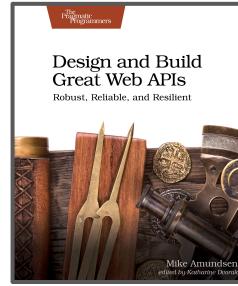


Building Great APIs from the Ground Up

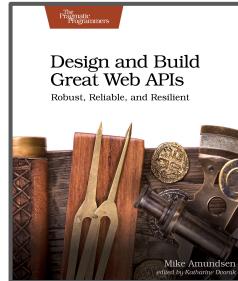
GOTO Amsterdam

Mike Amundsen
@mamund



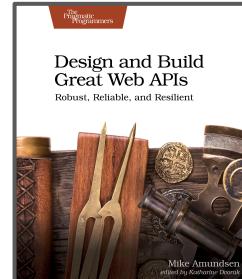
Morning

Design and Define



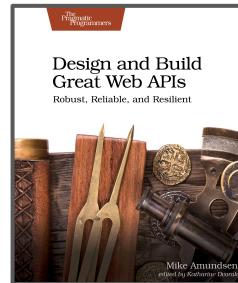
Design APIs

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



Exploring APIs -- BigCo, Inc

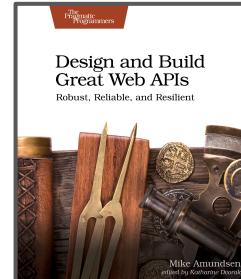
The Bayesian International Group of Companies (BigCo) was founded in 1827 in Scotland. Originally focused on demography and finance based on the work of Richard Price, by the 1900s BigCo was a manufacturer of important turn-of-the-century technologies including mobile x-ray machines and hydrophones by 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.

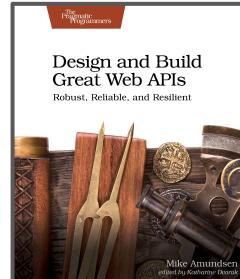
- Stories : <https://github.com/mamund/2019-06-goto-amsterdam/tree/master/stories>
- Company
- Accounts
- Activity



Exploring APIs -- Using curl



command line tool and library
for transferring data with URLs



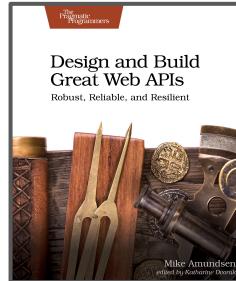
Exploring APIs -- Using curl

- 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

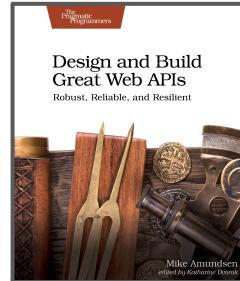
This folder holds results of various requests to running services.

- **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>



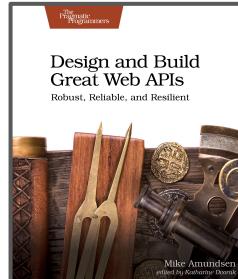
Web APIs

- HTTP
- The Web
- REST



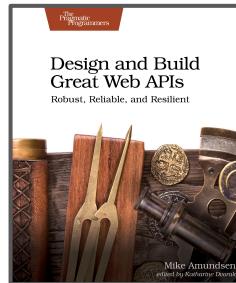
Web APIs -- HTTP (protocol)

- 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)

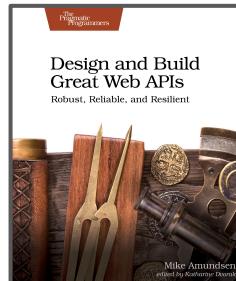
REQUEST

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

RESPONSE



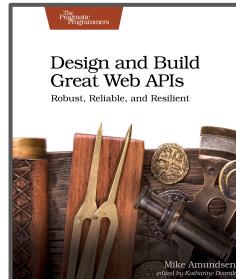
```
HTTP/2.0 200 OK
Date: Mon, 27 Jul 2019 12:28:53 GMT
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: application/json
Connection: Closed
```



Web APIs -- HTTP (methods)

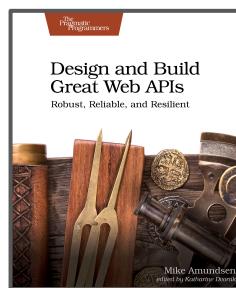
- HTTP clients use **Methods** to communicate intention
- GET, PUT, POST, DELETE
- Methods are NOT functions
- There are lots of registered methods

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



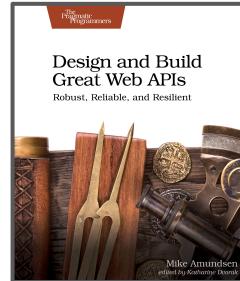
Web APIs -- HTTP (methods)

