ca technologies

From APIs to Microservices: Design and Build

Mike Amundsen, API Academy, CA Technologies @mamund

July 2017


Ctrl
PgUp
Mike Amundsen
@mamund

MENU

API ACADEMY

SERVICES

EVENTS

EBOOK

MICROSERVICE ARCHITECTURE: ALIGNING PRINCIPLES, PRACTICES & CULTURE

DESIGN AND APPLY MICROSERVICES TO EMBRACE CONTINUAL CHANGE IN THE DIGITAL ECONOMY

READ MORE

http://g.mamund.com/msabook

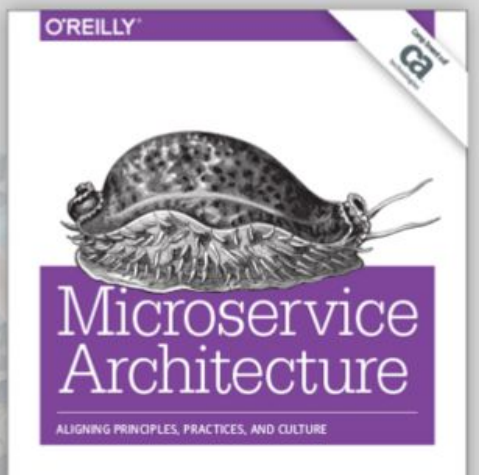
ca technologies

Microservice Architecture: Aligning Principles, Practices, and Culture

Microservices is the next evolution in software architecture designed to help organizations embrace continual change in the digital economy. But how do you design and apply an effective microservice architecture?

This new book from O'Reilly provides comprehensive guidance through seven valuable chapters that give you a deep-dive into:

- The benefits and principles of microservices
- A design-based approach to microservice architecture
- Lessons for applying microservices in practice



Overview

- The Business of APIs
- Microservices
- The Value of Design
- An API Design Methodology
- Three-Steps to API Implementation

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

The Business of APIs

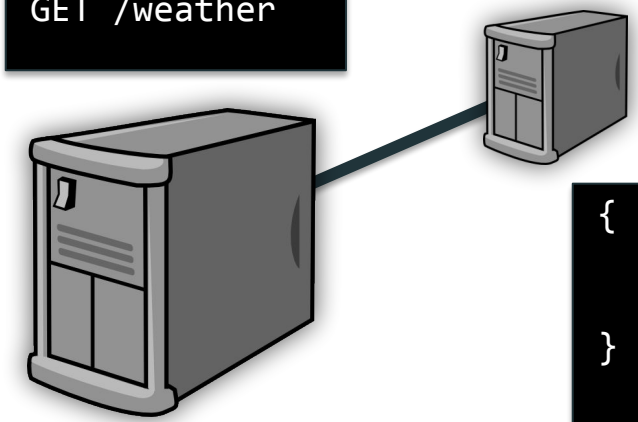
API

abbreviation

1. application programming interface


Allie
Should I get the new Blackberry Torch?
Chats (2)
Rick
Erik M. New pin-
I'm bbming u from my new playbook
Contacts (26)
Rick
BC Ferries lol.
Seriously?
Doing what?
Hey what are
q
w
e
t
i
o
p
a
s
g
l
aA
z
v
b
Space
123
BlackBerry

```
GET /weather
```



```
{
  “city”: “London”,
  “temp”: 22
}
```

APIs allow us to unlock hidden business value

- Create New Applications
- Identify New Revenue Streams
- Initiate New Businesses


good APIs make interaction easy

Typical Mobile App

Frontend
(client)

Backend
(server)

Build page and send

user input

temperature data (by location)

layout, graphics and text

Powered by APIs

Frontend
(client)

Backend
(server)

