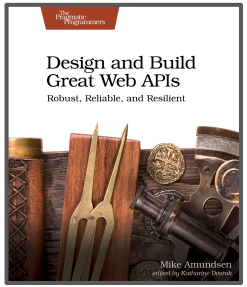
Building Great APIs
from the Ground Up

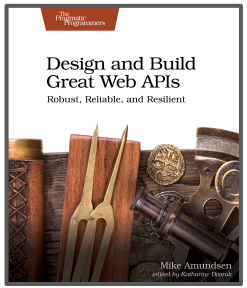
GOTO Amsterdam

Mike Amundsen
@mamund



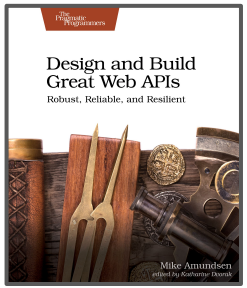
Morning

Design and Define



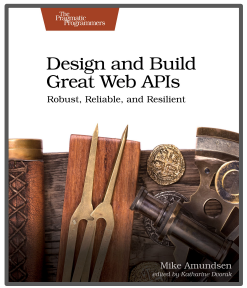
Design APIs

- Exploring APIs
- Why Web APIs?
- API Design Method



Exploring APIs -- BigCo, Inc

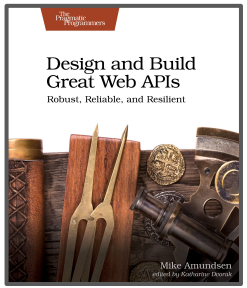
The Bayesian International Group of Companies (BigCo) was founded in 1827 in Scotland. Originally focused on demography and finance based on the work of Richard Price, by the 1900s BigCo was a manufacturer of important turn-of-the-century technologies including mobile x-ray machines and hydrophones by working with inventors Frederick Jones and Reginald Fessenden, respectively. BigCo was also one of the few non-US contractors attached to the Manhattan project, providing parts for the first nuclear reactors designed by Dr. Leona Wood.



Exploring APIs -- BigCo, Inc

We'll be helping BigCo, Inc. design, build, and deploy some new APIs. True to the API First design ethos, we'll only focus building the API and not spending time writing the services behind the API.

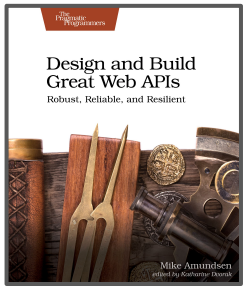
- Stories : https://github.com/mamund/2019-06-goto-amsterdam/tree/master/stories
- Company
- Accounts
- Activity



Exploring APIs -- Using **curl**

curl://

command line tool and library
for transferring data with URLs



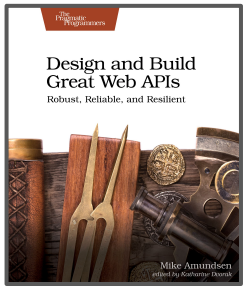
Exploring APIs -- Using **curl**

- Download & Install: https://curl.haxx.se
- Documentation: https://curl.haxx.se/docs/
- Starting URLs: https://github.com/mamund/api-starter/blob/master/requests/README.md

Requests

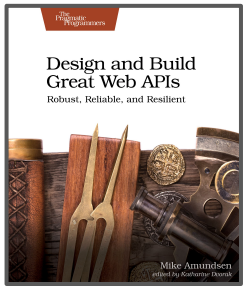
This folder holds results of various requests to running services.

- **Company** Service
  - https://evening-headland-10829.herokuapp.com
- **Account** Service (incomplete)
  - https://quiet-reaches-21904.herokuapp.com
- **Activity** Service
  - https://vast-mesa-64543.herokuapp.com



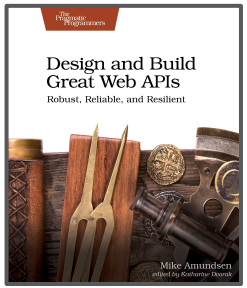
Web APIs

- HTTP
- The Web
- REST



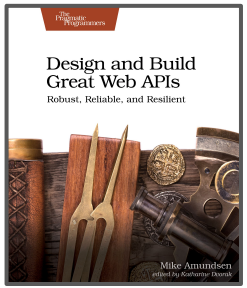
Web APIs -- HTTP (protocol)

- HTTP is an open standard protocol (IETF)
  - That means you can be WRONG!!!!<g>
- The big three:
  - Messages
  - Methods
  - Other Stuff



Web APIs -- HTTP (protocol)

- HTTP is all about messages
  - Not objects
  - Not functions
  - Just messages
- Every message has (up to) three parts
  - Start line
  - Header collection
  - Message body



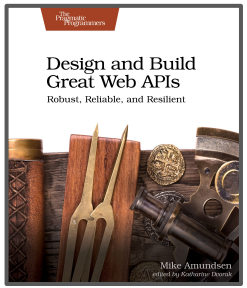
Web APIs -- HTTP (messages)

REQUEST

```
GET onboarding HTTP/1.1
User-Agent:
Host: apis.example.org
Accept: application/json
Accept-Language: en-us
Accept-Encoding: gzip, deflate
```

RESPONSE

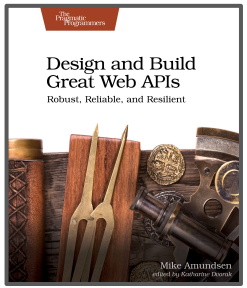
```
HTTP/2.0 200 OK
Date: Mon, 27 Jul 2019 12:28:53 GMT
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: application/json
Connection: Closed
```



Web APIs -- HTTP (methods)

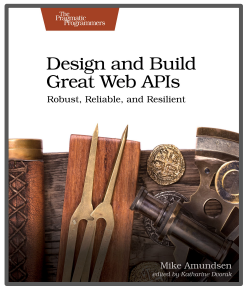
- HTTP clients use **Methods** to communicate intention
- GET, PUT, POST, DELETE ….
- Methods are NOT functions
- There are lots of registered methods

https://www.iana.org/assignments/http-methods/http-methods.xhtml



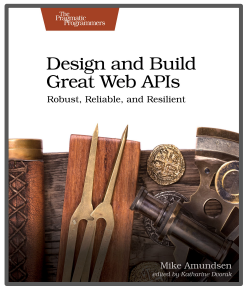
Web APIs -- HTTP (methods)

Method Name	Safe	Idempotent	Reference
ACL	no	yes	[RFC3744, Section 8.1]
BASELINE-CONTROL	no	yes	[RFC3253, Section 12.6]
BIND	no	yes	[RFC5842, Section 4]
CHECKIN	no	yes	[RFC3253, Section 4.4, Section 9.4]
CHECKOUT	no	yes	[RFC3253, Section 4.3, Section 8.8]
CONNECT	no	no	[RFC7231, Section 4.3.6]
COPY	no	yes	[RFC4918, Section 9.8]
DELETE	no	yes	[RFC7231, Section 4.3.5]
GET	yes	yes	[RFC7231, Section 4.3.1]
HEAD	yes	yes	[RFC7231, Section 4.3.2]
LABEL	no	yes	[RFC3253, Section 8.2]
LINK	no	yes	[RFC2068, Section 19.6.1.2]
LOCK	no	no	[RFC4918, Section 9.10]
MERGE	no	yes	[RFC3253, Section 11.2]
MKACTIVITY	no	yes	[RFC3253, Section 13.5]
MKCALENDAR	no	yes	[RFC4791, Section 5.3.1][RFC8144, Section 2.3]
MKCOL	no	yes	[RFC4918, Section 9.3][RFC5689, Section 3][RFC8144, Section 2.3]
MKREDIRECTREF	no	yes	[RFC4437, Section 6]
MKWORKSPACE	no	yes	[RFC3253, Section 6.3]
MOVE	no	yes	[RFC4918, Section 9.9]
OPTIONS	yes	yes	[RFC7231, Section 4.3.7]
ORDERPATCH	no	yes	[RFC3648, Section 7]
PATCH	no	no	[RFC5789, Section 2]
POST	no	no	[RFC7231, Section 4.3.3]
PRI	yes	yes	[RFC7540, Section 3.5]
PROPFIND	yes	yes	[RFC4918, Section 9.1][RFC8144, Section 2.1]
PROPPATCH	no	yes	[RFC4918, Section 9.2][RFC8144, Section 2.2]
PUT	no	yes	[RFC7231, Section 4.3.4]
REBIND	no	yes	[RFC5842, Section 6]
REPORT	yes	yes	[RFC3253, Section 3.6][RFC8144, Section 2.1]
SEARCH	yes	yes	[RFC5323, Section 2]
TRACE	yes	yes	[RFC7231, Section 4.3.8]
UNBIND	no	yes	[RFC5842, Section 5]
UNCHECKOUT	no	yes	[RFC3253, Section 4.5]
UNLINK	no	yes	[RFC2068, Section 19.6.1.3]
UNLOCK	no	yes	[RFC4918, Section 9.11]
UPDATE	no	yes	[RFC3253, Section 7.1]
UPDATEREDIRECTREF	no	yes	[RFC4437, Section 7]
VERSION-CONTROL	no	yes	[RFC3253, Section 3.5]