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]
UPDATEDIRECTREF	no	yes	[RFC4437, Section 7]
VERSION-CONTROL	no	yes	[RFC3253, Section 3.5]



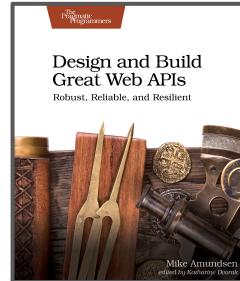
Web APIs -- HTTP (other stuff)

- 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)

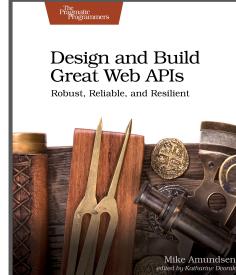


Web APIs -- HTTP (other stuff)

- **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)

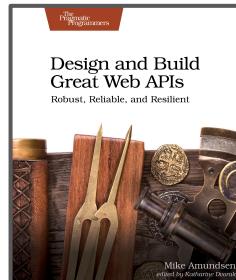
The bank-transfer dilemma

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

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

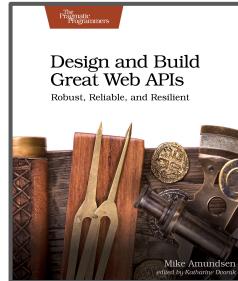
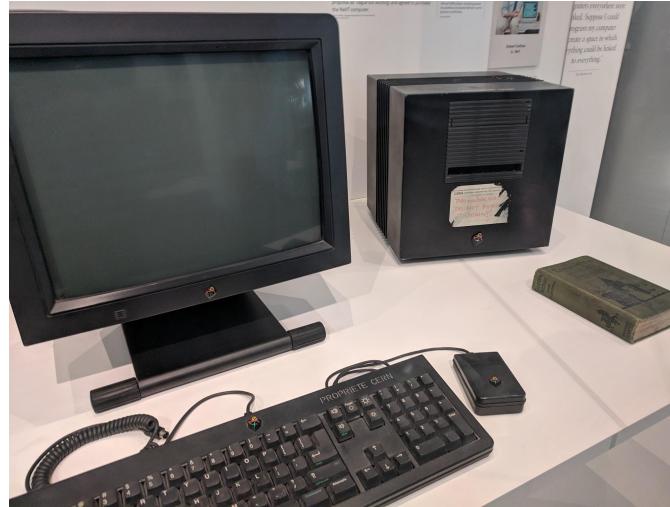


What if I never get a response?



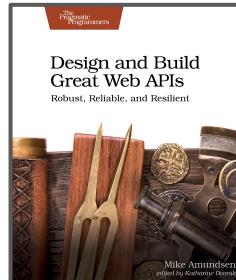
Web APIs -- The Web

- 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

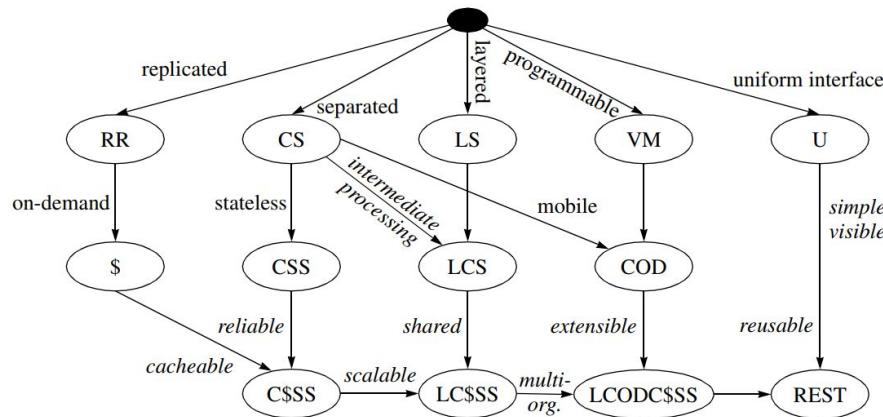
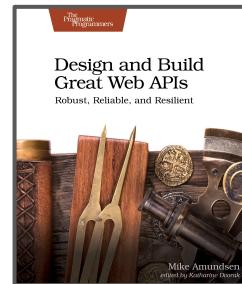