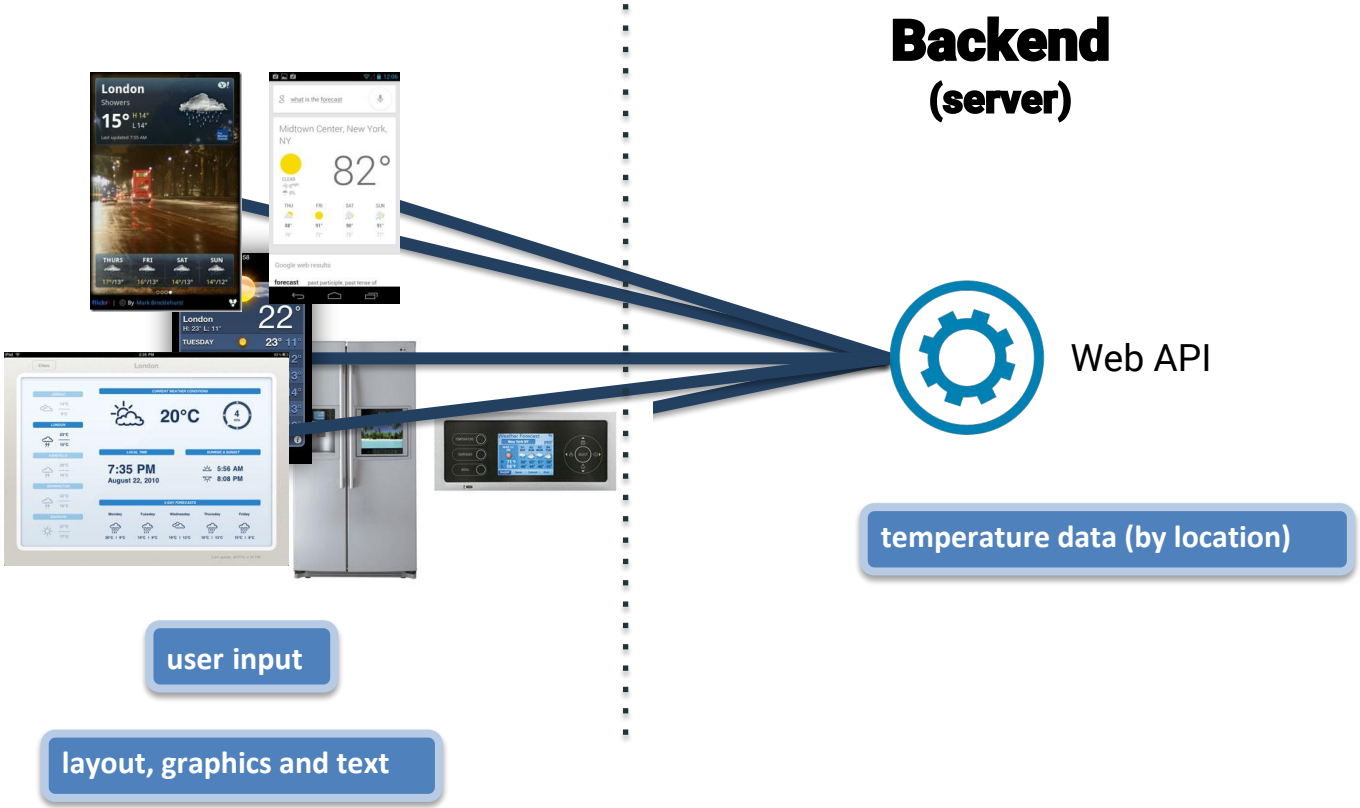
Web API

user input

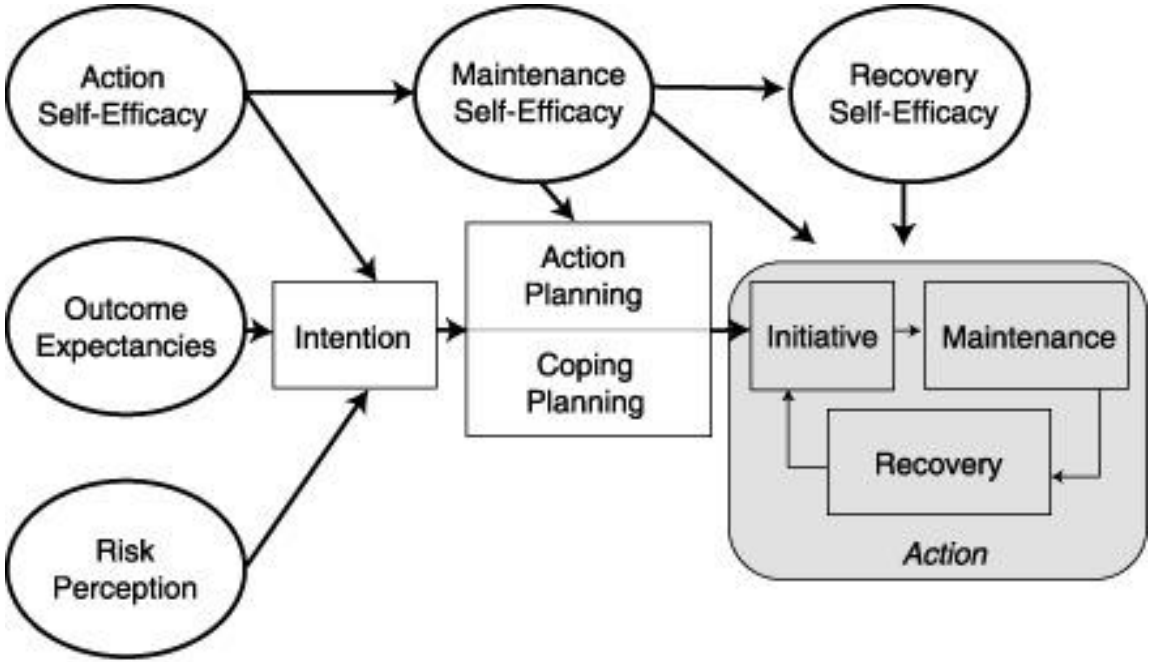
temperature data (by location)

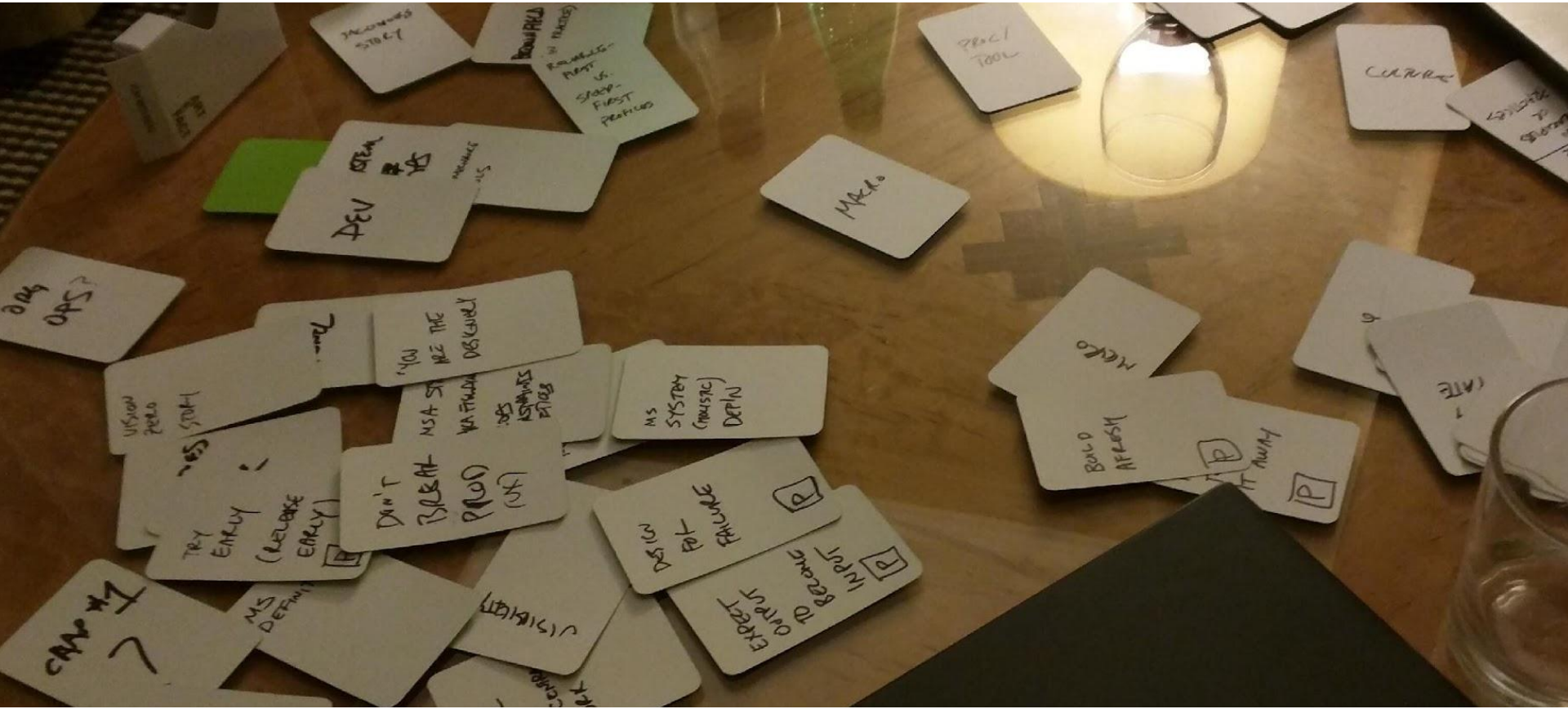
layout, graphics and text

APIs enable Multi-channel Delivery



Microservices





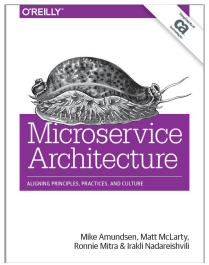

FLY CALIFORNIA


VAULT DOOR
WEIGHT: 22 1/2 Tons
THICKNESS: 22 Inches
STEEL: 11 Layers of Special Cutting and Drill Resistant
LOCKS: 4 Hamilton Watch Movements for Time Locks