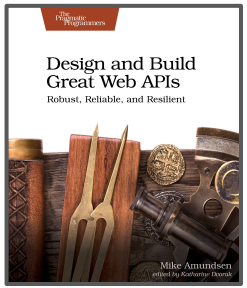
Web APIs -- HTTP (other stuff)

- HTTP offers two powerful assurances:
- **Safety** (GET vs. DELETE)
- **Idempotency** (PUT vs. POST)



Web APIs -- HTTP (other stuff)

- HTTP offers two powerful assurances:
- **Safety** (GET vs. DELETE)
- **Idempotency** (PUT vs. POST)

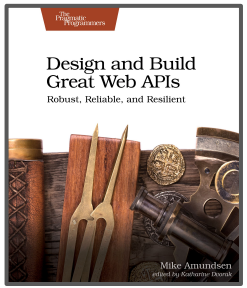


Web APIs -- HTTP (other stuff)

- **Safety** (GET vs. DELETE)

```
GET /company/delete?id=21
```

What's wrong with this HTTP request?



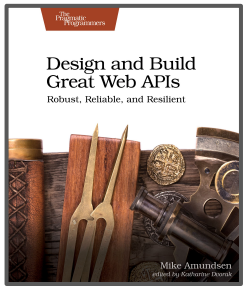
Web APIs -- HTTP (other stuff)

- **Idempotence** (PUT vs. POST)

The bank-transfer dilemma

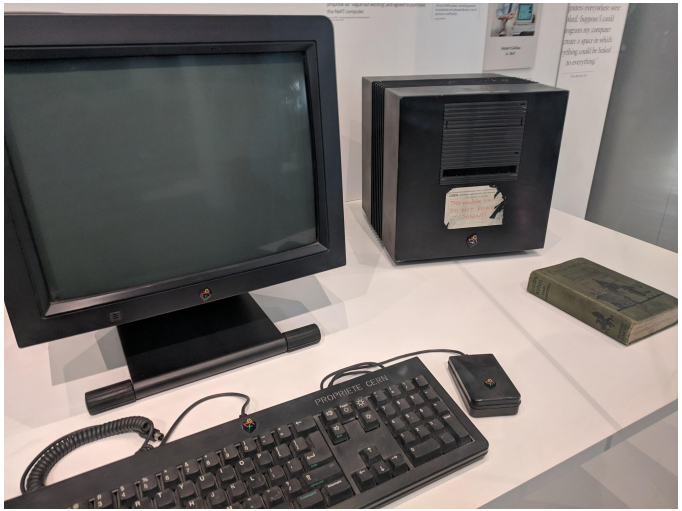
```
POST /bank-transfer HTTP/2.0
Host: apis.example.org
Accept: application/json
Content-Type: application/x-www-form-urnencoded

amount=1000&from-account=q1w2e3&to-account=zaxscd
```

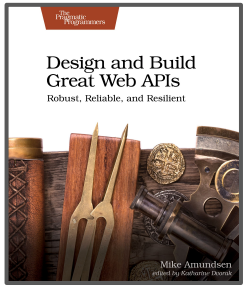
What if I never get a response?



Web APIs -- The Web

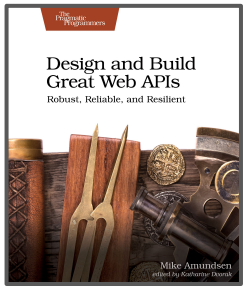
- Tim Berners-Lee built the first Web server and client in the 1980s
- HTTP & HTML
- Later CSS and Javascript was added by others





Web APIs -- The Web

- The WWW is not a standard or specification
- It is a set of common practices
- "A linked information system"
- Basic principles:
  - Pass messages
  - Include links to follow (GET)
  - Include forms to write data (POST)



Web APIs -- REST

- Roy Fielding created REST in 2000
- A list of properties and constraints
- REST not a standard or a common practice, it is a style

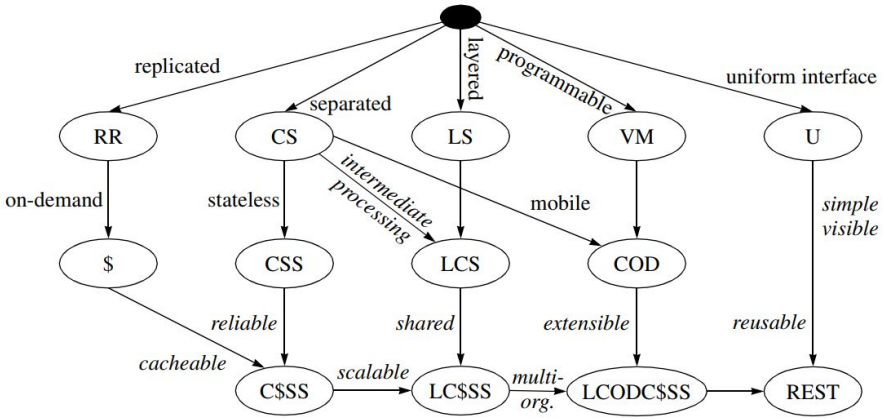
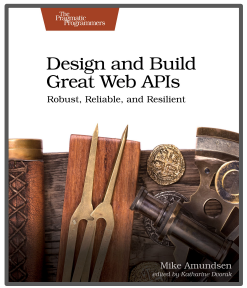


Figure 5-9. REST Derivation by Style Constraints



Web APIs -- REST

- List of System Properties
  - Performance
  - Scalability
  - Simplicity
  - Modifiability
  - Visibility
  - Portability
  - Reliability
- List of Implementation Constraints
  - Client-Server
  - Stateless
  - Caching
  - Uniform Interface
  - Layered System
  - Code on Demand

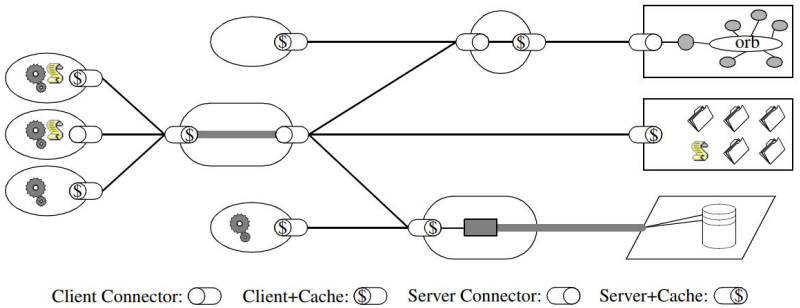
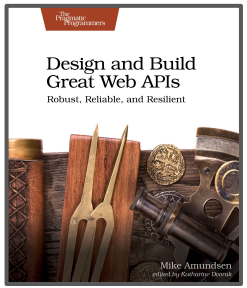
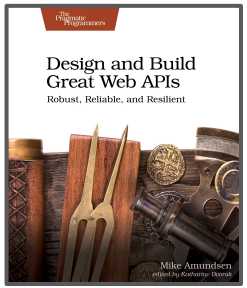