Figure 5-9. REST Derivation by Style Constraints



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

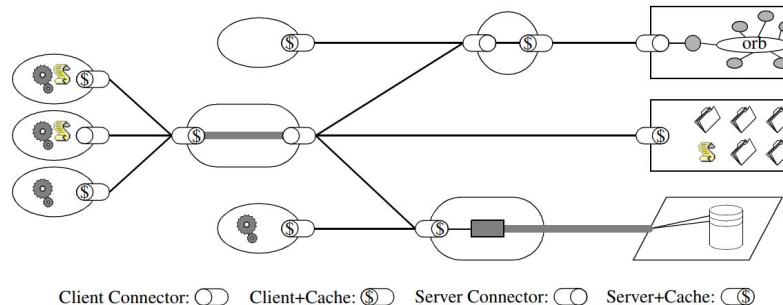
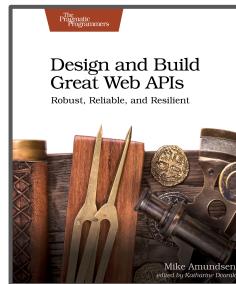
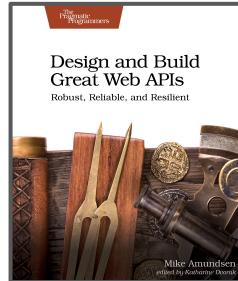


Figure 5-8. REST



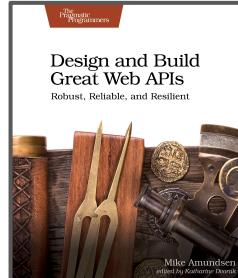
Web APIs

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



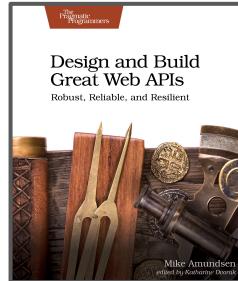
Exploring APIs

- BigCo & the Onboarding API
- Using `curl`



Designing APIs

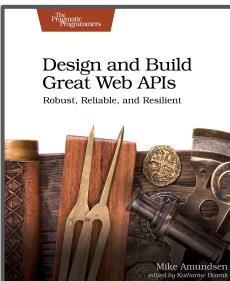
- Design Thinking
- API Design Process
- Using **schema.org**



Designing APIs - Design Thinking

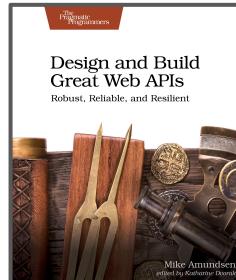
"A design discipline that uses the designer's sensibility and methods to match people's needs with what is technologically feasible and what a viable business strategy can convert into customer value and market opportunity."

-- Tim Brown, CEO of IDEO



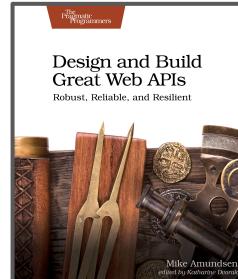
Designing APIs - Design Thinking

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



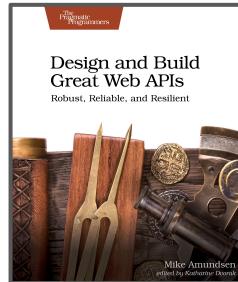
Designing APIs - API Design Process

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



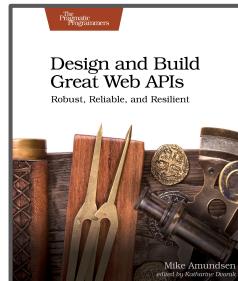
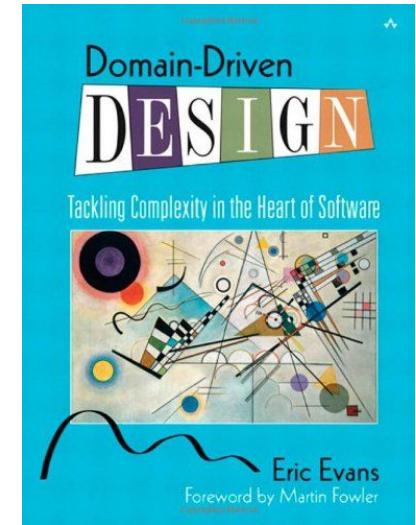
Designing APIs - List data and elements

- 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

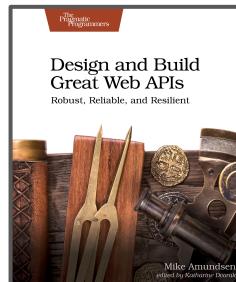


The screenshot shows the schema.org website interface. At the top, there is a red header bar with the "schema.org" logo on the left and a search bar containing the word "identifier" on the right. Below the header, there is a dark red navigation bar with three links: "Home", "Schemas", and "Documentation". The main content area has a white background. It displays a list of schema properties:

- identifier - schema.org**
<https://schema.org/identifier>

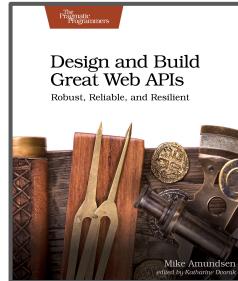
Schema.org Property: **identifier** - The **identifier** property represents any kind of **identifier** for any kind of Thing, such as ISBNs, GTIN codes, UUIDs etc.
- legislationIdentifier - pending.schema.org**
<https://schema.org/legislationIdentifier>

Schema.org Property: **legislationIdentifier** - An **identifier** for the legislation. This can be either a string-based **identifier**, like the CELEX at EU level or the NOR in ...
- JobPosting - schema.org**
<https://schema.org/JobPosting>



Diagramming APIs

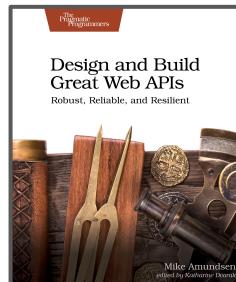
- API Workflow
- Using websequencediagrams.com



Diagramming APIs - API Workflow

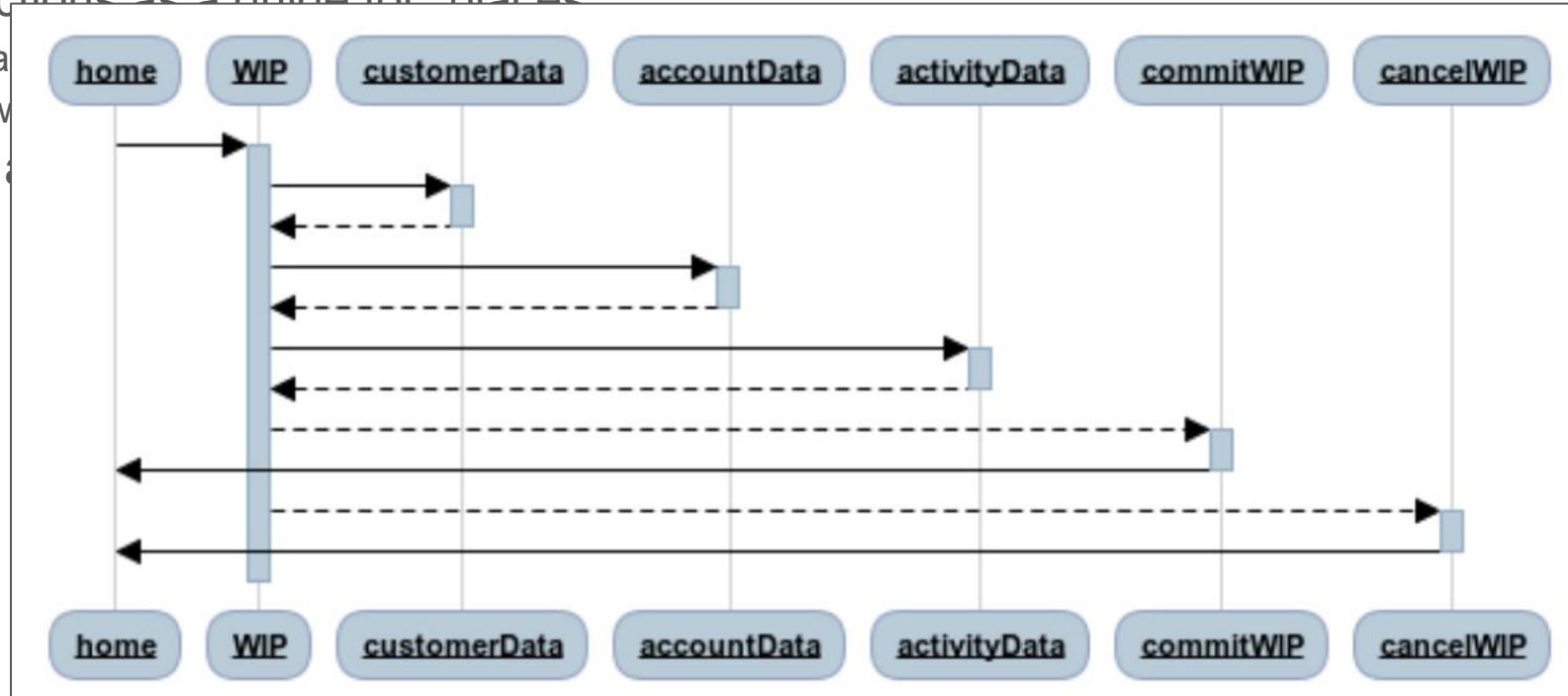
- Establish a "home"
- Use actions as a guide for "places"
 - Each action happens at a place
 - Always return "home"
- Rinse and repeat

```
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

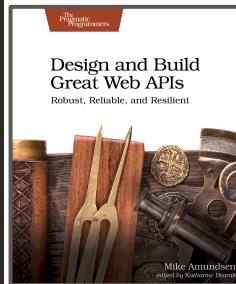
- Establish a "home"
- Use actions as a guide for "places"
 - Early
 - Always
- Rinse and repeat



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
```



