Beautiful REST+JSON APIs
DevNexus 2016

Les Hazlewood @lhazlewood
Apache Shiro Project Chair
Expert Group Member, JEE Application Security
CTO, Stormpath, stormpath.com

Stormpath.com

- User Management API for Developers
- Password security
- Authentication and Authorization
- LDAP & Active Directory Cloud Sync
- Instant-on, scalable, and highly available
- Free for developers

Outline

- APIs, REST & JSON
- REST Fundamentals
- Design

Base URL
Versioning
Resource Format
Return Values
Content Negotiation
References (Linking)
Pagination
Query Parameters
Create/Update
Search
Associations

Errors
IDs
Method Overloading
Resource Expansion
Partial Responses
Caching & Etags
Security
Multi Tenancy
Maintenance
Batch Operations

APIs

- Applications
- Developers
- Pragmatism over Ideology
- Adoption
- Scale

Why REST?

- Scalability
- Generality
- Independence
- Latency (Caching)
- Security
- Encapsulation

Why JSON?

- Ubiquity
- Simplicity
- Readability
- Scalability
- Flexibility

HATEOAS

- **H**ypermedia
- **A**s
- **T**he
- **E**ngine
- **O**f
- **A**pplication
- **S**tate

REST Is Easy

REST Is *&@#$! Hard

(for providers)

REST *can* be easy

(if you follow some guidelines)

Example Domain: Stormpath

- Applications
- Directories
- Accounts
- Groups
- Associations
- Workflows

Fundamentals

Resources

Nouns, not Verbs

Coarse Grained, not Fine Grained

Architectural style for use-case scalability

What If?

/getAccount

/createDirectory

/updateGroup

/verifyAccountEmailAddress

What If?

/getAccount
/getAllAccounts
/searchAccounts
/createDirectory
/createLdapDirectory
/updateGroup
/updateGroupName
/findGroupsByDirectory
/searchGroupsByName
/verifyAccountEmailAddress
/verifyAccountEmailAddressByToken
…

Smells like bad RPC. DON'T DO THIS.

Keep It Simple

The Answer

Fundamentally two types of resources:

Collection Resource

Instance Resource

Collection Resource

/applications

Instance Resource

```
/applications/a1b2c3
```

Behavior

- GET
- PUT
- POST
- DELETE
- HEAD

Behavior

`POST, GET, PUT, DELETE`

≠ 1:1

Create, Read, Update, Delete

Behavior

As you would expect:

`GET` = Read

`DELETE` = Delete

`HEAD` = Headers, no Body

Behavior

Not so obvious:

`PUT` and `POST` can *both* be used for Create *and* Update

PUT for Create

Identifier is known by the client:

```
PUT /applications/clientSpecifiedId

{
   …
}
```

PUT for Update

Full Replacement

```
PUT /applications/existingId
{
  “name”: “Best App Ever”,
  “description”: “Awesomeness”
}
```

PUT

Idempotent

POST as Create

On a parent resource

```
POST /applications

{
   “name”: “Best App Ever”
}

Response:

201 Created
Location: https://api.stormpath.com/applications/a1b2c3
```

POST as Update

On instance resource

```
POST /applications/a1b2c3

{
  “name”: “Best App Ever. Srsly.”
}

Response:

200 OK
```

POST

NOT Idempotent

Media Types

- Format Specification + Parsing Rules
- Request: `Accept` header
- Response: `Content-Type` header

- `application/json`
- `application/foo+json`
- `application/ion+json;v=1`
- …

Design Time!

Base URL

http(s)://foo.io

vs

http://www.foo.com/dev/service/api/rest

http(s)://foo.io

Rest Client
vs
Browser

Versioning

URL

```
https://api.stormpath.com/v1
```

vs.

Media-Type

```
application/json
application/foo+json;v=1
```

Resource Format

Media Type

```
Content-Type: application/json
```

When time allows:

```
application/foo+json
application/foo2+json;bar=baz
...
```

Media Type

Don't go overboard!

Media Type != Schema!

Most only need 1 *maybe* 2 media types:

- instance resource
- collection resource

```
application/ion+json
application/ion+json;bar=baz
...
```

camelCase

'JS' in 'JSON' = JavaScript

`myArray.forEach`
Not `myArray.for_each`

`account.givenName`
Not `account.given_name`

Underscores for property/function names are unconventional for JS. Stay consistent.