Figure 5-8. REST



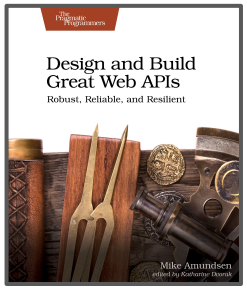
Web APIs

- HTTP (messages w/ intention)
- The Web (common practices for linking)
- REST (set of properties & constraints for building)



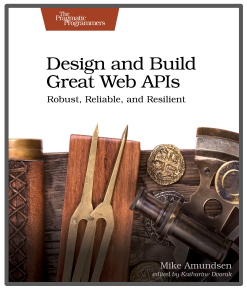
Exploring APIs

- BigCo & the Onboarding API
- Using **curl**



Designing APIs

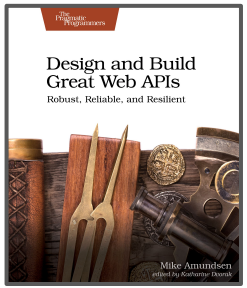
- Design Thinking
- API Design Process
- Using **schema.org**



Designing APIs - Design Thinking

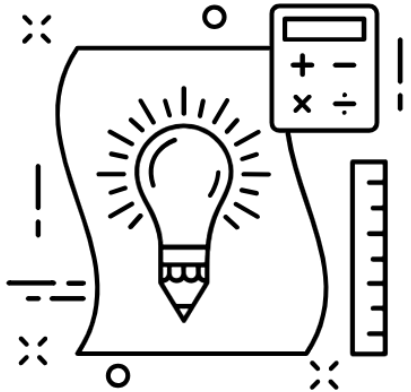
"A design discipline that uses the designer's sensibility and methods to match people's needs with what is technologically feasible and what a viable business strategy can convert into customer value and market opportunity."

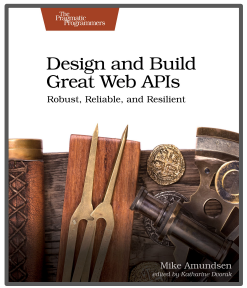
-- Tim Brown, CEO of IDEO



Designing APIs - Design Thinking

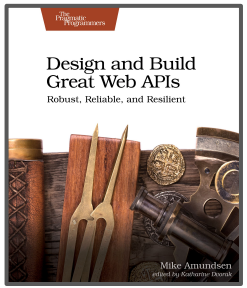
- Match people's needs
- Technologically feasible
- Viable business
- Customer value





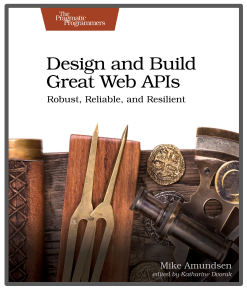
Designing APIs - API Design Process

- List all data and action names
- Create a workflow diagram
- Normalize your naming
- Write up an API Profile document



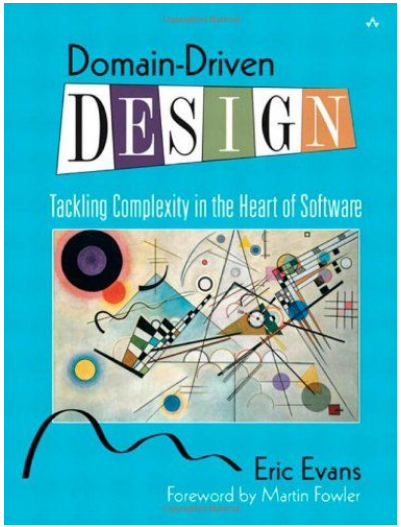
Designing APIs - List data and elements

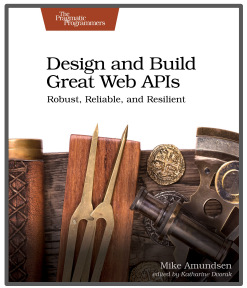
- Refer to the supplied onboarding-story.md
- What are the "things" or "properties"?
- What are the "tasks" or "actions"?
- Are any properties required?
- Are any properties enumerated?



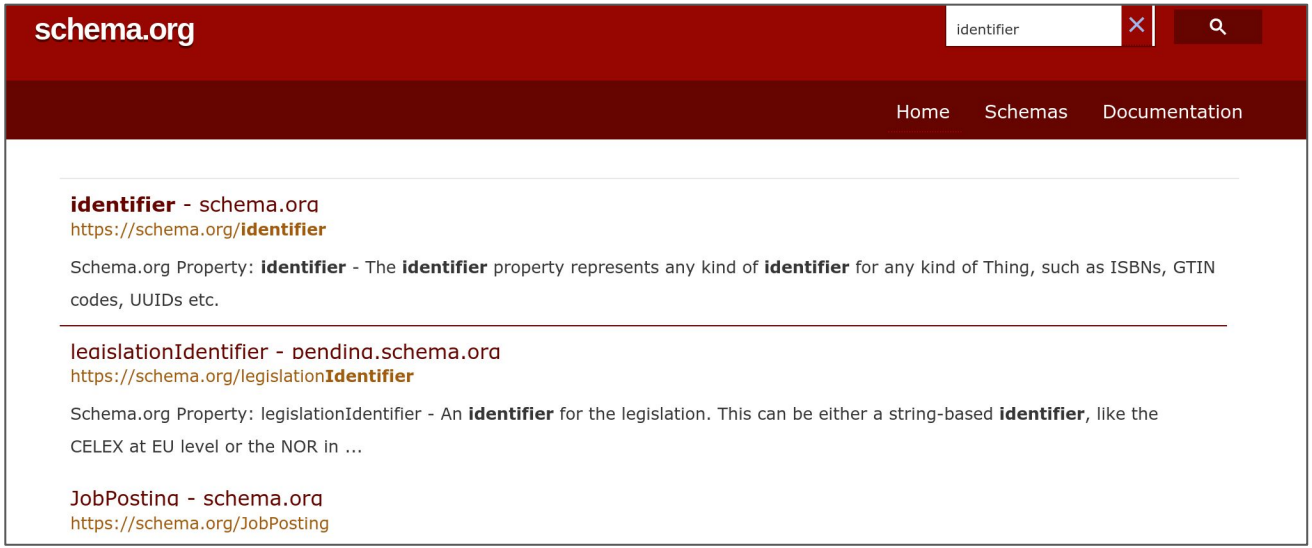
Designing APIs - Using **schema.org**

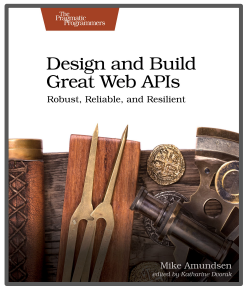
- Schema.org is an open web vocabulary
- Launched in 2011 by Google, Yahoo, Microsoft, Yandex
- Common vocabulary/glossary/terms
- Domain-Driven Design calls it "ubiquitous language"
- Use schema.org to "reconcile names" in your API





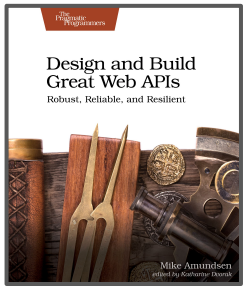
Designing APIs - Using **schema.org**





Diagramming APIs

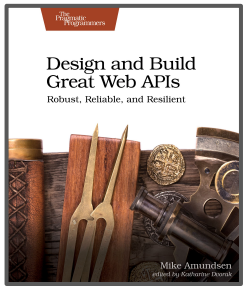
- API Workflow
- Using **websequencediagrams.com**