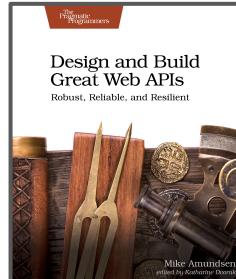
Diagramming APIs - API Workflow

- Decorate
- Actions
 - Collect
 - Write
- Rinse and repeat



Diagramming APIs - API Workflow

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



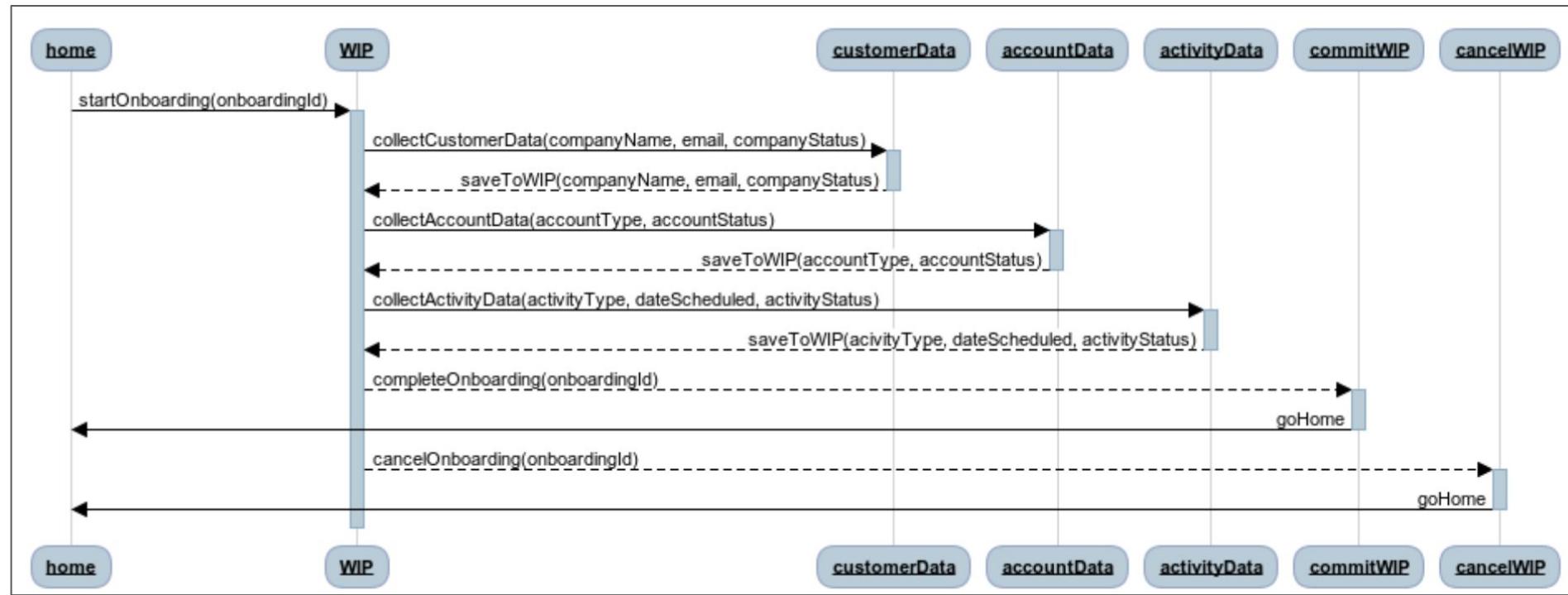
Diagramming APIs - API Workflow

- 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(activityType, 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

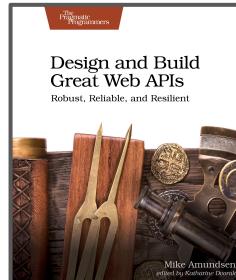
<https://www.websequencediagrams.com/>

- Documentation

<https://www.websequencediagrams.com/examples.html>

- Create an account (optional)

<https://www.websequencediagrams.com/> (and select "account")



Diagramming APIs - Using WSD

The screenshot shows the WebSequenceDiagrams (WSD) application interface. At the top, there is a navigation bar with icons for Styles, Share, Account, My files, and More. The main area has a title "WebSequenceDiagrams" with a yellow icon. A message "Not saved" is displayed above a sequence diagram editor. The editor shows a sequence of interactions between two participants, A and B. The interactions are labeled "text". On the left, there is a sidebar with a list of recent diagrams and a cartoon illustration of a person thinking.

