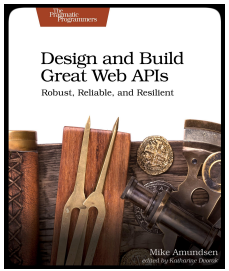
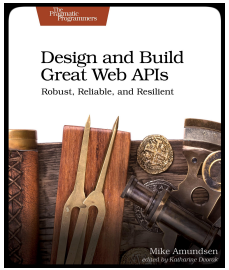
Designing Great APIs
Part Two

@mamund
Mike Amundsen

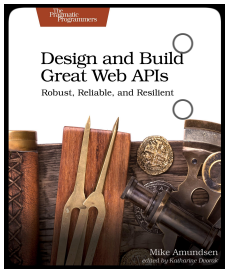


Welcome Back

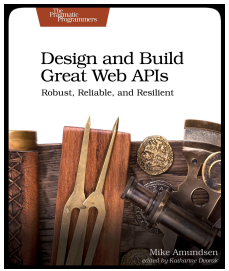


API Stories, Models and Diagrams

- The Importance of Stories
  - Every API starts with a story
  - Use the API Story Builder
- The Power of Models
  - Models help us focus on a solution
  - Normalize your properties against dictionaries
- The Value of Diagrams
  - Diagrams make our solution visible
  - Use WSD or some other diagram format

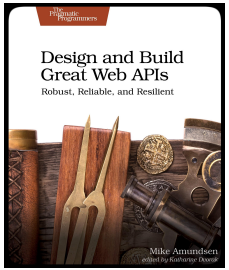


Assignment: Review



Overnight Assignment for API Design

- Pick an API story as a starter
- Produce the default API Model
- Normalize the Model against schema.org
- Create a WSD Diagram of the credit-check API



Overnight Assignment : API Story

13 lines (9 sloc) | 1.09 KB — Raw | Blame | History

Company Story at BigCo, Inc.

Purpose

We keep track of companies for BigCo, Inc.

Data

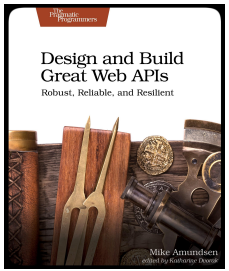
Data we include in a company record includes company name, street address, city, state/province, postal code, country, telephone, and email. Each record has a status value (pending, suspended, active, closed). We also track the date/time the record was created and the last date/time it was updated. We keep copies of the records, even after they have been deleted.

Actions

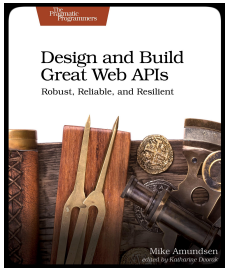
Typical work on the company records include getting the list of company records, reading a single record, creating, updating, and deleting records. You can also update the status of a single record. Finally, you can get a filtered list of all redords (support for filtering by status, by country, by state, and by company name).

Processing

Each company has a unique identifier in the system. Right now that is a combination of the first four letters of their company name and date the company was added to the system (in the format YYYYMMDD. We add another digit if adding that does not result in a unique identifier in the system.



Overnight Assignment : API Model



32 lines (28 sloc) | 661 Bytes — Raw | Blame | History

Company Vocabulary

Data Elements

- companyId
- companyName
- streetAddress
- city
- stateProvince
- postalCode
- country
- telephone
- email
- status (suspended, active, pending,closed)
- dateCreated
- dateUpdated

Action Elements

- list
- create
  - companyName[R], streetAddress, city, stateProvince, postalCode, country(US), telephone, email[R], status(pending)[R]
- read
  - companyId[R]
- update
  - companyId[R], companyName[R], streetAddress, city, stateProvince, postalCode, country(US), telephone, email[R], status[R]
- delete
  - companyId[R]
- filter
  - status, country, state/province, companyName

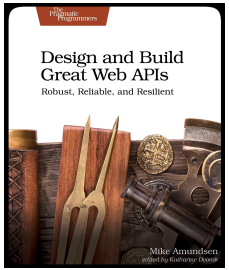
Overnight Assignment : Normalized API Model