Diagramming APIs - API Workflow

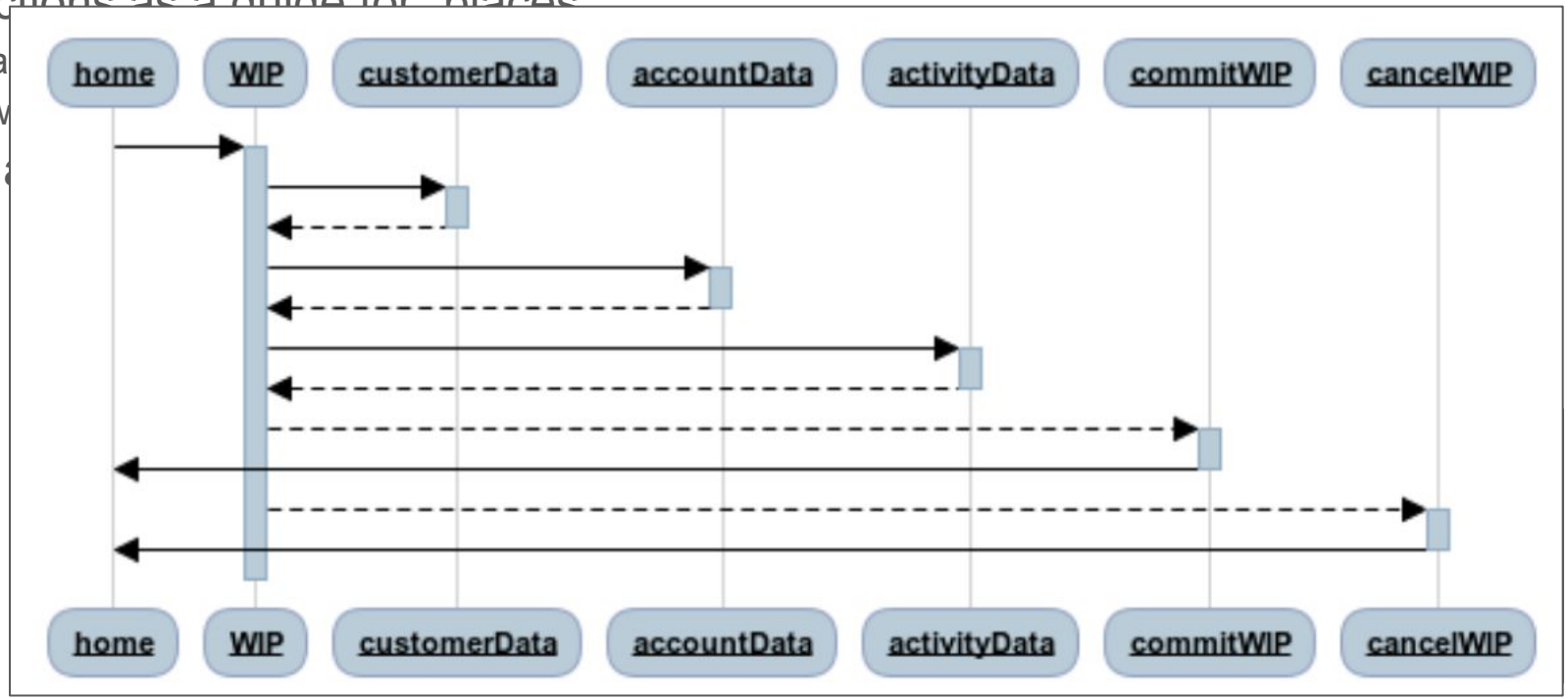
- Establish a "home"
- Use actions as a guide for "places"
  - Each action happens at a place
  - Always return "home"
- Rinse and repeat

```
home->+WIP:
WIP->+customerData:
customerData-->-WIP:
WIP->+accountData:
accountData-->-WIP:
WIP->+activityData:
activityData-->-WIP:
WIP-->+commitWIP:
commitWIP->-home:
WIP-->+cancelWIP:
cancelWIP->-home:
```



Diagramming APIs - API Workflow

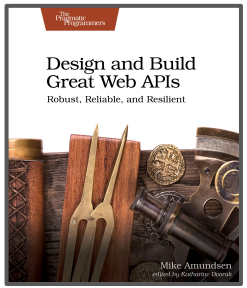
- Establish a "home"
- Use actions as a guide for "places"
  - Ea
  - Alw
- Rinse a



Diagramming APIs - API Workflow

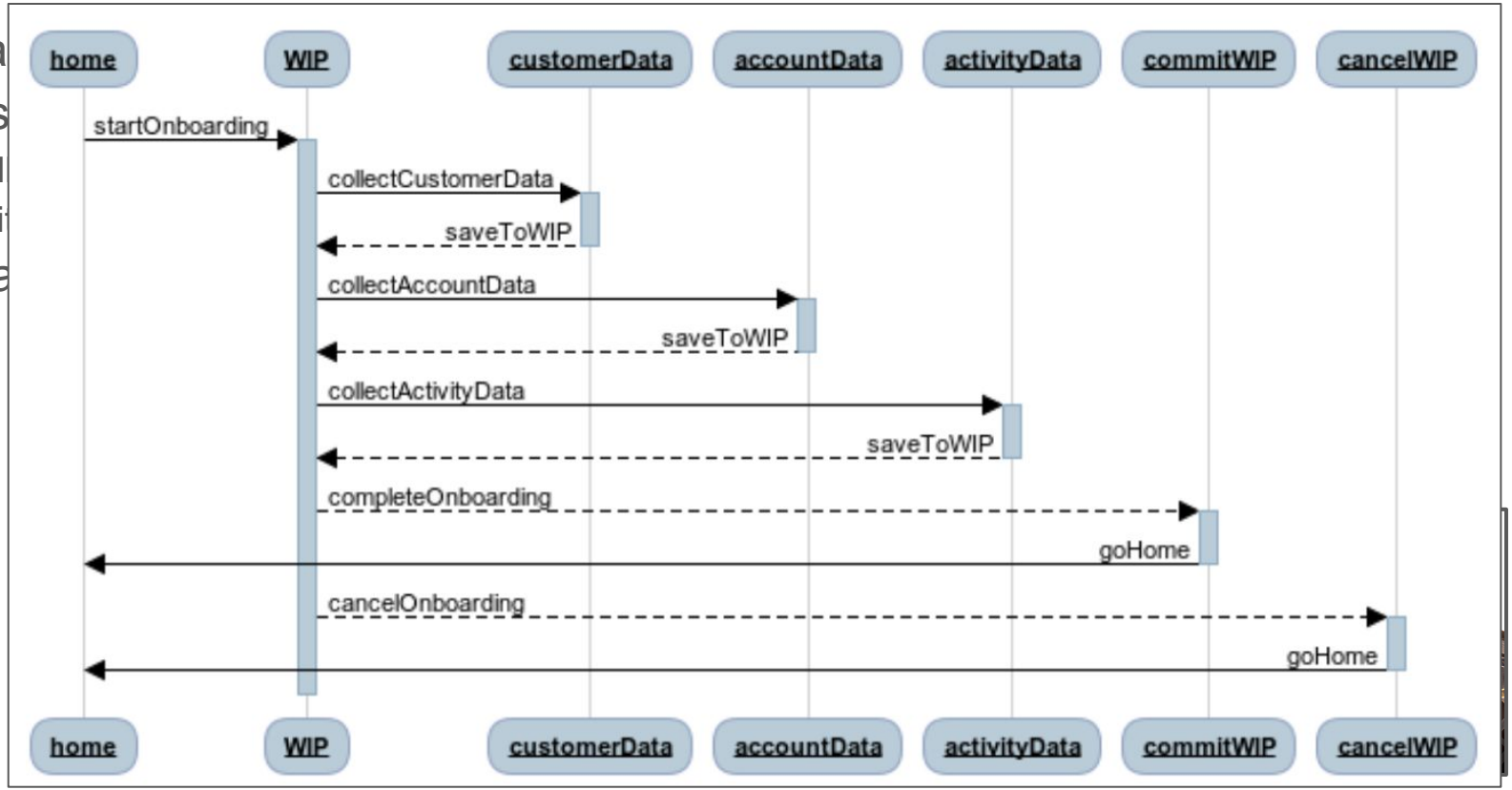
- Decorate each transition with an action
- Actions have two parts
  - Collect data
  - Write/send data
- Rinse and repeat

```
home->+WIP: startOnboarding
WIP->+customerData: collectCustomerData
customerData-->-WIP: saveToWIP
WIP->+accountData: collectAccountData
accountData-->-WIP:saveToWIP
WIP->+activityData: collectActivityData
activityData-->-WIP:saveToWIP
WIP-->+commitWIP:completeOnboarding
commitWIP->-home:goHome
WIP-->+cancelWIP:cancelOnboarding
cancelWIP->-home:goHome
```