Not saved

```
1  
2 A->B: text  
3  
4 |
```

A sequence diagram showing interactions between participants A and B. The interactions are labeled "text".

Build APIs
and Resilient

Mike Amundsen
Project by Kachmarj (Kong)

Diagramming APIs - Bonus Utility `wsdgen`

- Generates WSD images from command-line
- Download repo from github

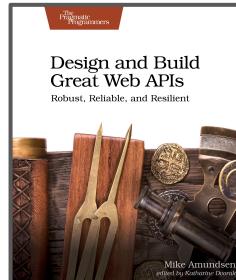
<https://github.com/mamund/wsd-gen>

- Install using npm:

```
npm install -g wsdgen
```

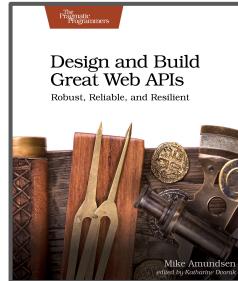
- Usage:

```
wsdgen onboardingAPI.wsd
```



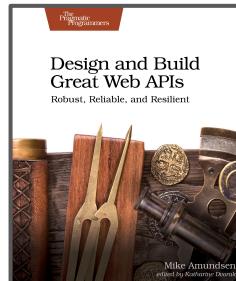
Describing APIs

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



Describing APIs - Descriptions vs. Definitions

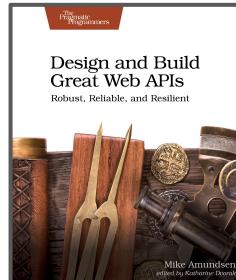
- Descriptions are implementation-agnostic
- Descriptions are the bridge between design and implementation
- Examples of API description formats
 - Dublin Code Application Profile (DCAP), 2009
 - Application-Level Profile Semantics (ALPS), 2011
 - JSON Home, 2012
 - Profiled Hypertext Application Language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



Describing APIs - Data and Actions

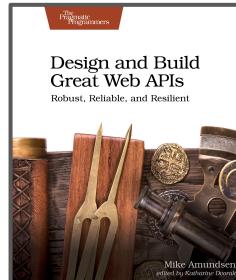
- List actions and properties
- Indicate action intentions
- DO NOT document
 - URLs
 - Protocol Methods
 - Message Formats

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



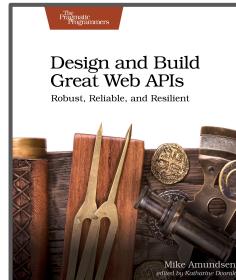
Describing APIs - Using ALPS

- Start with a minimal ALPS XML doc

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

    <!-- properties -->

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



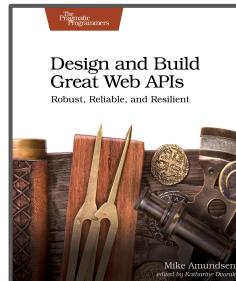
Describing APIs - Using ALPS

- 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

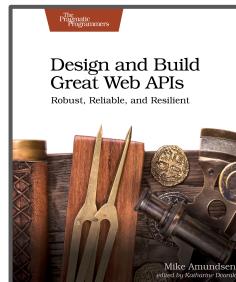
- List Actions (API vocabulary)

```
<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>
```



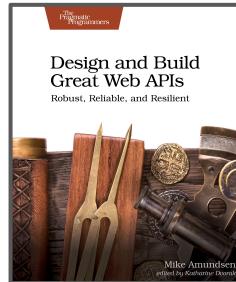
Describing APIs - Using ALPS

- 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

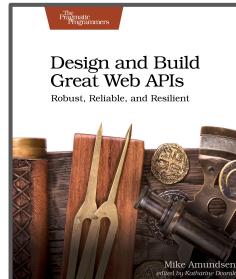
<https://github.com/mamund/wsd-util>

- Install using npm:

```
npm install -g ./
```

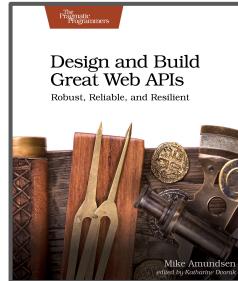
- Usage:

```
wsd2alps onboardingAPI.wsd
```



