler.com/2.0/</a></td>
</tr>
<tr>
<td>LinkedIn</td>
<td><a href="http://api.linkedin.com/v1/people/~/connections">http://api.linkedin.com/v1/people/~/connections</a></td>
</tr>
<tr>
<td>Foursquare</td>
<td><a href="https://api.foursquare.com/v2/venues/40a55d80f964a52020f31ee3?oauth_token=XXX&amp;v=YYYYMMDD">https://api.foursquare.com/v2/venues/40a55d80f964a52020f31ee3?oauth_token=XXX&amp;v=YYYYMMDD</a></td>
</tr>
<tr>
<td>Freebase</td>
<td><a href="https://www.googleapis.com/freebase/v1/search?query=nirvana&amp;indent=true">https://www.googleapis.com/freebase/v1/search?query=nirvana&amp;indent=true</a></td>
</tr>
</tbody>
</table>
VERSIONING IN THE URI PATH

• Pro(s):
  • Simple to segregate old APIs for backwards compatibility

• Con(s):
  • Requires lots of client changes as you version
    E.g. version # has to change in every client

  • Increases the size of the URI surface area you have to support
VERSIONING IN THE URI PARAMETERS

• http://api.netflix.com/catalog/titles/series/70023522?v=1.5
• http://..//api/Employees
• http://..//api/Employees?v=3.1
VERSIONING IN THE URI PARAMETERS

• Pro(s):
  • Without version, users always get latest version of API
  • Little client change as versions mature

• Con(s):
  • Can surprise developers with unintended changes
HYPERMEDIA CONTROL

```
{
  "users": [
    {
      "displayName": "Jane Doe",
      "links": [
        { "rel": "user", "href": "/users/1234" },
        { "rel": "friends", "href": "/users/1234/friends" }
      ]
    },
    {
      "displayName": "John Doe",
      "links": [
        { "rel": "user", "href": "/users/5678" },
        { "rel": "friends", "href": "/users/5678/friends" }
      ]
    }
  ]
}
```
{"users": [  
   {  
      "displayName": "Jane Doe",
      "links": [  
         { "rel": "user", "href": "/users/1234" },  
         { "rel": "userV2", "href": "/users/v2/1234" },  
         { "rel": "friends", "href": "/users/1234/friends" }  
      ]  
   },  
   {  
      "displayName": "John Doe",
      "links": [  
         { "rel": "user", "href": "/users/5678" },  
         { "rel": "userV2", "href": "/users/v2/5678" },  
         { "rel": "friends", "href": "/users/5678/friends" }  
      ]  
   }  
],}
{
    "users": [
    {
        "displayName": "Jane Doe",
        "links": [
            { "rel": "user", "href": "/users/1234" },
            { "rel": "userV2", "href": "/users/v2/1234" },
            { "rel": "userV3", "href": "/users/v3/1234" },
            { "rel": "userV4", "href": "/users/v4/1234" },
            { "rel": "userV5", "href": "/users/v5/1234" },
            { "rel": "friends", "href": "/users/1234/friends" },
            { "rel": "friendsV2", "href": "/users/1234/V2/friends" }
        ]
    },
    {
        "displayName": "John Doe",
        "links": [
            { "rel": "user", "href": "/users/5678" },
            { "rel": "userV2", "href": "/users/v2/5678" },
            { "rel": "userV3", "href": "/users/v3/5678" },
            { "rel": "userV4", "href": "/users/v4/5678" },
            { "rel": "userV5", "href": "/users/v5/5678" },
            { "rel": "friends", "href": "/users/5678/friends" },
            { "rel": "friendsV2", "href": "/users/5678/V2/friends" }
        ]
    }
    ]
}
VERSIONING WITH CONTENT NEGOTIATION

- GET /api/employee/978
  HOST: http://.../
  Accept: application/myapp.v3.employee

- GET /api/employee/123
  HOST: http://.../
  Accept: application/myapp.v3.employee.json
GET /customer/123 HTTP/1.1

Accept: application/vnd.company.myapp.customer-v1+xml

HTTP/1.1 200 OK

Content-Type: application/vnd.company.myapp.customer-v1+xml

<customer>
  <name>Neil Armstrong</name>
</customer>

NEWER CLIENTS MAKE A DIFFERENT CALL

---
GET /customer/123 HTTP/1.1
Accept: application/vnd.company.myapp.customer-v2+xml