Balancing Speed and Safety at Scale

The Microservice Way

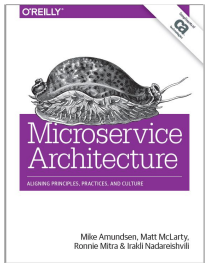
A ***microservice*** is an independently deployable component of bounded scope that supports interoperability through message-based communication.



The Microservice Way

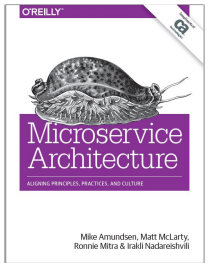
A ***microservice*** is an independently deployable component of bounded scope that supports interoperability through message-based communication.

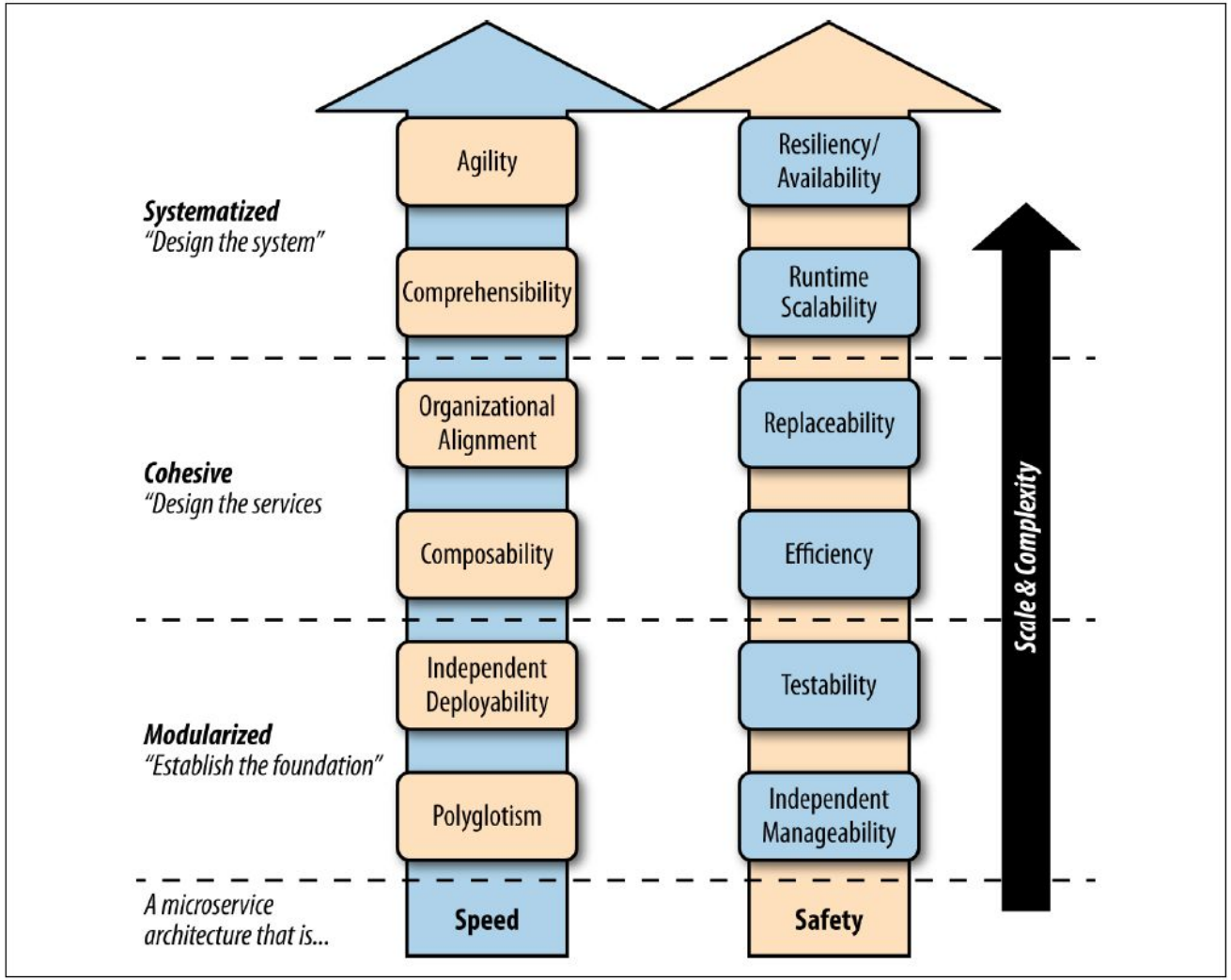
Microservice architecture is a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices.



The Microservice Way

- Microservices are ideal for **big systems**
- Microservice architecture is **goal-oriented**
- Microservices focus on **replaceability**

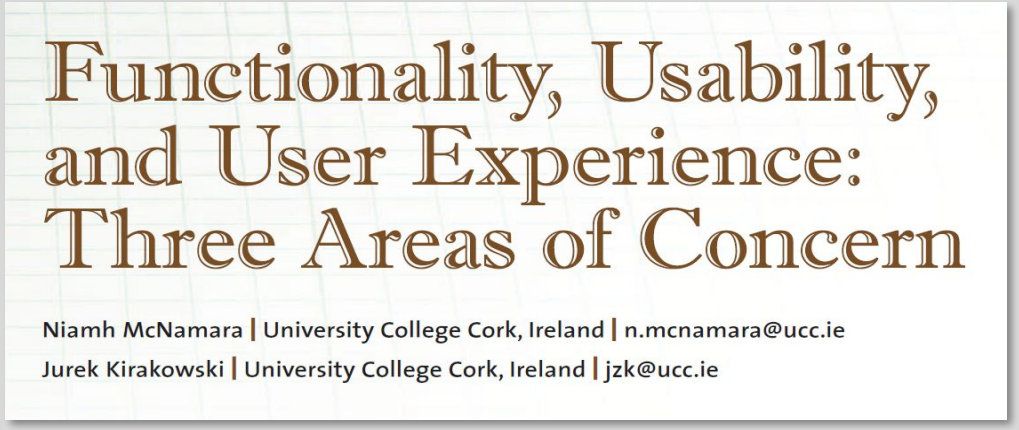



Agility
Resiliency/ Availability
Systematized
"Design the system"
Comprehensibility
Runtime Scalability
Organizational Alignment
Replaceability
Cohesive
"Design the services
Composability
Efficiency
Scale & Complexity
Independent Deployability
Testability
Modularized
"Establish the foundation"
Polyglotism
Independent Manageability
A microservice architecture that is...
Speed
Safety

The Value of Design

“Few people think about it or are aware of it. But there is nothing made by human beings that does not involve a design decision somewhere.”