Date/Time/Timestamp

There's already a standard. Use it: ISO 8601

Example:

```
{
  …,
  "createdAt": "2013-07-10T18:02:24.343Z",
  ...
}
```

Use UTC!

createdAt / updatedAt

createdAt / updatedAt

Most people will want this at some point

```
{
  …,
  “createdAt”: “2013-07-10T18:02:24.343Z”,
  “updatedAt”: “2014-09-29T07:02:48.761Z”
}
```

Use UTC!

Response Body

GET obvious

What about POST?

Return the representation in the response when feasible.

Add override (`?_body=false`) for control

Content Negotiation

Header

- `Accept` header
- Header values comma delimited
- `q` param determines precedence, defaults to 1, then conventionally by list order

```
GET /applications/a1b2c3
Accept: application/json,
text/plain;q=0.8
```

Resource Extension

`/applications/a1b2c3.json`

`/applications/a1b2c3.csv`

…

Conventionally overrides `Accept` header

HREF

- Distributed Hypermedia is paramount!
- **Every accessible Resource has a canonical unique URL**
- Replaces IDs (IDs exist, but are opaque).
- Critical for linking, as we'll soon see

Instance w/ HREF (v1)

```
GET /accounts/x7y8z9


200 OK

{
  “href”: “https://api.stormpath.com/v1/accounts/x7y8z9”,
  “givenName”: “Tony”,
  “surname”: “Stark”,
  ...
}
```

Resource References
aka ‘Linking’
(v1)

- Hypermedia is paramount.
- Linking is fundamental to scalability.

- Tricky in JSON
- XML has it (XLink), JSON doesn't
- How do we do it?

Instance Reference (v1)

```
GET /accounts/x7y8z9
```

```
200 OK
{
  “href”: “https://api.stormpath.com/v1/accounts/x7y8z9”,
  “givenName”: “Tony”,
  “surname”: “Stark”,
  …,
  “directory”: ????
}
```

Instance Reference (v1)

```
GET /accounts/x7y8z9
```

```
200 OK
{
  “href”: “https://api.stormpath.com/v1/accounts/x7y8z9”,
  “givenName”: “Tony”,
  “surname”: “Stark”,
  …,
  “directory”: {
    “href”: “https://api.stormpath.com/v1/directories/g4h5i6”
  }
}
```

Collection Reference (v1)

```
GET /accounts/x7y8z9
```

```
200 OK
{
  “href”: “https://api.stormpath.com/v1/accounts/x7y8z9”,
  “givenName”: “Tony”,
  “surname”: “Stark”,
  …,
  “groups”: {
    “href”: “https://api.stormpath.com/v1/accounts/x7y8z9/groups”
  }
}
```

Linking v2 (recommended)

Instance HREF (v2)

```
GET /accounts/x7y8z9
```

```
200 OK
{
  "meta": {
    "href": "https://api.stormpath.com/v1/accounts/x7y8z9",
  },
  "givenName": "Tony",
  "surname": "Stark",
  …
}
```

Instance Reference (v2)

```
GET /accounts/x7y8z9
```

```
200 OK
{
  "meta": { ... },
  "givenName": "Tony",
  "surname": "Stark",
  …,
  "directory": {
    "meta": {
      "href": https://api.stormpath.com/v1/directories/g4h5i6
    }
  }
}
```

Collection Reference (v2)

```
GET /accounts/x7y8z9
```

```
200 OK
{
  "meta": { ... },
  "givenName": "Tony",
  "surname": "Stark",
  …,
  "groups": {
    "meta": {
      "href": "https://api.stormpath.com/v1/accounts/x7y8z9/groups",
      "rel": ["collection"]
    }
  }
}
```

Reference Expansion

(aka Entity Expansion, Link Expansion)

Account and its Directory?

GET /accounts/x7y8z9?expand=directory

```
200 OK
{
  "meta": {...},
  "givenName": "Tony",
  "surname": "Stark",
  …,
  "directory": {
    "meta": { ... },
    "name": "Avengers",
    "description": "Hollywood's plan for more $",
    "createdAt": "2012-07-01T14:22:18.029Z",
    …
  }
}
```

Partial Representations

```
GET
/accounts/x7y8z9?fields=givenName,surname,
directory(name)
```

Collections!