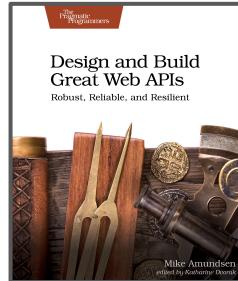
Defining APIs

- Sketching to Discover
- Prototyping to Define

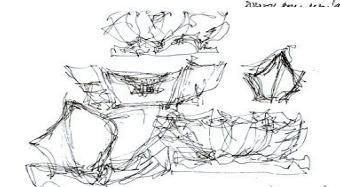
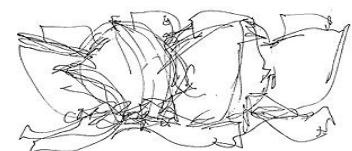
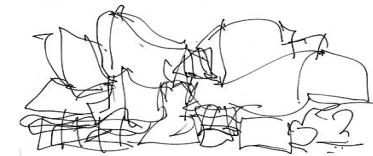
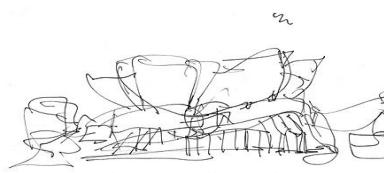
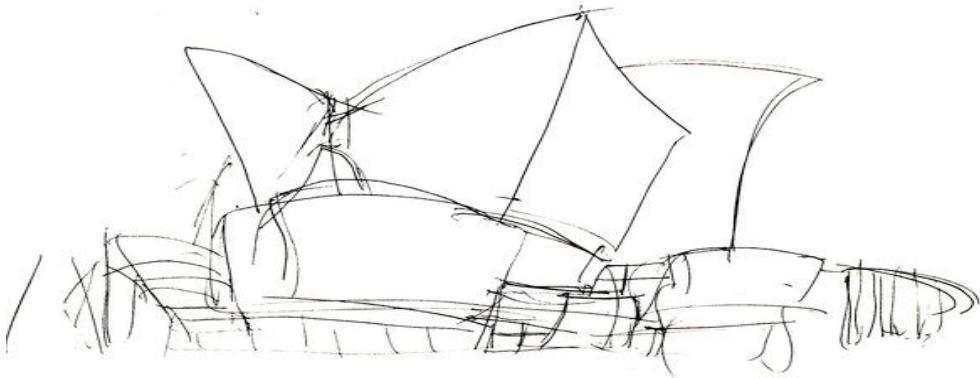


Sketching APIs

- Sketching
- Using **Blueprint**



Frank Gehry Sketches



frank Gehry
Gehry Hall
problems

frank Gehry

Frank Gehry on Design

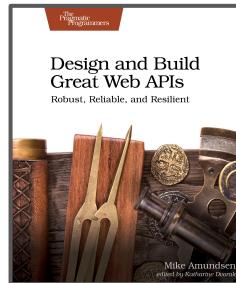
An architect is given a program, budget, place, and schedule. Sometimes the end product rises to art



Frank Gehry

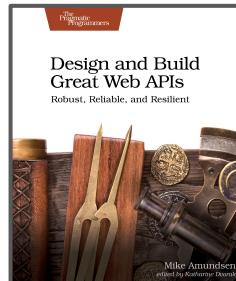
Sketching APIs -- Sketching

- 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

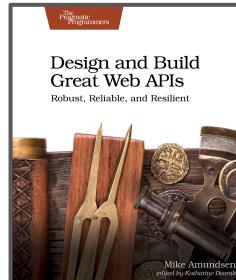


Sketching APIs -- Using Blueprint

- 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

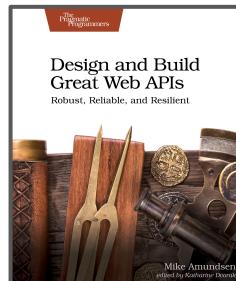
<https://help.apiary.io/tools/apiary-editor/>

- 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



The screenshot shows the Blueprint API editor interface. At the top, there's a header with a logo, the title "Onboarding API", the author "Mike Amundsen • onboardingapi", and navigation links for "Documentation", "Inspector", "Editor" (which is selected), "Tests", and user profile. Below the header are buttons for "Link this Project to GitHub", "A", and "Valid document" (with a green checkmark). On the right, there are "Preview", "On" (switch), and "Save" buttons.

The main area contains the API definition code:

```
1 FORMAT: 1A
2 HOST: http://polls.apiblueprint.org/
3
4 # Onboarding API
5
6 Polls is a simple API allowing consumers to view polls and vote in them.
7
8 ## Questions Collection [/questions]
9
10 ### List All Questions [GET]
11
12 + Response 200 (application/json)
13
14 [
15   {
16     "question": "Favourite programming language?",
17     "published_at": "2015-08-05T08:40:51.620Z",
18     "choices": [
19       {
20         "choice": "Swift",
21       }
22     ]
23   }
24 ]
```

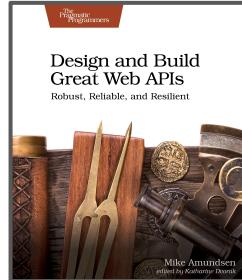
To the right of the code editor, the generated API documentation is shown:

Onboarding API

INTRODUCTION

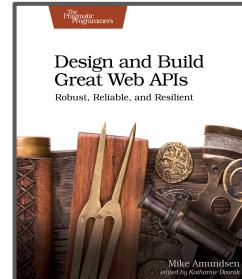
Polls is a simple API allowing consumers to view polls and vote in them.