Bill Moggridge
Interaction Design Pioneer



design

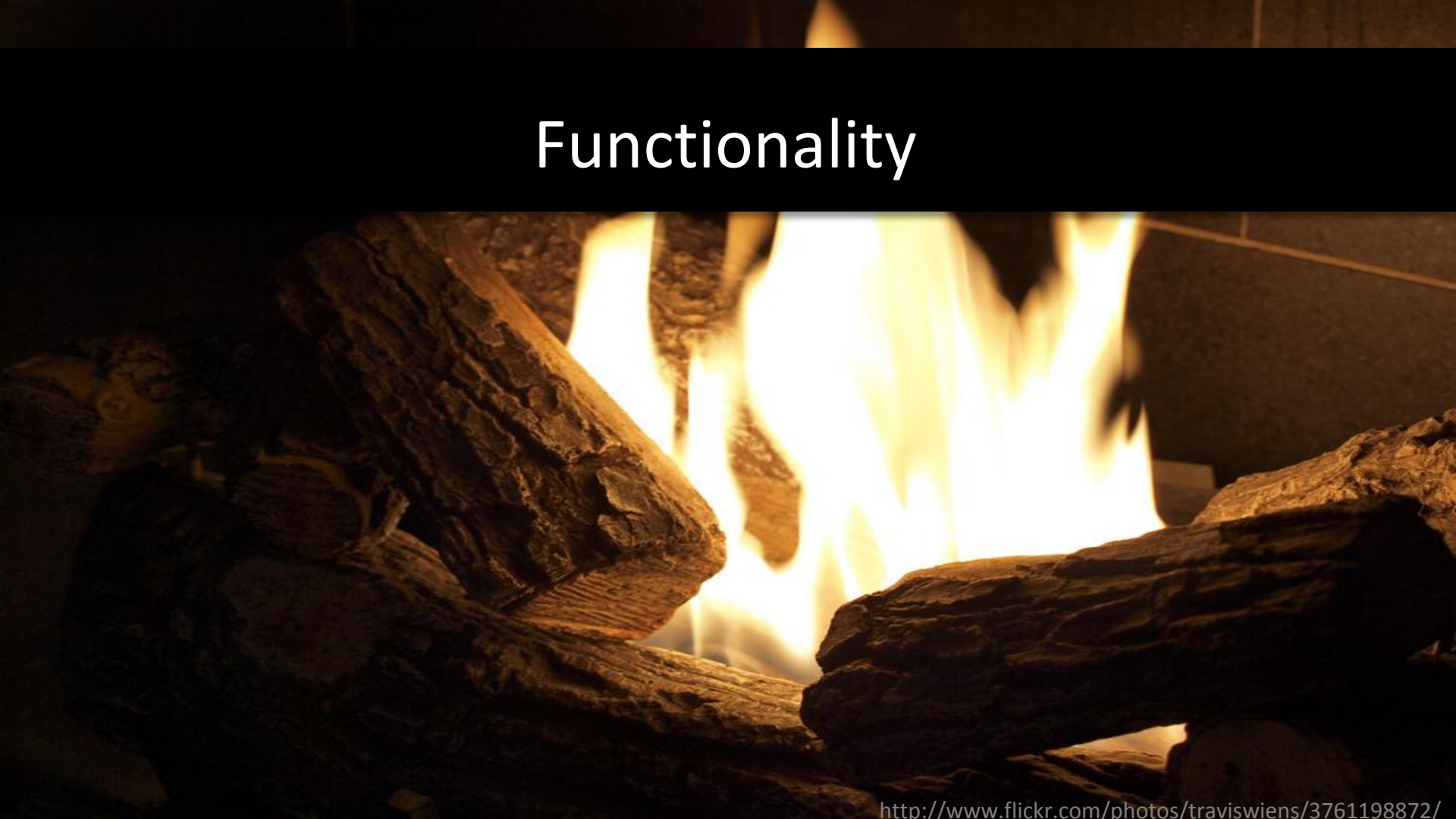
Functionality

Usability

Experience

http://www.flickr.com/photos/monicamuller/3171329060

Functionality



http://www.flickr.com/photos/traviswiens/3761198872/

Usability

Experience

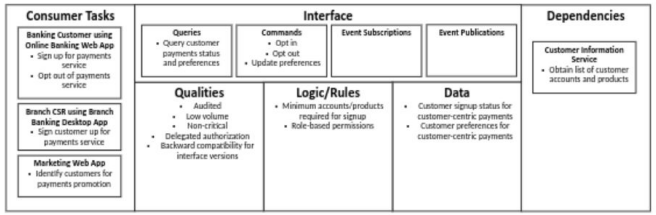
An API Design Methodology

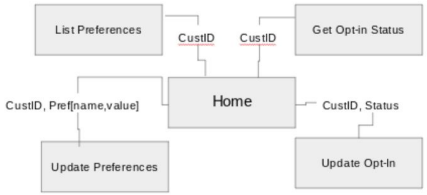
```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

An API Design Methodology

A repeatable process to govern the creation of interfaces

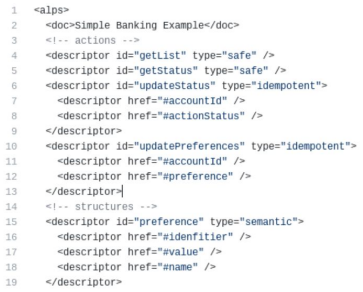
- Produce a Service Canvas
- Draw a Diagram
- Apply Vocabularies
- Create Description Document