32 lines (28 sloc) | 1.22 KB — Raw | Blame | History

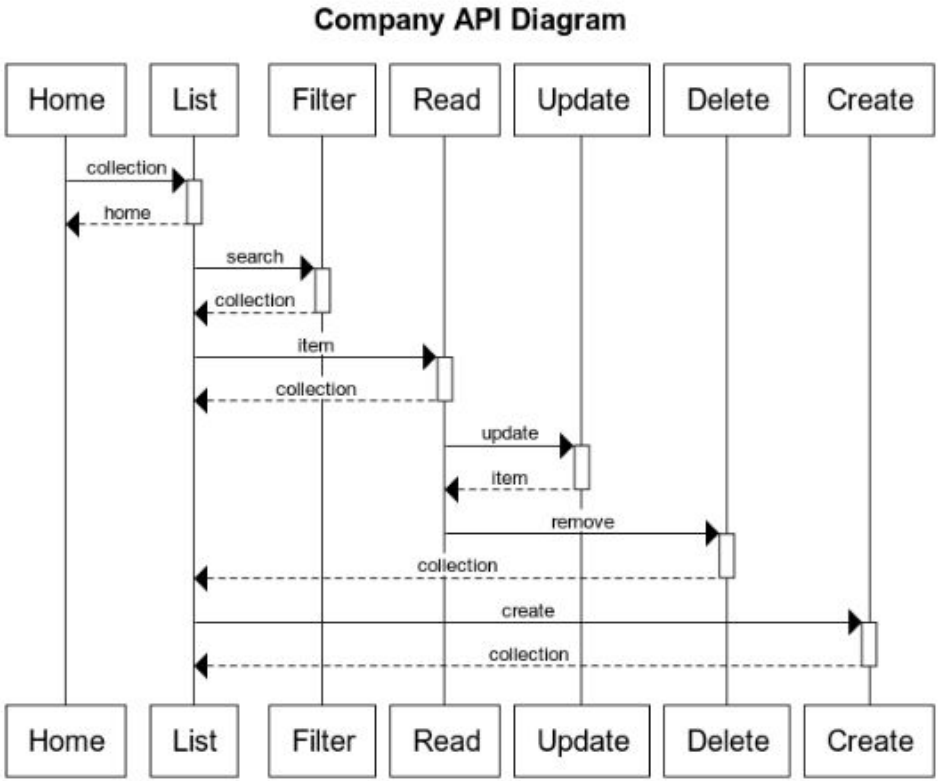
Company Vocabulary

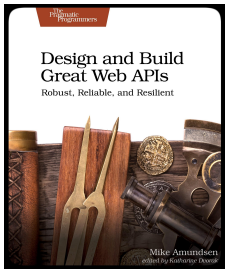
Data Elements

- companyId -> identifier = https://schema.org/identifier
- companyName -> legalName = https://schema.org/legalName
- streetAddress -> streetAddress = https://schema.org/streetAddress
- city -> locality = https://schema.org/addressLocality
- stateProvince -> addressRegion = https://schema.org/addressRegion
- postalCode -> postalCode = https://schema.org/postalCode
- country -> addressCountry = https://schema.org/addressCountry
- telephone -> telephone = https://schema.org/telephone
- email -> email = https://schema.org/email
- status (suspended, active, pending,closed) -> status = https://schema.org/status
- dateCreated -> dateCreated = https://schema.org/dateCreated
- dateUpdated -> dateMOdified = https://schema.org/dateModified

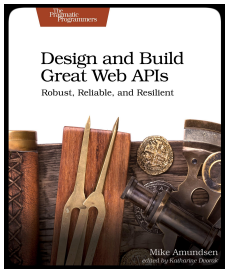


Overnight Assignment : API Diagram



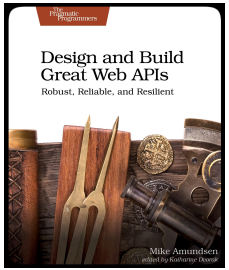


Assignment:Stand-Up

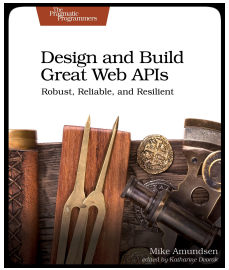


API Design Method

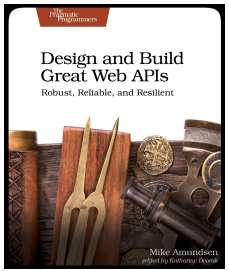
- Story
  - Every API starts with a story
- Model
  - Models help us focus
- Normalize
  - Create shared understanding across APIs
- Diagram
  - Make your solutionvisible
- Describe
  - Produce a machine-readable description