Sketches are made to be thrown away.



Prototype APIs

- Prototyping
- Using **Swagger**

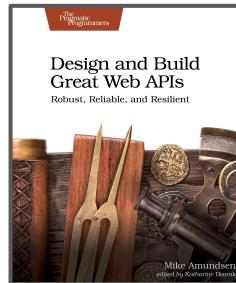


Prototyping in Sculpture



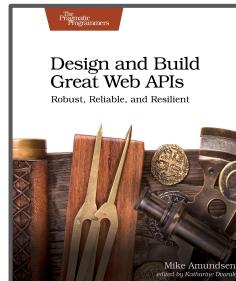
Prototype APIs -- Prototyping

- 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

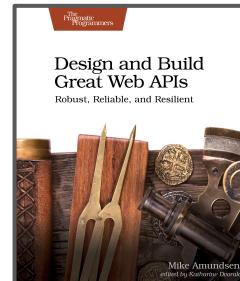


Prototype APIs -- Using Swagger

- 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

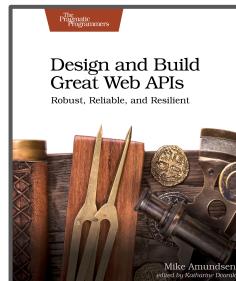
<https://swagger.io/docs/>

- Create an account (optional)

<https://app.swaggerhub.com>



OPENAPI
INITIATIVE



Prototype APIs -- Using Swagger

- Convert APIB to Swagger

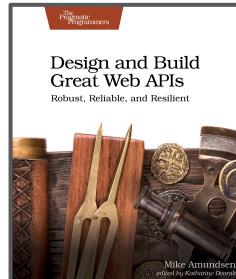
<https://github.com/kminami/apib2swagger>



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

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

```
|
```



Prototype APIs -- Using Swagger

- Copy/Paste into Swagger Editor



Swagger Editor. Supported by SMARTBEAR

```
1 swagger: '2.0'
2 info:
3   title: Onboarding API
4   version: ''
5   description: Polls is a simple API allowing consumers to view
       polls and vote in them.
6 host: polls.apiblueprint.org
7 basePath: /
8 schemes:
9   - http
10 paths:
11   /questions:
12     get:
13       responses:
14         '200':
15           description: OK
16           headers: {}
17           examples:
18             application/json:
19               - question: Favourite programming language?
20               published_at: '2015-08-05T08:40:51.620Z'
21               choices:
22                 - choice: Swift
23                 votes: 2048
```



Onboarding API
[Base URL: polls.apiblueprint.org/]

Polls is a simple API allowing consumers to view polls and vote in them.

Schemes Authorize 

default

Prototype APIs -- Using Swagger

- Generate SwaggerUI documentation

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



Swagger. Supported by SMARTBEAR

/swagger.json

Explore

Onboarding API

[Base URL: polls.apiblueprint.org/]
</swagger.json>

Polls is a simple API allowing consumers to view polls and vote in them.

Schemes

HTTP

Authorize

default

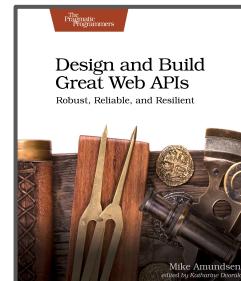
GET /questions List All Questions

POST /questions Create a New Question

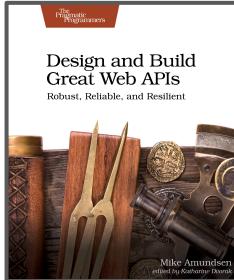
This screenshot shows the Swagger UI interface for the Onboarding API. It displays the API's documentation, including its base URL, the path to the Swagger JSON file, and a brief description. The UI includes dropdown menus for selecting schemes (HTTP) and authorization, and a sidebar for navigating through different API definitions (e.g., default). Below the sidebar, two API endpoints are listed: a GET request for '/questions' to list all questions, and a POST request for '/questions' to create a new question.



OPENAPI
INITIATIVE

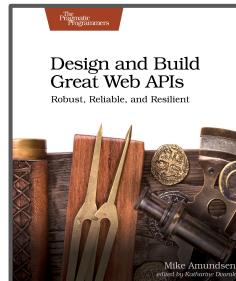


Prototypes are made to be tested.



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