```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

Produce a Service Canvas



```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

Design Canvas: Customer-centric Payments Management Service

Consumer Tasks

Banking Customer using Online Banking Web App
- Sign up for payments service
- Opt out of payments service

Branch CSR using Branch Banking Desktop App
- Sign customer up for payments service

Marketing Web App
- Identify customers for payments promotion

Interface

Queries
- Query customer payments status and preferences

Commands
- Opt in
- Opt out
- Update preferences

Event Subscriptions

Event Publications

Qualities
- Audited
- Low volume
- Non-critical
- Delegated authorization
- Backward compatibility for interface versions

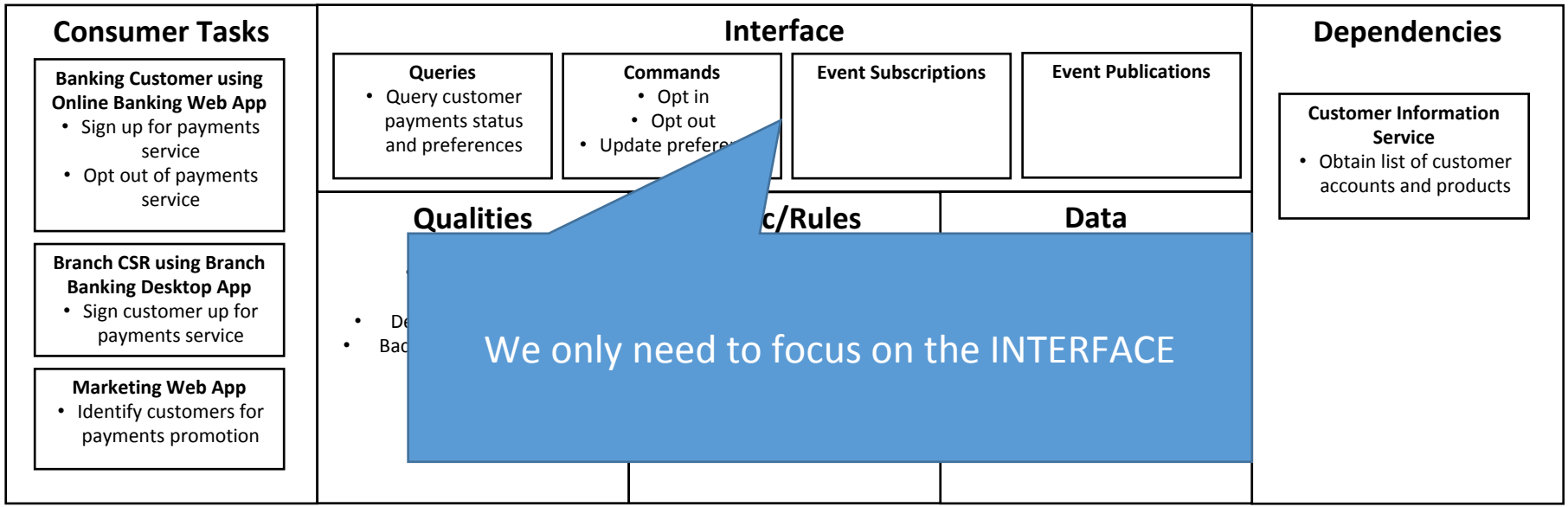
Logic/Rules
- Minimum accounts/products required for signup
- Role-based permissions

Data
- Customer signup status for customer-centric payments
- Customer preferences for customer-centric payments

Dependencies

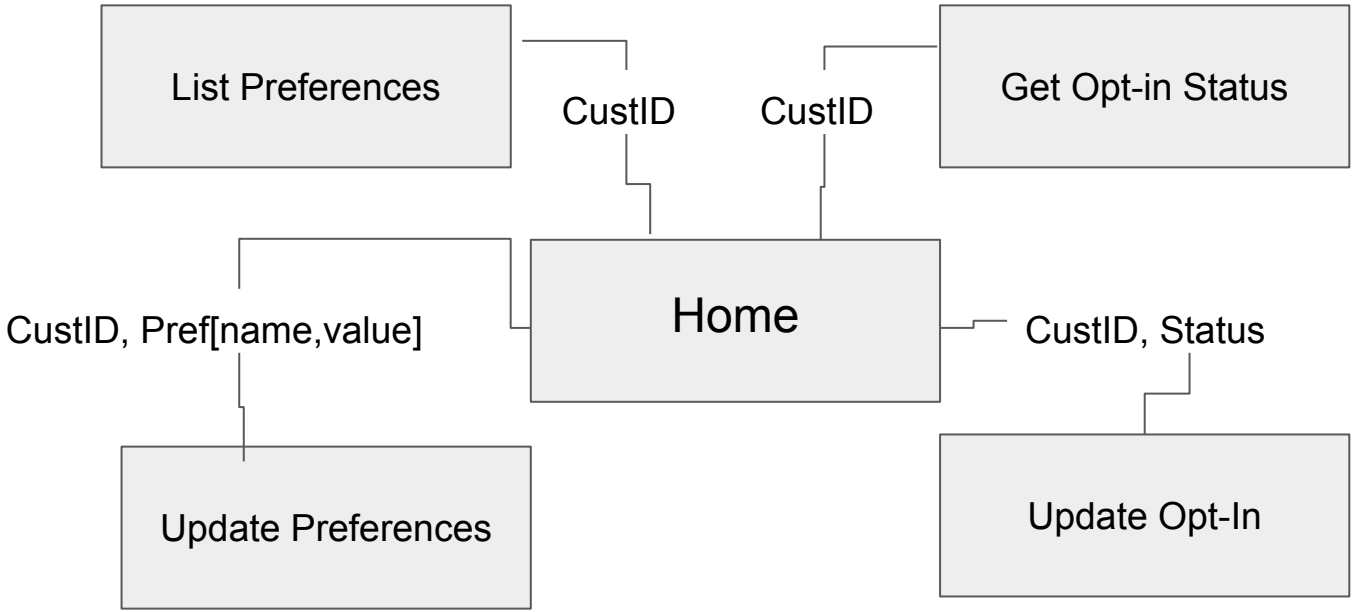
Customer Information Service
- Obtain list of customer accounts and products

List all the Actions

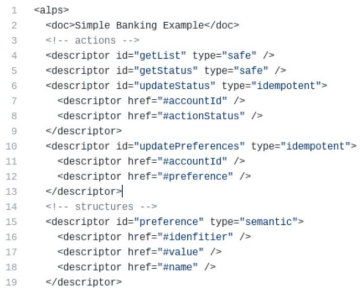


Draw a Diagram

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```


List Preferences
CustID
CustID
Get Opt-in Status
CustID, Pref[name,value]
Home
CustID, Status
Update Preferences
Update Opt-In

Apply Vocabularies