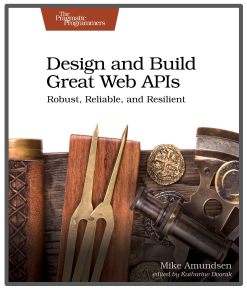
Diagramming APIs - API Workflow

- Decora
- Actions
  - Col
  - Writ
- Rinse a



Diagramming APIs - API Workflow

- Add arguments to each action
  - List args for reading
  - List args for sending
- Rinse and repeat



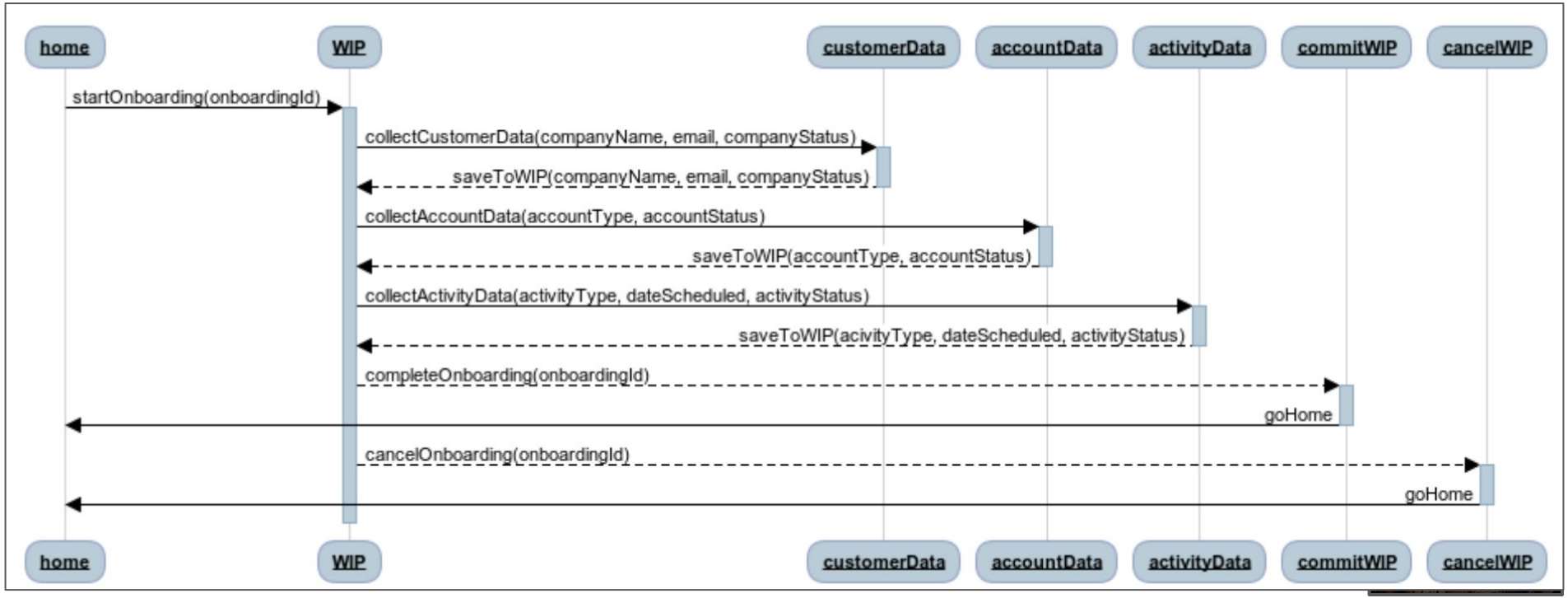
Diagramming APIs - API Workflow

- Add arguments to each action
  - List args for reading
  - List args for sending
- Rinse and repeat

```
home->+WIP: startOnboarding(onboardingId)
WIP->+customerData: collectCustomerData(companyName, email, companyStatus)
customerData-->-WIP: saveToWIP(companyName, email, companyStatus)
WIP->+accountData: collectAccountData(accountType, accountStatus)
accountData-->-WIP:saveToWIP(accountType, accountStatus)
WIP->+activityData: collectActivityData(activityType, dateScheduled, activityStatus)
activityData-->-WIP:saveToWIP(acivityType, dateScheduled, activityStatus)
WIP-->+commitWIP:completeOnboarding(onboardingId)
commitWIP->-home:goHome
WIP-->+cancelWIP:cancelOnboarding(onboardingId)
cancelWIP->-home:goHome
```

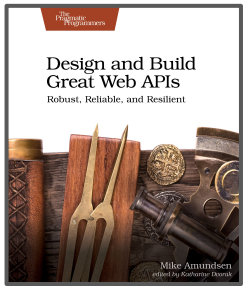
Diagramming APIs - API Workflow

- Add arguments to each action

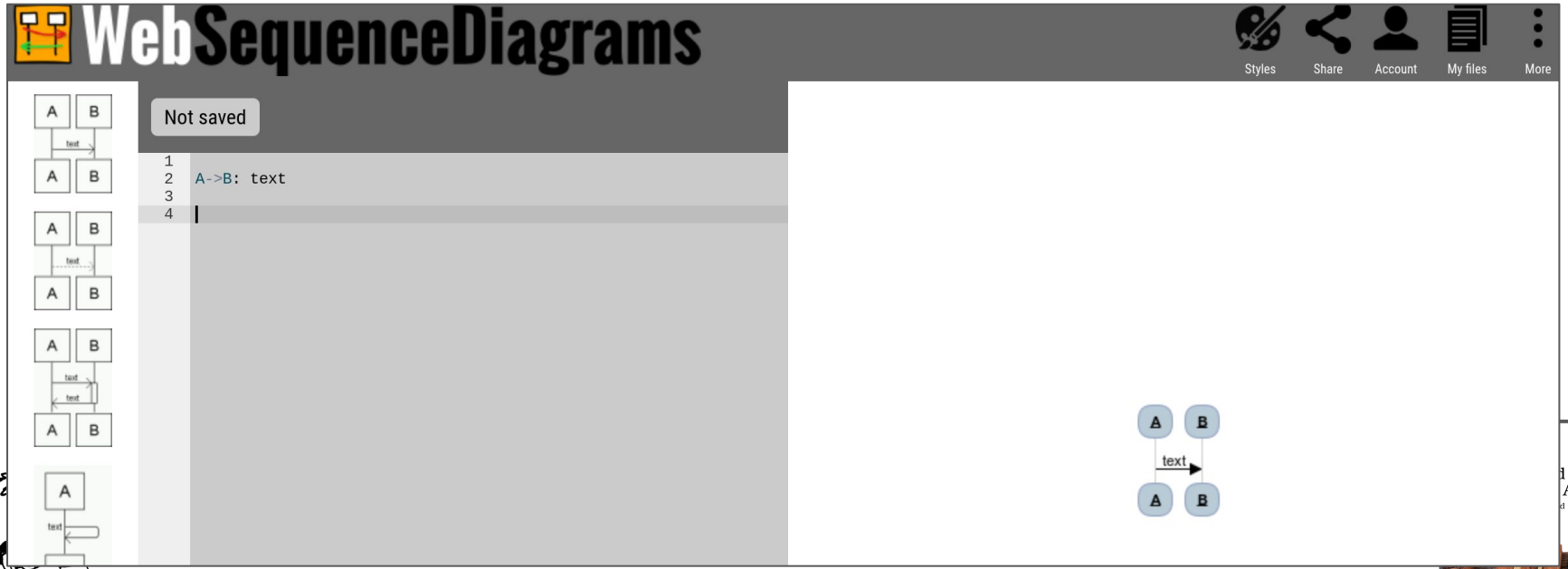


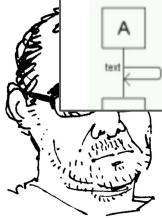
Diagramming APIs - Using WSD

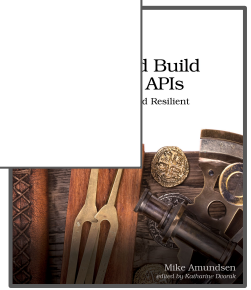
- No need to install
  https://www.websequencediagrams.com/
- Documentation
  https://www.websequencediagrams.com/examples.html
- Create an account (optional)
  https://www.websequencediagrams.com/ (and select "account")



