Ole Lensmar – CTO – SmartBear Software

PAST, PRESENT AND FUTURE OF APIS FOR MOBILE AND WEB APPS
Once upon a time…
We tried to connect (early 90:ies)

- Multiple protocols / initiatives
  - DCE/RPC (OSF)
  - CORBA (OMG)
  - COM / DCOM (Microsoft)
  - J2EE / RMI (Sun)

- They all had their challenges
  - Proprietary, Complex, Limited, etc.
Then - the Internet came along…

- HTTP
  - lightweight “universal” text-based protocol

- XML
  - “lightweight” text markup syntax

- “POX” – plain old XML

- HTTP+XML became XML-RPC

- SOAP (Microsoft)
  - “Simple Object Access Protocol”
  - XML-based messaging protocol – transport independent
REST arrives – and SOAP evolves

- REST was introduced by Roy T. Fielding

- SOAP 1.1
  - WSDL, XML-Schema
  - W3C recommendation in 2003

- WS-I Basic Profile (2004)
  - Guidelines on how to implement SOAP related-standards
  - Doc/Literal replaces RPC
  - Top-down vs Bottom up design
... and Web APIs emerge

- Salesforce launches XML API (2000)
  - “Salesforce Automation”

- eBay launches their API (2000)
  - Initially limited to select partners/developers

- Social
  - Del.icio.us (2003)
  - Flickr (2004)
  - Facebook, Twitter, Google Maps, etc (2006)

- All APIs were central to the reach and success of their providers
Web Applications evolve

- Web 2.0 Technology Stack
  - AJAX (Asynchronous JavaScript and XML)
  - HTML5 / CSS
  - JSON

- Web starts turning into a platform
  - ProgrammableWeb launches 2005
  - API Management (Mashery)
  - Still mainly XML
At the same time - SOAP gets “enterprisy”

- QoS specifications – WS-*
  - WS-Addressing
  - WS-Security
  - WS-Reliable Messaging
  - etc

- SOA Architectures become “mainstream”

- Limited use of SOAP for public APIs
  - Difficult to consume from Web 2.0 stack
The client landscape continued to change

- Mobile takes the lead with native/hybrid/web applications
  - Mostly API-driven

- Single Page Applications (SPA)
  - HTML5
  - Dynamic UI that pulls all data from backend via APIs

- Device proliferation
  - Android, iOS, Windows Phone, TVs, Consoles, etc..
  - APIs enable adoption to new devices
APIs fuel cloud and infrastructure

- Amazon
  - S3 – cloud-based storage (2006)
  - EC2 - re-sizable compute capacity (2006)
  - Both REST APIs that lacked web interfaces for several years!

- Twilio – voice and messaging (2007)

-> APIs enable a “building-block” approach to applications and architectures
APIs at the heart of applications

Mobile
Web / SPA
Devices
Integrations

Storage API
Compute API
Messaging API
eCommerce API
And APIs just continue to grow…

- SOA architectures are moving to REST from SOAP

- API Directories
  - programmableweb.com
  - apihub.com
  - publicapis.com

- apicommons.org
  - Collection of shared API definitions

- QoS
  - OAuth, OpenID Connect, Tokens, etc.
From an architectural point of view…
20 years ago

ACME Corp
(does monolithics)
10 years ago

ACME Corp (does SOA)

Web app

API

RMI

SOA P

MQ

Corp
And now…

ACME Corp (does APIs)

Web app

API

Corp

APP

API

App + API

IoT Device

Corporation

API

Device
API Oriented Architecture

- Key ingredients in a distributed application architecture can be consumed / provided via APIs
  - Storage
  - Messaging
  - eCommerce
  - Virtualization
  - Compute / Provisioning
  - Etc..

- Focus on core business
Let’s back up a little…
What is REST?