```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

Sources for Vocabularies

- IANA Link Relation Values
- schema.org
- microformats
- Dublin Core
- Activity Streams
- Industry Vocabularies (BIAN, etc.)
- Your Enterprise Vocabularies

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

Before Applying Vocabularies

- CustID,
- CustomerName,
- AccountName,
- AccountType
- Optin-Status(in, out)
- Preference(Name, Value, Prompt)
- GetStatus
- GetPreferences
- UpdateStatus
- UpdatePreferences

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

After Applying Vocabularies

- BankAccount.identifier,
- Customer.familyName,
- Customer.givenName,
- BankAccount.name,
- BankAccount.category
- ActionStatus("in", "out")
- ItemList(identifier, value, name)
- **GetStatus**
- **GetPreferences**
- **UpdateStatus**
- **UpdatePreferences**

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

Create a Description Document

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

Description vs. Definitions

- Describing the interface doesn't define it.
- Description languages
  - ALPS
  - DCAP
  - JSON Home

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

Description vs. Definitions

- Describing the interface doesn't define it.
- Description languages
  - ALPS
  - DCAP
  - JSON Home
- Definition languages
  - WSDL
  - Swagger
  - RAML
  - Blueprint

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

Description vs. Definitions

- Describing the interface doesn't define it.
- Description languages
  - **ALPS**
  - DCAP
  - JSON Home
- Definition languages
  - WSDL
  - Swagger
  - RAML
  - Blueprint

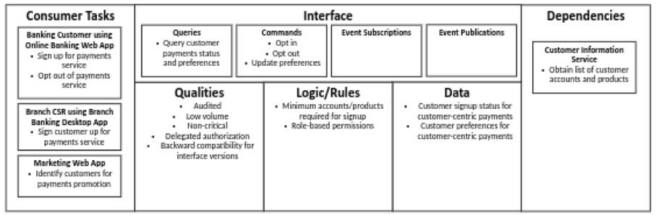
```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

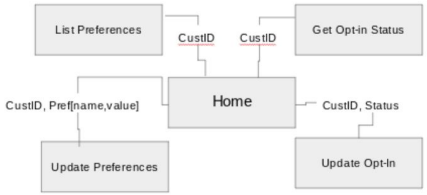
```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