Diagramming APIs - Using WSD





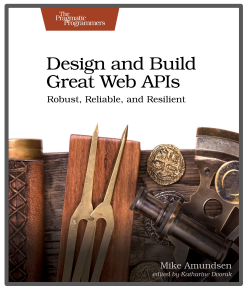


Diagramming APIs - Bonus Utility **`wsdgen`**

- Generates WSD images from command-line
- Download repo from github
  https://github.com/mamund/wsd-gen

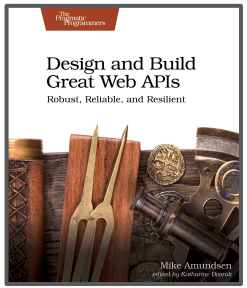
- Install using npm:
  `npm install -g wsdgen`

- Usage:
  `wsdgen onboardingAPI.wsd`



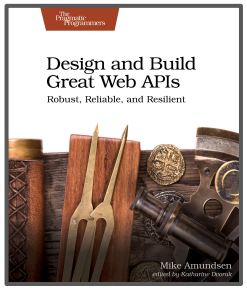
Describing APIs

- Descriptions vs. Definitions
- Data and Actions
- Using **ALPS**



Describing APIs - Descriptions vs. Definitions

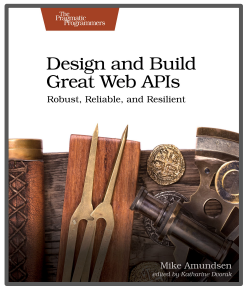
- Descriptions are implementation-agnostic
- Descriptions are the bridge between design and implementation
- Examples of API description formats
  - Dublin Code Application Profile (DCAP), 2009
  - Application-Level Profile Semantics (ALPS), 2011
  - JSON Home, 2012
  - Profiled Hypertext Application Language (PHTAL), 2019



Describing APIs - Data and Actions

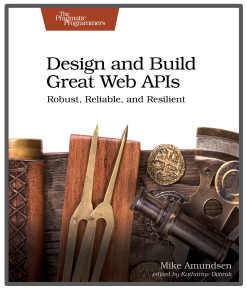
- List actions and properties
- Indicate action intentions
- DO NOT document
  - URLs
  - Protocol Methods
  - Message Formats

BTW - We'll deal with definition languages this afternoon!



Describing APIs - Using ALPS

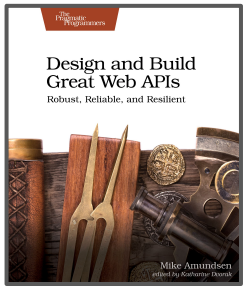
- Comes in two flavors (XML and JSON)
- Simple model
  - `<alps> <doc>...</doc> <descriptors /> </alps>`
  - `{"alps" : {"doc" : {"text" : "..."}, "descriptors" : [] } }`
- Descriptors detail actions or properties
  - `<descriptor id="..." name=""... type="..." href="..." />`
- Descriptor types are:
  - `semantic`
  - `safe`
  - `unsafe`
  - `idempotent`



Describing APIs - Using ALPS

- Start with a minimal ALPS XML doc

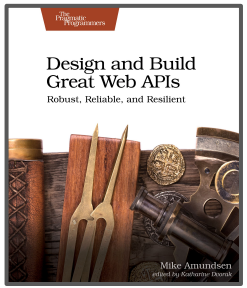
```
<alps version="1.0">
    <doc>Sample ALPS Document</doc>

    <!-- properties -->

    <!-- actions -->
</alps>
```



Describing APIs - Using ALPS

- List properties (API vocabulary)

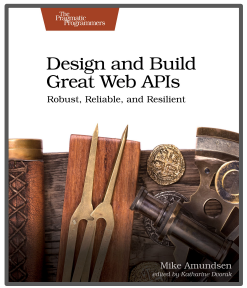
```
<alps version="1.0">
    <doc>Sample ALPS Document</doc>

    <!-- properties -->
    <descriptor id="onboardingId" />
    <descriptor id="status" />
    <descriptor id="dateCreated" />
    <descriptor id="dateUpdated" />

    <!-- actions -->
</alps>
```



Describing APIs - Using ALPS

- List Actions (API vocabulary)

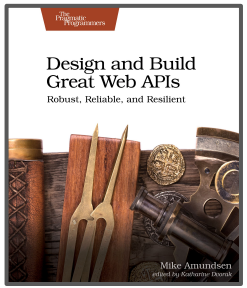
```
<alps version="1.0">
    <doc>Sample ALPS Document</doc>

    <!-- properties -->
    <descriptor id="onboardingId" />
    <descriptor id="status" />
    <descriptor id="dateCreated" />
    <descriptor id="dateUpdated" />

    <!-- actions -->
    <descriptor id="StartOnboarding" type="unsafe" />
    <descriptor id="CollectionCompanyData" type="safe" />
    <descriptor id="SaveCompanyData" type="unsafe" />

</alps>
```



Describing APIs - Using ALPS

- Optionally, add resources and containers

```
<!-- actions -->
<descriptor id="StartOnboarding" type="unsafe" rt="work-in-progress" />
<descriptor id="CollectionCompanyData" type="safe" />
<descriptor id="SaveCompanyData" type="unsafe" rt="work-in-progress"/>

<!-- containers -->
<descriptor id="work-in-progress">
    <descriptor href="#onboardingId" />
    <descriptor href="#status" />
    <descriptor href="#dateCreated" />
    <descriptor href="#dateUpdated" />
<descriptor>

<!-- resources -->
<descriptor id="home">
    <descriptor href="#work-in-progress" />
    <descriptor href="StartOnboarding" />
</descriptor>
```

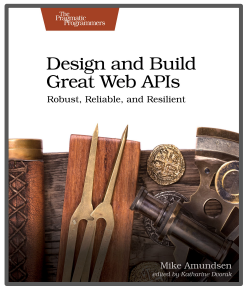


Diagramming APIs - Bonus Utility `wsd2alps`

- Generates ALPS (JSON) from WSD on command-line
- Download repo from github
  https://github.com/mamund/wsd-util

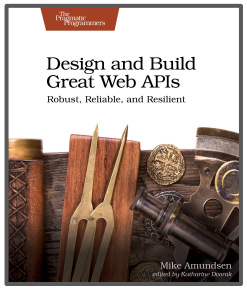
- Install using npm:
  ```
  npm install -g ./
  ```