HTTP/1.1 200 OK
Content-Type: application/vnd.company.myapp.customer-v2+xml

<customer>
  <firstName>Neil</firstName>
  <lastName>Armstrong</lastName>
  <salutation>Mr.</salutation>
</customer>
VERSIONING WITH CONTENT NEGOTIATION

- Pro(s):
  - Packages API and Resource Versioning in one
  - Removes versioning from API so clients don't have to change

- Con(s):
  - Adds complexity - adding headers isn't easy on all platforms
  - Can encourage increased versioning which causes more code churning
VERSIONING WITH CUSTOM HEADER

GET /api/customer/123

HOST: http://...

x-MyApp-Version: 2.1
GET /api/customer/123

HOST: http://...

x-MyApp-Version: 2013-08-13
VERSIONING WITH CUSTOM HEADER

• Pro(s):
  • Separates Versioning from API call signatures
  • Not tied to resource versioning (e.g. Content Type)

• Con(s):
  • Adds complexity - adding headers isn't easy on all platforms
WHICH ONE TO CHOOSE?

• There is no easy answer

• Versioning with Content Negotiation and Custom Headers are popular now

• Versioning with URI Components are more common

• These are easier to implement

• But you should version from the first release of your API
YOU MUST VERSION API!
API SECURITY
SECURITY APIS

• Threats
• Protecting APIs
• Authentication
• API Keys: How they work?
• User Authentication and Authorization
• SAML
• Understanding OAuth
WHY NEED TO SECURE?

❖ US laws and Regulations

❖ International laws and regulation
Privacy: KPMG Data life cycle
Data Security lifecycle

- **Create**:
  - Classify
  - Assign Rights

- **Store**:
  - Access Controls
  - Encryption
  - Rights Management
  - Content Discovery

- **Use**:
  - Activity Monitoring and Enforcement
  - Rights Management
  - Logical Controls
  - Application Security

- **Share**:
  - CMP (DLP)
  - Encryption
  - Logical Controls
  - Application Security

- **Archive**:
  - Encryption
  - Asset Management

- **Destroy**:
  - Crypto-Shredding
  - Secure Deletion
  - Content Discovery
WHY PROTECT APIS?

You can’t use network layer protection (firewall, SSL, IDS, hardening) to stop or detect application layer attacks.
• Server-to-Server Authentication
  • API Keys and Shared Secrets
• User Proxy Authentication
  • OAuth
• Direct User Authentication
  • Cookies or token
• Credential:
  • A fact that describe an identity

• Authentication:
  • Validate a set of credentials to identify an entity (whether virtual or actual)

• Authorization:
  • Verification that an entity has rights to access a resource or action
Confidentiality: Symmetric Encryption: Efficient

Data Accountability and Trust Act, 2006

https://keybase.io/triplesec/
http://code.google.com/p/jscryptolib/
https://keybase.io/triplesec/

In-Browser Magical Demo

This is to test message.

key

- pbkdf2 (pass 1) 32/32
- scrypt 65536/65536
- pbkdf2 (pass 2) 6/6
- aes 16/16
- twofish 12/12
- openssl 6/6
Confidentiality: Asymmetric Encryption

http://www.securecottage.com/demo/rsa2.html
http://support.persits.com/encrypt/demo_text.asp
http://support.persits.com/pdf/demo_encrypt.aspx
AUTHENTICATING USERS

- Within system
- OAuth
OAuth
Open Authentication (OAuth)

http://brentertainment.com/oauth2/
DOCUMENTATION

My super cool API
Created by Guillaume Lafoge
Contact the developer

My super cool API Data

GET /cars/  Loads a list of Car
POST /cars/  Adds a Car
GET /cars/{carid}  Loads a Car

Implementation Notes
Loads a Car

Response Class (Status 200)
Model  Model Schema
{
  "id": "sample id",
  "name": "sample name"
}

Response Content Type  application/json

Parameters
Parameter  Value
carid  [required]

Description  Parameter Type  Data Type
Identifier of the Car  path  string

Response Messages
HTTP Status Code  Reason  400 status response
Try it out!
# OAuth and Swagger Demo

## Swagger Petstore

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. For this sample, you can use the api key `special-key` to test the authorization filters.