Design Artifacts

- Service Canvas
- Diagram
- Description Document

Check these into source control






```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

Three Steps to API Implementation

Implementing APIs

- APIs are Interfaces, Not Functionality
- Implementing APIs means translating the Design
- Three-phase API Implementation
  - Sketching
  - Prototyping
  - Building

APIs are Interfaces

- You're not designing the functionality of a service
- You MAY already have that functionality somewhere
- You MAY need to create the functionality
- Focus on the "API-First"

You MAY already have the functionality

- Your job is to act as a "proxy" between the interface design and the existing functionality

- Identify the existing functionality (e.g. the MSC or portion of a monolith

You MAY need to create the functionality

- Your job is to act as a "guide" for the new functionality
- Offer a "shell" for future functionality
- Be prepared to do conversions

API First, API Forever

- Assume the API will not change, but the implementation details will

- Once released to production, it is easier to modify functionality than interfaces

trans·late

/trans'lāt,tranz'lāt/

verb

verb: **translate**; 3rd person present: **translates**; past tense: **translated**; past participle: **translated**; gerund or present participle: **translating**

1. express the sense of (words or text) in another language.
   "the German original has been **translated into** English"
   synonyms: render, put, express, convert, change; More

Translating the Design

- Translations are always approximate
- In many cases we "lose something in the translation…"
- API styles affect translation
  - SOAP
  - CRUD
  - Hypermedia
  - Reactive
- Don't worry if the translation is not exact.

Three-Phase API Implementation

Three-Phase API Implementation

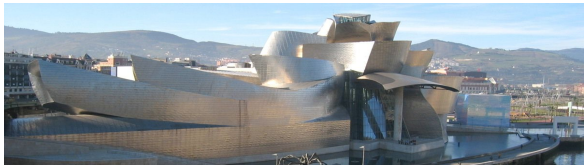
- To reduce cost and risk, take a three-phase approach
- **Sketching** - disposable experiments
- **Prototyping** - testable examples
- **Building** - production implementation

Reduce Cost and Risk in API Implementation

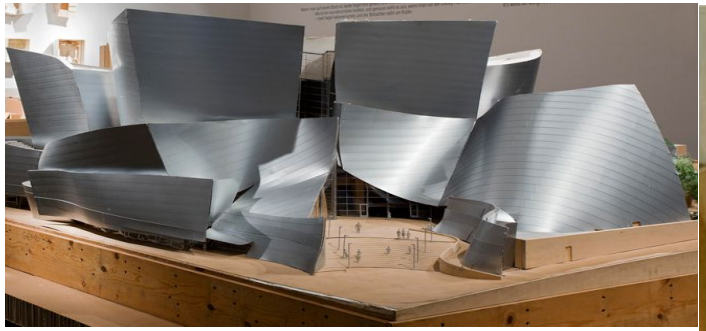
- Implementation can be costly
- Mistakes may be uncovered along the way
- Uncover mistakes early when they are inexpensive to fix
- Put off writing code for as long as possible.

Frank Gehry

An architect is given a program, budget, place, and schedule. Sometimes the end product rises to art



Frank Gehry

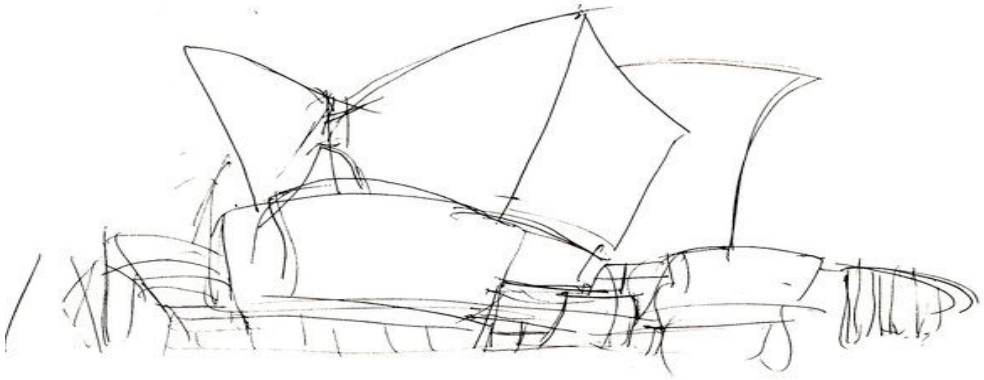


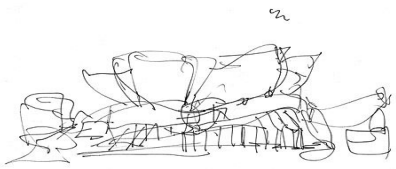
sketch

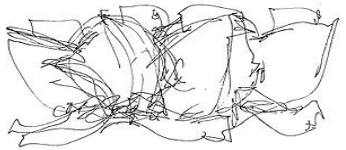
/skeCH/

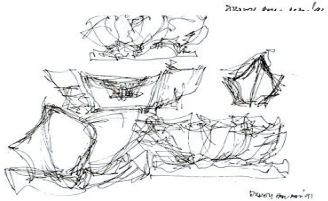
noun

1. a rough or unfinished drawing or painting, often made to assist in making a more finished picture.
   "a charcoal sketch"
   synonyms: (preliminary) drawing, outline; More





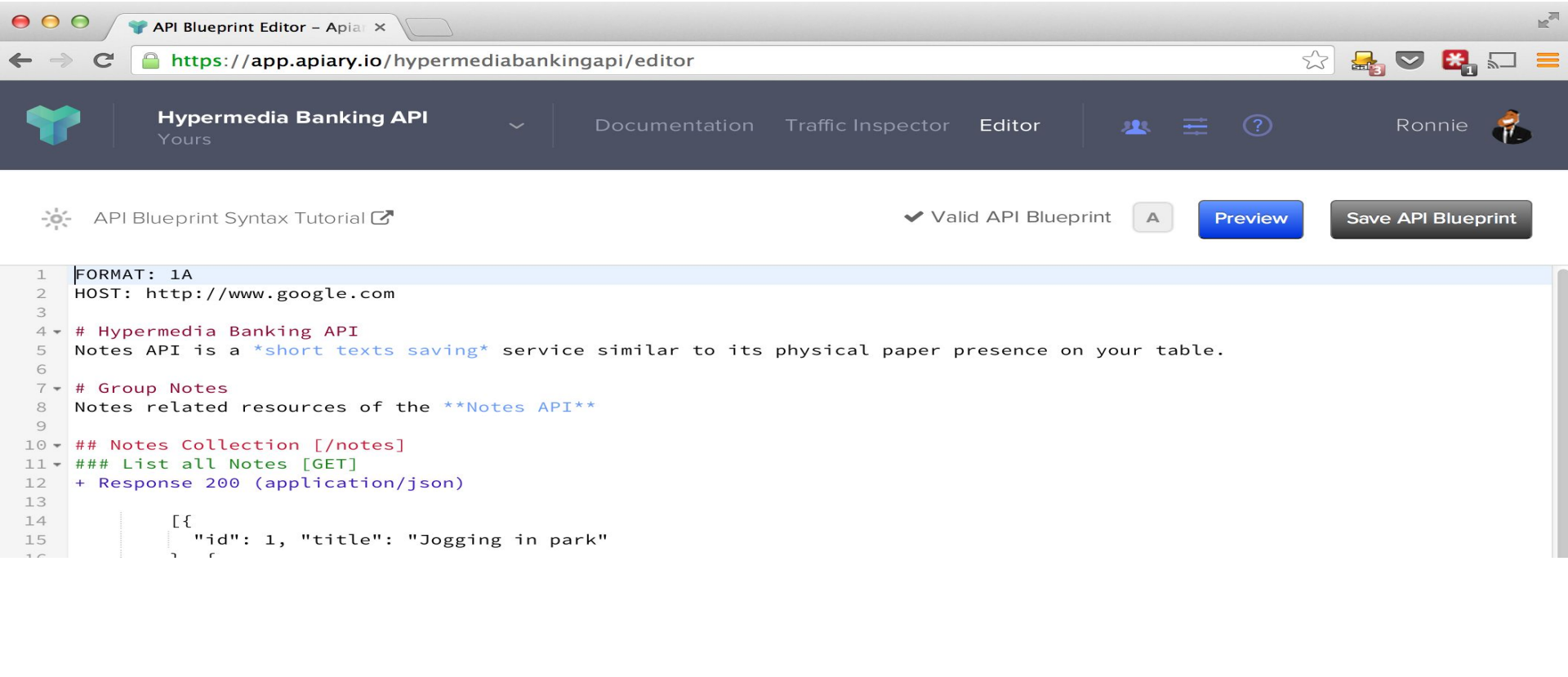




Sketching APIs

- Sketches are terse, rough drawings
- They give the general idea of a thing
  but lack important details.
- Usually, one can glean the basics from a sketch but
- Sketches usually are just explorations of ideas,
  not fully-formed items.

Apiary Blueprint



Sketching APIs

- Use tools like Apiary Editor to create a sketch.
- Show it to others (devs, stakeholders) and get their feedback.
- If possible use simple API consumer tools (curl, NodeJS, etc.) to test.
- Continue to modify the simple sketches as needed

Sketches are made to be thrown away.

pro·to·type

/ˈprōdəˌtīp/

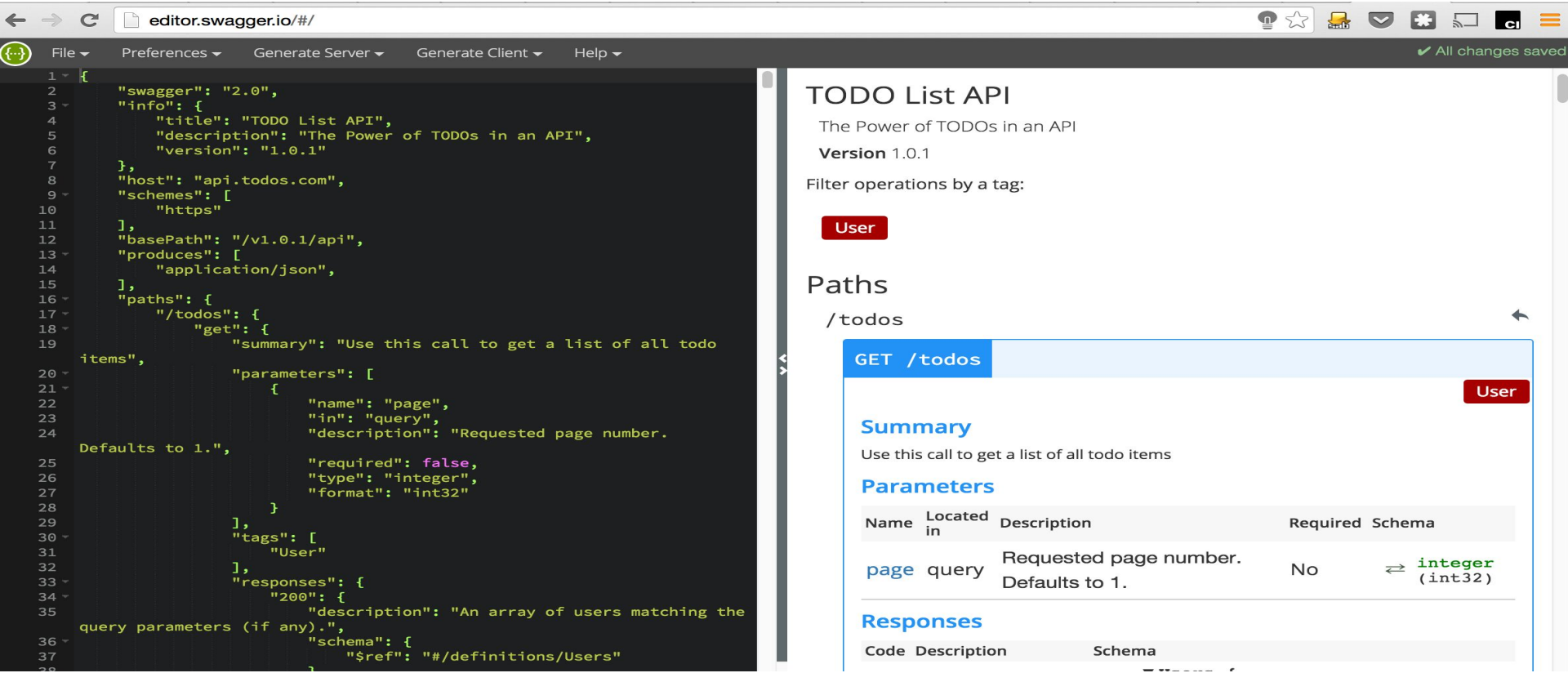
noun

1. a first, typical or preliminary model of something, especially a machine, from which other forms are developed or copied.
   "the firm is testing a prototype of the weapon"

Prototyping APIs

- Prototypes look like the real thing, but are not. They’re "fakes."
- They let you work up something with all the details of a real API, but without the actual functionality behind it.
- They’re an inexpensive way to work out the details
- Use them to discover challenges before you go into production.

Swagger Editor



Prototyping APIs

- Use tools like Live API Creator, NodeJS express, or other code-generating platforms.
- It's also a good idea to use service-virtualization frameworks to mock up the response data.
- If possible, include access-control checking when running tests against the prototype.
- If possible use existing production-level API consumers to test out the prototype.

Prototypes are made to be tested.

build

/bild/

verb

1. construct (something, typically something large) by putting parts or material together over a period of time.
   "the factory was built in 1936"
   synonyms: construct, erect, put up, assemble; More

Building APIs

- API builds are the real thing
- Production-ready, access-controlled, resilient, scalable.
- Building the production implementation means
  - Working out all the kinks
  - Supporting all the use-cases identified during the sketch and prototype phases.

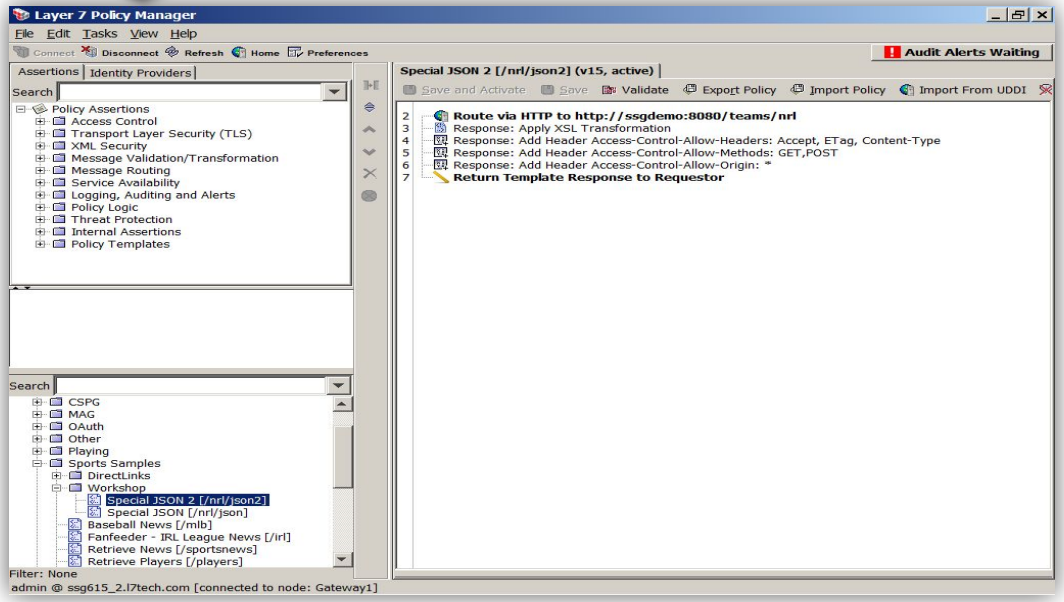
Building APIs