BREAK

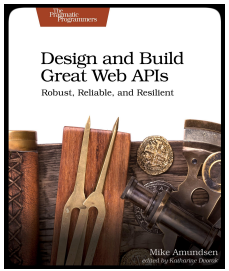


Describing APIs with ALPS



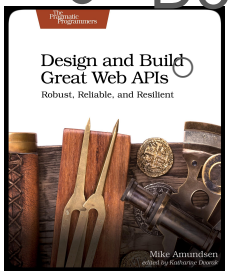
Describe the API

- Design phase is not implementation phase
- API definitions are varied
  - WSDL, WADL, OpenAPI, AsyncAPI, protobuf, SDL
- Use an implementation-agnostic detailed description



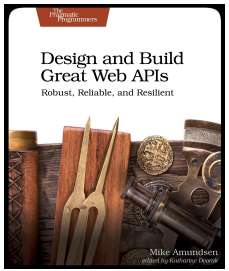
Describe the API : ALPS

- Application-Level Profile Semantics
  - Amundsen-Richardson-Foster (2011)
- Identifies all interface properties
  - Id, familyName, givenName, telephone, etc.
- Identifies all interface actions
  - saveCompany, setStatus, approvePayroll, etc.
- Does not include implementation details
  - URLs, schemas, methods, response codes, etc.



Describe the API : ALPS

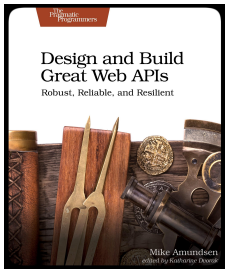
```
alps:
  version: '1.0'
  description: ALPS document for BigCo Credit Check API
```



Describe the API : ALPS

```
alps:
  version: '1.0'
  descri                                    gCo Credit Check API
```

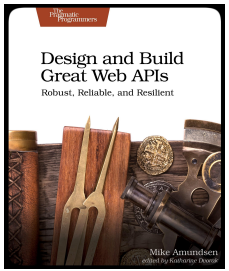
```
  - id: creditCheckItem
    type: safe
    returns: '#ratingItem'
    descriptors:
      - href: '#id'

  - id: creditCheckForm
    type: unsafe
    returns: '#ratingItem'
    descriptors:
      - href: '#id'
      - href: '#companyName'
      - href: '#ratingValue'
```



Describe the API : ALPS

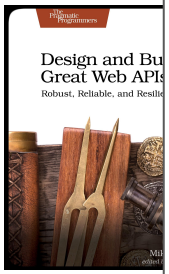
```
alps:
  version: '1.0'
  descri
    - id: creditCheckItem
      type: safe
      returns: '#ratingItem'
      descriptors:
        - href: '#id'

    - id: creditCheckForm
      type: unsafe
      returns: '#ratingItem'
      descriptors:
        - href: '#id'
        - href: '#companyName'
        - href: '#ratingValue'
```

```
  - id: ratingItem
    type: semantic
    descriptors:
      - id: id
        type: semantic
      - id: companyName
        type: semantic
      - id: ratingValue
        type: semantic
      - id: dateCreated
        type: semantic
      - id: dateUpdated
        type: semantic
```



Describe the API : ALPS

- ALPS identifies
  - The state to pass in each message
  - The safety & idempotence of each action
- ALPS is part of Pivotal's Spring frameworks



Preface

Project Metadata

1. Dependencies

Reference Documentation

2. Introduction

3. Getting started

4. Repository resources

5. Paging and Sorting

6. Domain Object Representations (Object Mapping)

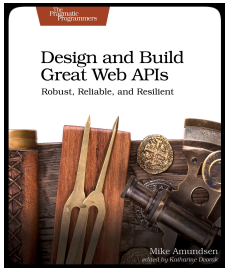
12.1. Application-Level Profile Semantics (ALPS)

> ALPS is a data format for defining simple descriptions of application-level microformats. An ALPS document can be used as a profile to explain the a application-agnostic media type (such as HTML, HAL, Collection+JSON, Sir documents across media types.
>
> *— M. Admundsen / L. Richardson / M. Foster*
>
> *https://tools.ietf.org/html/draft-amundsen-richardson-foster-alps-00*

Spring Data REST provides an ALPS document for every exported repository. It co transitions and the attributes of each repository.

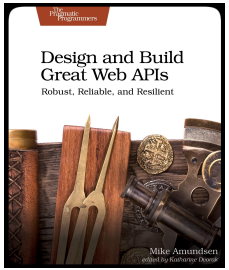
Describe the API : ALPS

- Publish the profile
- Check it into the project repository
- Use it as a guide going forward

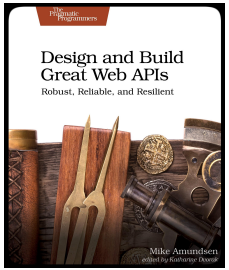



```yaml
alps:
  version: '1.0'
  description: ALPS document for BigCo Onboarding API

  descriptors:
    - id: home
      type: safe
      returns: '#onboarding'

```

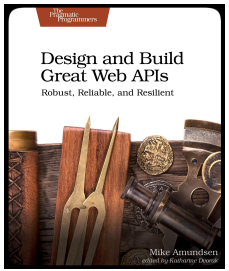
Let's Discuss!