- Usage:
  ```
  wsd2alps onboardingAPI.wsd
  ```



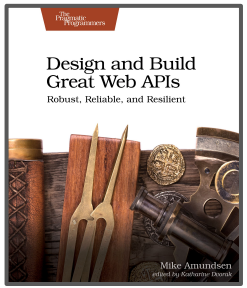
Defining APIs

- Sketching to Discover
- Prototyping to Define

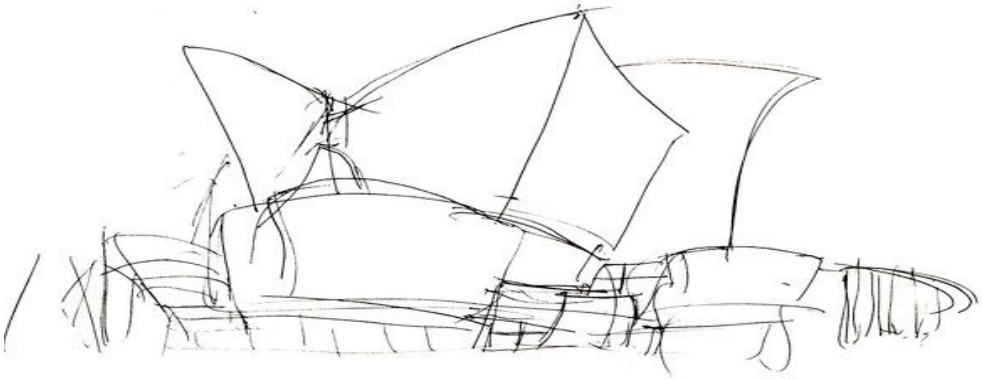


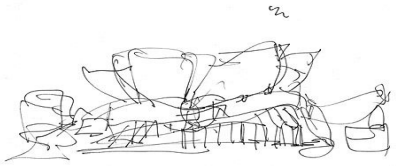
Sketching APIs

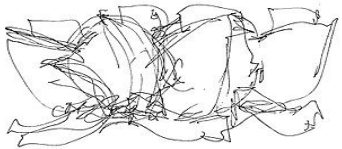
- Sketching
- Using **Blueprint**

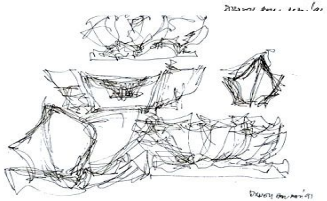


Frank Gehry Sketches







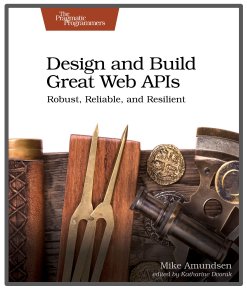


Frank Gehry on Design

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

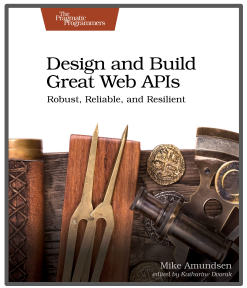
Sketching APIs -- Sketching

- 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.



Sketching APIs -- Sketching

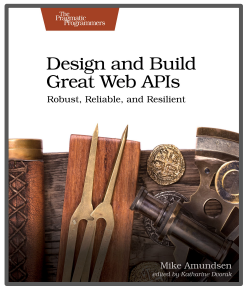
- Create a sketch (using **Blueprint**).
- Show it to others (devs, stakeholders) and get their feedback.
- If possible use simple API consumer tools (curl, NodeJS, etc.) to test.
- Continue to modify the simple sketches as needed



Sketching APIs -- Using Blueprint

- Created in 2013 by Jakub Nesetril
- Focused on quickly mocking API request/response
- Based on Markdown
- Sold to Oracle in 2017

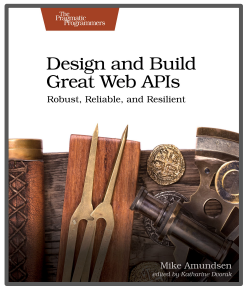
apiary



Sketching APIs -- Using Blueprint

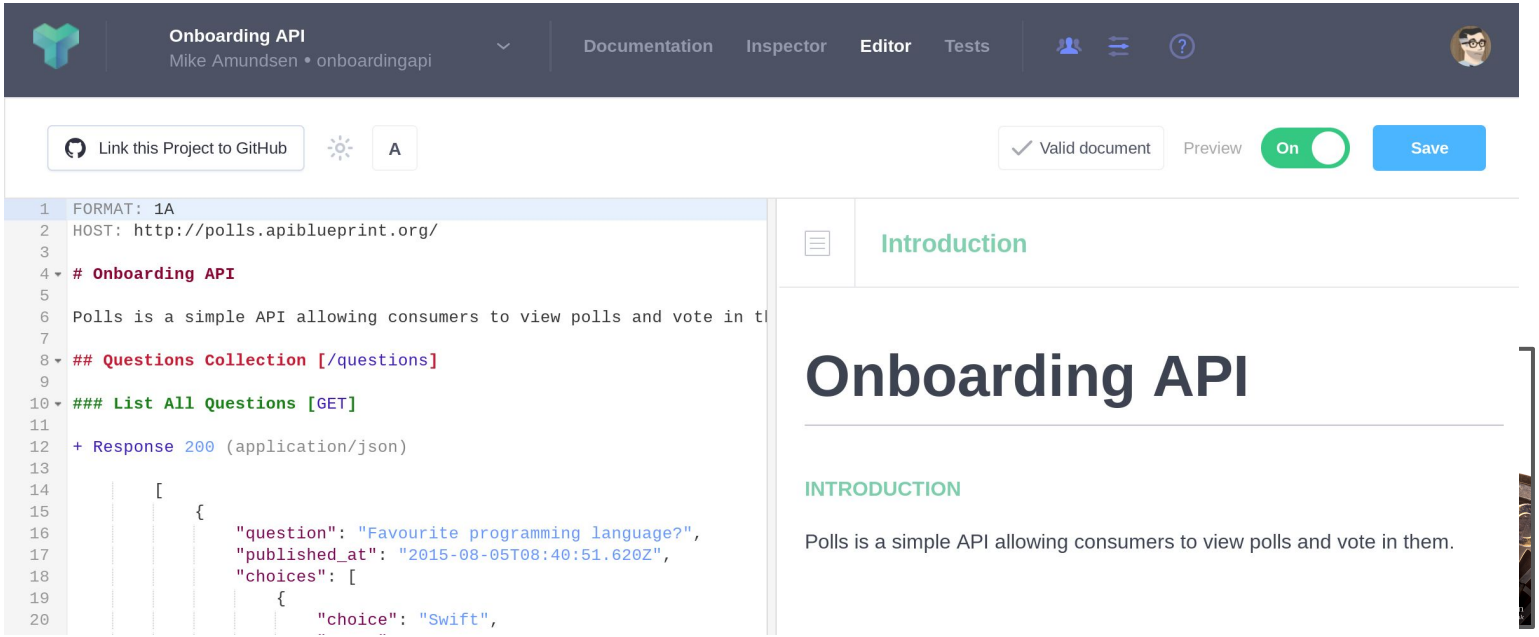
- No download needed
  https://app.apiary.io/onboardingapi/editor
- Documentation
  https://help.apiary.io/tools/apiary-editor/
- Create Account (optional)
  https://login.apiary.io/

apiary

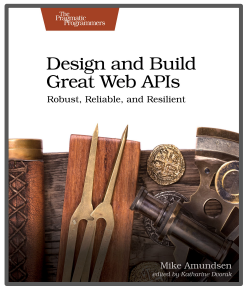


Sketching APIs -- Using Blueprint

- Write (or copy/paste) APIB doc into Editor
- Copy/Paste into local doc to save to disk

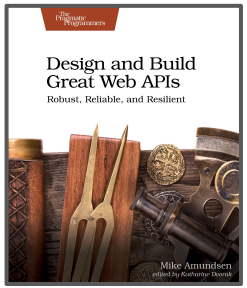


Sketches are made to be thrown away.



Prototype APIs

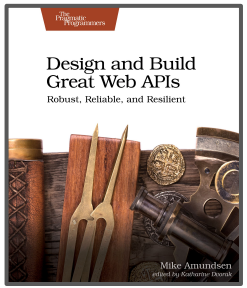
- Prototyping
- Using **Swagger**



Prototyping in Sculpture

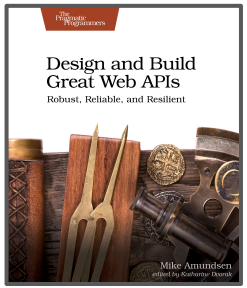
Prototype APIs -- Prototyping

- 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.