- REST is an architectural style – not a technology!
- Resources are identified with URIs
  - /users/12343/address
  - /cities/boston/hotels?area=downtown
- HTTP Verbs are used for actions
  - GET – retrieve resource representation(s)
  - POST – create resource(s) at URI (not idempotent)
  - PUT – replace resources identified by URI (idempotent)
  - DELETE – delete specified resource(s)
  - PATCH – update specified resource
- Representations and Content-types control semantics
Hypermedia APIs

- HATEOAS – Hypermedia As The Engine Of Application State
- Embed links to applicable actions in REST responses – clients shouldn’t need to know in advance what can be done next

- Pros
  - Designed for scale
  - Change and Context tolerant
  - Allows “discovery” of APIs

- Cons
  - Hypermedia is a “human” concept - client logic can get complex
  - Requires aggressive caching for performance
REST Maturity Model

- Level 0: The Swamp of POX
- Level 1: Resources
- Level 2: HTTP Verbs
- Level 3: Hypermedia Controls

Glory of REST
Hypermedia Example

POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
    <patient id = "jsmith"/>
</appointmentRequest>
Hypermedia Example

POST /slots/1234 HTTP/1.1
[various other headers]

HTTP/1.1 201 Created
Location: http://royalhope.nhs.uk/slots/1234/appointment
[various headers]

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
  <link rel = "/linkrels/appointment/cancel"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/addTest"
        uri = "/slots/1234/appointment/tests"/>
  <link rel = "self"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/changeTime"
        uri = "/doctors/mjones/slots?date=20100104@status=open"/>
  <link rel = "/linkrels/appointment/updateContactInfo"
        uri = "/patients/jsmith/contactInfo"/>
  <link rel = "/linkrels/help"
        uri = "/help/appointment"/>
</appointment>
POST /slots/1234 HTTP/1.1
[various other headers]

<!-- AppointmentRequest REST interface -->

HTTP/1.1 201 Created
Location: http://royalhope.nhs.uk/slots/1234/appointment
[various headers]

<!-- Appointment element -->

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
  <link rel = "/linkrels/appointment/cancel"
       uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/addTest"
       uri = "/slots/1234/appointment/tests"/>
  <link rel = "self"
       uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/changeTime"
       uri = "/doctors/mjones/slots?date=20100104@status=open"/>
  <link rel = "/linkrels/appointment/updateContactInfo"
       uri = "/patients/jsmith/contactInfo"/>
  <link rel = "/linkrels/help"
       uri = "/help/appointment"/>
</appointment>
REST API Descriptions

- API metadata can be used for
  - documentation,
  - validation + testing
  - code-generation

- Swagger - code oriented (bottom-up)
  - large community + great tools for code generation