```
var express = require('express'),
app = express();
var port = 8080;
app.listen(port);

app.get("/tasks", function(req, res) {
    res.status(200).send(‘<response>
        <tasks>
            <task>
            <name>Pick up Kai</name>
            <priority>1</priority>
        </tasks>
    </response>’);
}
```

Building APIs

```
GET
/tasks
```



Building APIs

- Each implementation has their own challenges to overcome.
- Each deserves their own guidance and style-guides.
  - Gateway Policies
  - ESB Rules
  - Scripting (NodeJS)
  - Code (Java/C#)
- All require exhaustive testing at the unit, acceptance, and integration levels.
- All require detailed access control.

Production APIs are made last.

So...

Summary

- API enable Multi-channel Delivery
- Functionality, Usability, Experience
- Canvas, Diagram, Description
- Sketch, Prototype, Build

```
<alps>
  <doc>Simple Banking Example</doc>
  <!-- actions -->
  <descriptor id="getList" type="safe" />
  <descriptor id="getStatus" type="safe" />
  <descriptor id="updateStatus" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#actionStatus" />
  </descriptor>
  <descriptor id="updatePreferences" type="idempotent">
    <descriptor href="#accountId" />
    <descriptor href="#preference" />
  </descriptor>
  <!-- structures -->
  <descriptor id="preference" type="semantic">
    <descriptor href="#idenfitier" />
    <descriptor href="#value" />
    <descriptor href="#name" />
  </descriptor>
```

ca technologies

From APIs to Microservices: Design and Build

Mike Amundsen, API Academy, CA Technologies @mamund

July 2017