Find out more about Swagger

- [http://swagger.io](http://swagger.io)
- Contact the developer
- [Apache 2.0](http://www.apache.org/licenses/LICENSE-2.0)

### pet: Everything about your Pets

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/pet</td>
<td>Add a new pet to the store</td>
</tr>
<tr>
<td>PUT</td>
<td>/pet</td>
<td>Update an existing pet</td>
</tr>
<tr>
<td>GET</td>
<td>/pet/findByStatus</td>
<td>Finds Pets by status</td>
</tr>
<tr>
<td>GET</td>
<td>/pet/findByTags</td>
<td>Finds Pets by tags</td>
</tr>
<tr>
<td>DELETE</td>
<td>/pet/{petId}</td>
<td>Deletes a pet</td>
</tr>
<tr>
<td>GET</td>
<td>/pet/{petId}</td>
<td>Find pet by ID</td>
</tr>
<tr>
<td>POST</td>
<td>/pet/{petId}</td>
<td>Updates a pet in the store with form data</td>
</tr>
</tbody>
</table>
JSON WEB TOKENS

https://jwt.io/introduction/

Compact and self-contained way for securely transmitting information between parties as a JSON object
1. POST /users/login with username and password
   3. Returns the JWT to the Browser

2. Creates a JWT with a secret

4. Sends the JWT on the Authorization Header
   6. Sends response to the client

5. Check JWT signature. Get user information from the JWT
Use an API gateway

- Single entry point
- API Gateway
  - Client specific APIs
  - Protocol translation
  - Traditional server-side web application
  - Browser/Native App
  - View
  - Controller
  - Model

- REST
  - Product Info service
- REST
  - Recommendation Service
- AMQP
  - Review service

http://microservices.io/patterns/apigateway.html
HYPERMEDIA: LINKS IN API

```xml
<search>
  <items>
    <sku>555-ABC</sku>
    <quantity>1</quantity>
  </items>
  <id>156</id>
  <status>In Stock</status>
  <onHand>25</onHand>
  <links>
    <link rel="order" url="http://api.mycompany.com/orders/156" />  
    <link rel="invoice" url="http://api.mycompany.com/payments/156" />  
    <link rel="payment" url="http://api.mycompany.com/invoices/156" />  
    <link rel="return" url="http://api.mycompany.com/returns/156" />  
  </links>
</search>
```
```json
{
  "items": [
    {
      "name": "Neil Young",
      "created": "2013-04-29T13:47:59.918Z",
      "_links": {
        "self": {
          "href": "http://localhost:8080/api/artist/1"
        }
      }
    },
    {
      "name": "Joe Strummer",
      "created": "2013-04-29T13:47:59.918Z",
      "_links": {
        "self": {
          "href": "http://localhost:8080/api/artist/2"
        }
      }
    }
  ],
  "_links": {
    "parent": {
      "href": "http://localhost:8080/api"
    }
  }
}
```
RETURNING MORE ACTIONS

```json
{
  "Department": "Enforcement",
  "Id": "123",
  "Links": [
    {
      "Rel": "GetDetails",
      "Url": "/api/employees/56789"
    },
    {
      "Rel": "Fire",
      "Url": "/api/employees/56789"
    }
  ],
  "Name": "John Q Law"
}
```
HAL: HYPERTEXT APPLICATION LANGUAGE

- HAL is a simple format that gives a consistent and easy way to hyperlink between resources in your API.

Amazon Chooses HAL Media Type for AppStream API

http://www.infoq.com/news/2014/03/amazon-hal-appstream
{  "_links": {"self": { "href": "/orders" },  "curies": [{  "name": "ea",  "href": "http://example.com/docs/rels/{rel}"},  "templated": true ]},  "ea:find": {  "href": "/orders{id}"},  "templated": true,  "ea:admin": [{  "href": "/admins/2",  "title": "Fred" }, {  "href": "/admins/5",  "title": "Kate" }],  "currentlyProcessing": 14,  "shippedToday": 20,  "_embedded": {  "ea:order": [{"_links": {  "self": { "href": "/orders/123" },  "ea:basket": { "href": "/baskets/98712" },  "ea:customer": { "href": "/customers/7809" } },  "total": 30.00,  "currency": "USD",  "status": "shipped" }, {"_links": {  "self": { "href": "/orders/124" },  "ea:basket": { "href": "/baskets/97213" },  "ea:customer": { "href": "/customers/12369" } },  "total": 20.00,  "currency": "USD",  "status": "processing" }]}
}
GET /stockLevels

{ stockLevels: [ ... ] }

API MANAGEMENT

Secure

Analyze

Manage

Scale

Create
A Checklist for Every API Call

Managing the Complete API Lifecycle
TEST REST APIS
POSTMAN DEMO
DESIGNING RESTFUL APIS

• RESTful APIs
• Hypermedia
• Principles for designing APIs
• SAML and OAuth
• Customer driven contract
• API testing