Collections

- A first class resource ‘citizen’
- Own href / metadata
- Own properties
- Different from all other collections

GET /accounts/x7y8z9/groups

```
200 OK
{
  "meta": { ... },
  "offset": 0,
  "limit": 25,
  "size": 289,
  "first": { "meta":{"href": "…/accounts/x7y8z9/groups?offset=0"}},
  "previous": null,
  "next": { "meta":{"href": "…/accounts/x7y8z9/groups?offset=25"}},
  "last": { "meta":{"href": "…"}},
  "items": [
    {
      "meta": { "href": "…", ...}
    },
    …
  ]
}
```

Pagination

Collection Resource supports query params:

- Offset
- Limit

```
…/applications?offset=50&limit=25
```

GET /accounts/x7y8z9/groups

```
200 OK
{
  “meta”: { ... },
  “offset”: 0,
  “limit”: 25,
  “first”: { “meta”:{“href”: “…/accounts/x7y8z9/groups?offset=0”}},
  “previous”: null,
  “next”: { “meta”:{“href”: “…/accounts/x7y8z9/groups?offset=25”}},
  “last”: { “meta”:{“href”: “…”}},
  “items”: [
    {
      “meta”: { “href”: “…”, ...}
    },
    {
      “meta”: { “href”: “…”, ...}
    },
    …
  ]
}
```

Sorting

```
GET .../accounts?
    orderBy=surname,givenName%20desc
```

Search

“Find all accounts with a ‘company.com’ email address that can login to a particular application”

```
GET /applications/x7y8z9/accounts?email=*company.com

200 OK
{
  “meta”: { ... },
  “offset”: 0,
  “limit”: 25,
  “first”: { “meta”:{
      “href”:“/applications/x7y8z9/accounts?email=*company.com&offset=0”}
  },
  “previous”: null,
  “next”: { “meta”:{
      “href”: “/applications/x7y8z9/accounts?email=*company.com&offset=25”}
  },
  “last”: { “meta”:{“href”: “…”}},
  “items”: [
    { “meta”: { “href”: “…”, ...} },
    { “meta”: { “href”: “…”, ...} },
    …
  ]
}
```

Search cont'd

- Filter search

  .../accounts?**q**=some+value

- Attribute Search

  .../accounts?**surname**=Joe&**email**=*company.com

Search cont'd

- Starts with

```
?email=joe*
```

- Ends with

```
?email=*company.com
```

- Contains

```
?email=*foo*
```

Search cont'd

- Range queries

"all accounts created between September 1st and the 15th"

```
.../accounts?createdAt=[2014-09-01,2014-09-15]
```

Creating And Updating

Remember HATEOS?

“Let me go read the docs to figure out how to POST”

NO! This is not HATEOS.

How do browsers work?

Forms

Forms: Create / Update

```
{
  "foo": "bar",
  "baz": "boo",
  ...

  "createAccount": {
    "meta": {
      "href": "…/v1/users",
      "type": "form"
      "method": "POST",
    },
    "items": [
      { "type": "text",
        "name": "login",
        "required": "true",
        "label": "Username or Email"
      },
      {  "type": "password",
         "name": "password",
         "required": "true",
         "label": "Password"
      }
    ]
  }
}
```

Forms: Search / Query

```
{
  ...

  "findAccounts": {
    "meta": { "href": "…/v1/users", "type": "form", "method": "GET" },
    "items": [
      { "type": "text",
        "name": "username",
        "label": "Username"
        "orderable": true
      },
      { "type": "email",
        "name": "email",
        "label": "Email"
        "orderable": true
      },
      { "type": "text",
        "name": "givenName",
        "label": "First Name"
        "orderable": true
      },
      { "type": "text",
        "name": "surname",
        "label": "Last Name"
        "orderable": true
      },
    ]
  }
}
```

Many To Many

Group to Account

- A group can have many accounts
- An account can be in many groups
- Each mapping is a resource:

`GroupMembership`

GET /groupMemberships/231k3j2j3

```
200 OK
{
  "meta":{"href": "…/groupMemberships/231k3j2j3"},
  "account": {
    "meta":{"href": "…"}
  },
  "group": {
    "meta"{"href": "…"}
  },
  …
}
```

GET /accounts/x7y8z9

```
200 OK
{
  “meta”:{“href”: “…/accounts/x7y8z9”},
  “givenName”: “Tony”,
  “surname”: “Stark”,
  …,
  “groups”: {
    “meta”:{“href”: “…/accounts/x7y8z9/groups”}
  },
  “groupMemberships”: {
    “meta”:{“href”: “…/groupMemberships?accountId=x7y8z9”}
  }
}
```

Async or Long-Lived Operations

```
POST /emails

{
   “from”: me@somewhere.com,
   “subject”: “Hi!”
   “body”: “...”
}
```

```
204 Accepted
Location: /emails/23Sd932sSl

{
   "status": "queued",
   ...
}
```

```
GET /emails/23Sd932sSl
Expires: 2014-09-29T18:00:00.000Z

{
   "status": "sent",
   ...
}
```

Batch Operations

- Each batch is represented as a resource
- Batches are likely to be a collection
- Batches are likely to have a status
- Batch deletes easier than create/update

Batch Delete

"Delete all company.com accounts"

```
DELETE /accounts?
        email=*@company.com
```

Batch Create / Update

Already have a Collection concept. Use it.

Batch Create or Update

POST /accounts

```
{
  "items": [
    { ... account 1 ... },
    { ... account 2 ... },
    ...
  ]
}
```

Batch Operations: The ‘Catch’

HTTP Caching is bypassed entirely ☹

204 Accepted
Location: /batches/a1b2c3

```
{
  "status": "processing", //overall status
  "size": "n",
  "limit": 25,
  ...,
  "items": {
    { response 1 (w/ individual status) ...},
    { response 2 (w/ individual status) ...},
    ...
  }
}
```

Errors

- As descriptive as possible
- As much information as possible
- Developers are your customers

POST /directories

409 Conflict

```
{
  “status”: 409,
  “code”: 40924,
  “property”: “name”,
  “message”: “A Directory named ‘Avengers’ already exists.”,
  “developerMessage”: “A directory named ‘Avengers’ already exists.  If you have a stale local cache, please expire it now.”,
  “moreInfo”: “https://www.stormpath.com/docs/api/errors/40924”
}
```

Security

Avoid sessions when possible

- Authenticate every request if necessary
- Stateless

Authorize based on resource content, NOT URL!

Use Existing Protocol:

- Oauth 1.0a, Oauth2, Basic over SSL only

Custom Authentication Scheme:

- Only if you provide client code / SDK
- Only if you really, *really* know what you're doing

Use API Keys and/or JWTs instead of Username/Passwords

401 vs 403

- 401 “Unauthorized” *really* means Unauthenticated

  “You need valid credentials for me to respond to this request”

- 403 “Forbidden” *really* means Unauthorized

  “Sorry, you’re not allowed!”

HTTP Authentication Schemes

- Server response to issue challenge:

`WWW-Authenticate:` <*scheme name*>
`realm=`"Application Name"

- Client request to submit credentials:

`Authorization:` <*scheme name*> <*data*>

API Keys

- Entropy
- Password Reset
- Independence
- Scope
- Speed
- Limited Exposure
- Traceability

IDs

- IDs should be opaque
- Should be globally unique
- Avoid sequential numbers (contention, fusking)
- Good candidates: UUIDs, ‘Url64’

HTTP Method Overrides

```
POST /accounts/x7y8z9?_method=DELETE
```

Caching & Concurrency Control

Server (initial response):

```
ETag: "686897696a7c876b7e”
```

Client (later request):

```
If-None-Match: "686897696a7c876b7e”
```

Server (later response):

```
304 Not Modified
```

Maintenance

Use HTTP Redirects

Create abstraction layer / endpoints when migrating

Use well defined custom Media Types

IETF RFC?

ionwg.github.io/draft-ion.html

- **Free** for developers
- Eliminate months of development
- Automatic security best practices
- Single Sign On for your apps
- API Authentication & Key Management
- Token Authentication for SPAs / Mobile
- Authorization

Libraries and integrations:
https://docs.stormpath.com