Prototype APIs -- Prototyping

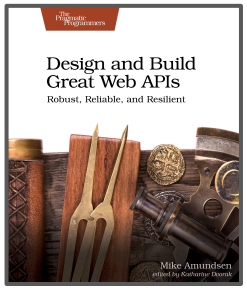
- Select a likely sketch
- Create a prototype of it (using **Swagger**).
- Show it to others (devs, stakeholders) and get their feedback.
- If possible, use production-level API consumer tools to test.
- Continue to modify the simple sketches as needed



Prototype APIs -- Using Swagger

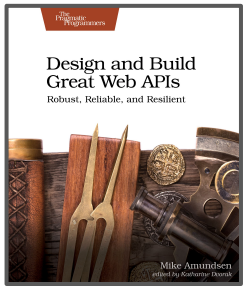
- Swagger (AKA OpenAPI)
- Created in 2011 by Tony Tam (Wordnik)
- Focused on auto-generated Docs & SDKs
- Open API Initiative created in 2015 (Linux Foundation)

OPENAPI
INITIATIVE



Prototype APIs -- Using Swagger

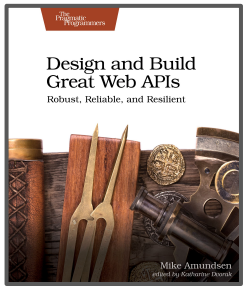
- No download needed
  https://editor.swagger.io/
- Documentation
  https://swagger.io/docs/
- Create an account (optional)
  https://app.swaggerhub.com



Prototype APIs -- Using Swagger

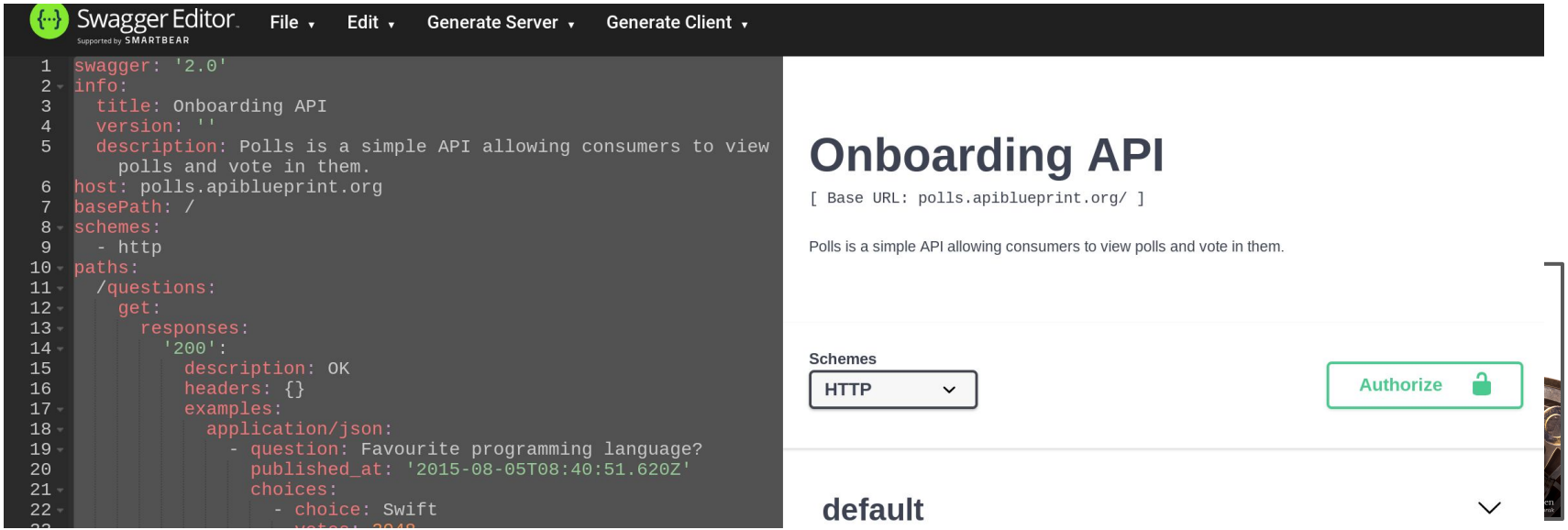
- Convert APIB to Swagger
  https://github.com/kminami/apib2swagger

```
apib2swagger -i onboardingAPI.apib -o onboarding.json

apib2swagger -i onboardingAPI.apib --yaml -o onboarding.json
```



Prototype APIs -- Using Swagger

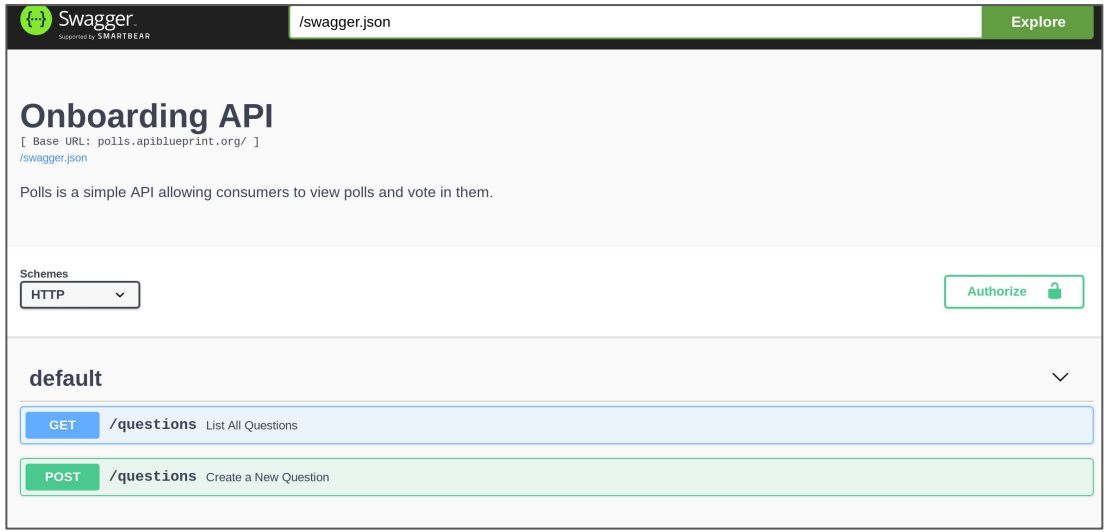
- Copy/Paste into Swagger Editor



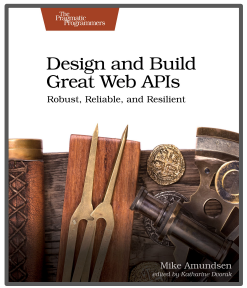
Prototype APIs -- Using Swagger

- Generate SwaggerUI documentation

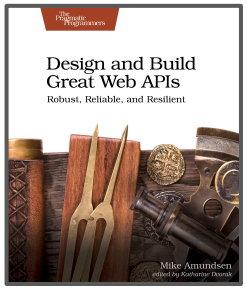
```
apib2swagger -i onboarding.apib -s -p 3000
```



OPENAPI INITIATIVE

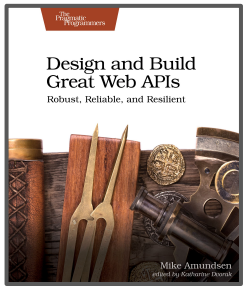


Prototypes are made to be tested.



Morning -- Summary

- Session One
  - Web APIs (HTTP, WWW, REST)
  - Exploring APIs (curl)
  - Designing APIs (Design Process)
- Session Two
  - Importance of Iteration
  - Sketching (apib)
  - Prototyping (Swagger)



Building Great APIs
from the Ground Up

Morning

Mike Amundsen
@mamund