- RAML, API-Blueprint - design-oriented (top-down)
  - great authoring tools

- WADL - inspired by WSDL – never caught on
Swagger Example

```json
{
    apiVersion: "1.0.0",
    swaggerVersion: "1.2",
    basePath: "http://petstore.swagger.wordnik.com/api",
    produces: [
        "application/json",
        "application/xml",
        "text/plain",
        "text/html"
    ],
    apis: [
        {
            path: "/pet/{petId}",
            operations: [
                {
                    method: "GET",
                    summary: "Find pet by ID",
                    notes: "Returns a pet based on ID",
                    type: "Pet",
                    nickname: "getPetById",
                    authorizations: { },
                    parameters: [
                        {
                            name: "petId",
                            description: "ID of pet that needs to be fetched",
                            required: true,
                            type: "integer",
                            format: "int64",
                            paramType: "path",
                            allowMultiple: false,
                            minimum: "1.0",
                            maximum: "100000.0"
                        }
                    ],
                    responseMessages: [
                        {
                            code: 400,
                            message: "Invalid ID supplied"
                        },
                        {
                            code: 404,
                            message: "Pet not found"
                        }
                    ]
                }
            ]
        }
    ]
}```
Swagger Sample App
This is a sample server Petstore server. You can find out more about Swagger at [http://swagger.wordnik.com](http://swagger.wordnik.com) or on irc.freenode.net, #swagger. For this sample, you can use the api key "special-key" to test the authorization filters

**Terms of service**
**Contact the developer**
**Apache 2.0**

### pet : Operations about pets

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/pet/{petId}</td>
<td>Find pet by ID</td>
</tr>
<tr>
<td>DELETE</td>
<td>/pet/{petId}</td>
<td>Deletes a pet</td>
</tr>
<tr>
<td>PATCH</td>
<td>/pet/{petId}</td>
<td>partial updates to a pet</td>
</tr>
<tr>
<td>POST</td>
<td>/pet/{petId}</td>
<td>Updates a pet in the store with form data</td>
</tr>
<tr>
<td>POST</td>
<td>/pet/uploadImage</td>
<td>uploads an image</td>
</tr>
<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>
</tbody>
</table>
Looking ahead – REST faces some challenges…
Experience APIs

- Model APIs after user experience – not resources

- NetFlix
  - 800+ devices / homescreens
  - Each homescreen made multiple REST calls – doesn’t scale
  - Solution – build one API call for each device;
    - /api/homescreen/ps4

- Orchestrate / Aggregate needed internal APIs on the server
Binary Protocols

- Several CORBA-like alternative
  - Thrift (Facebook)
  - Protobuf (Google)
  - Avro (Apache/Hadoop)

- All have an IDL with language bindings

- Problems solved:
  - Performance (processing and bandwidth)
  - Type-safety / interop
  - Improved QoS built in the protocol
Async / Real-Time APIs…

- **API-driven applications** often poll for data updates
  - Imposes bandwidth + performance overhead
  - Insufficient for “real” real-time needs

- **Real-Time APIs** push data to clients when needed

- **WebSockets (W3C)**
  - Supported in all major browsers
  - Full-duplex communication over a single TCP connection

- **Webhooks**
  - User-defined HTTP callbacks
  - Supports REST concepts
SDKs vs APIs...

- **SDKs greatly ease adoption**
  - No need to learn / implement underlying protocol
  - Native language bindings natural for developers
  - API provider has flexibility to change

- **SDKs pose great challenges too**
  - Dependencies
  - Versioning
  - Support

SMARTBEAR
And of course - the Internet of Things

- IoT devices have limited power and bandwidth
  - low complexity and footprint for APIs
  - publish/subscribe instead of request/response
  - minimized on-the-wire formats
  - automatic (re)connection management

- A number of real-time protocols in use
  - MQTT, CoAP, AMQP, STOMP, etc

- IoT Brokers connect and integrate multiple devices
So if you’re building APIs, what should be keeping you up at night?
API Hierarchy of Needs

- Usability
  - is it easy to setup and use?
- Functionality
  - can it be used in unexpected ways?
- Reliability
  - does it work repeatedly?
- Proficiency
  - does it increase developers' skills?
- Creativity
Who is going to use your API?
User Experience

=

Developer Experience
Align with your users technology

- SOAP / REST / Corba / etc…
- XML / JSON / YAML / etc…
- Honor QoS and Security
Help users understand your API
Help users consume your API

- Code samples in common languages
- Native SDKs
- Provide sandbox environments
Provide API Metadata

- Validation
- Code Generation
- Coverage
- Understanding
- Simulation

wsdl, swagger, wadl, hal, json schema, apiary.io, xml schema, ws-*, apiary, api-docs, raml, iodocs, etc
Align with your users domain

- Process / Workflow
- Nomenclature
- Related APIs
A 3:30:3 Litmus test for APIs

- 3 Minutes to understand what an API does
- 30 seconds to sign up
- 3 minutes to the first request

(Ori Pekelman)
Recommendations…
API First

- APIs are at the heart of
  - Mobile Strategies
  - Web Strategies
  - Partner / Integration Strategies
  - Developer / Community Strategies
  - Cloud / Infrastructure Strategies

- APIs should be treated as a first-level citizen - not as an after sight
Technology & Implementation

- Avoid (REST) religion
- Choose what’s best for you and your users
- Understand the importance of DX – both internally and externally
- Use your public APIs internally!
And please…

Love your APIs – and they’ll love you back!
Thanks!

- @olensmar
- ole.lensmar@smartbear.com