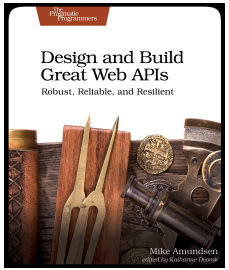
BREAK



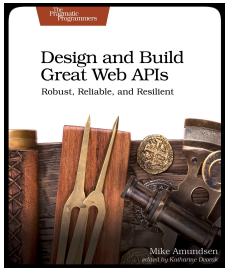
Exercise: Describing your API



Exercise: Stand-Up

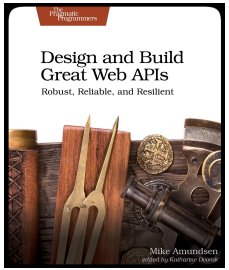


BREAK

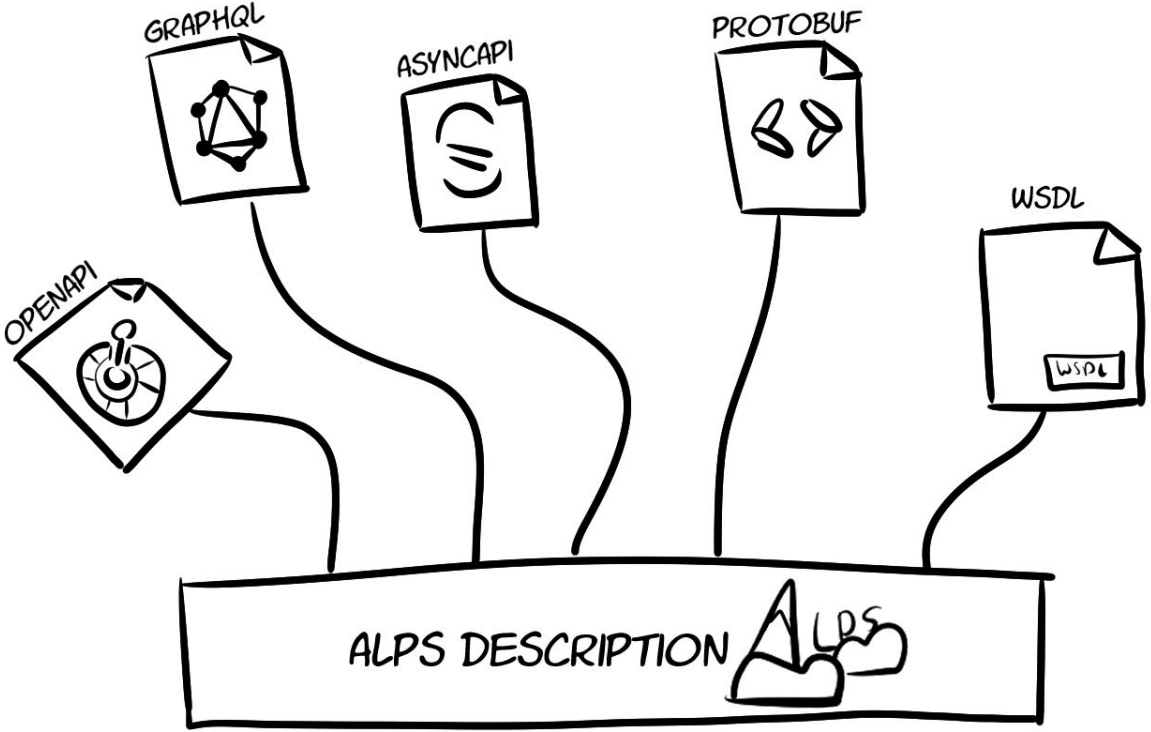


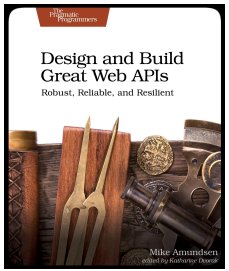
A Method for Unified API Design

- Break the HTTP-centric grip on your API design process
- Embrace interface descriptions (ALPS)
- Enable translations (OpenAPI, AsyncAPI, SDL, proto, etc.)
- Future-proof your design process

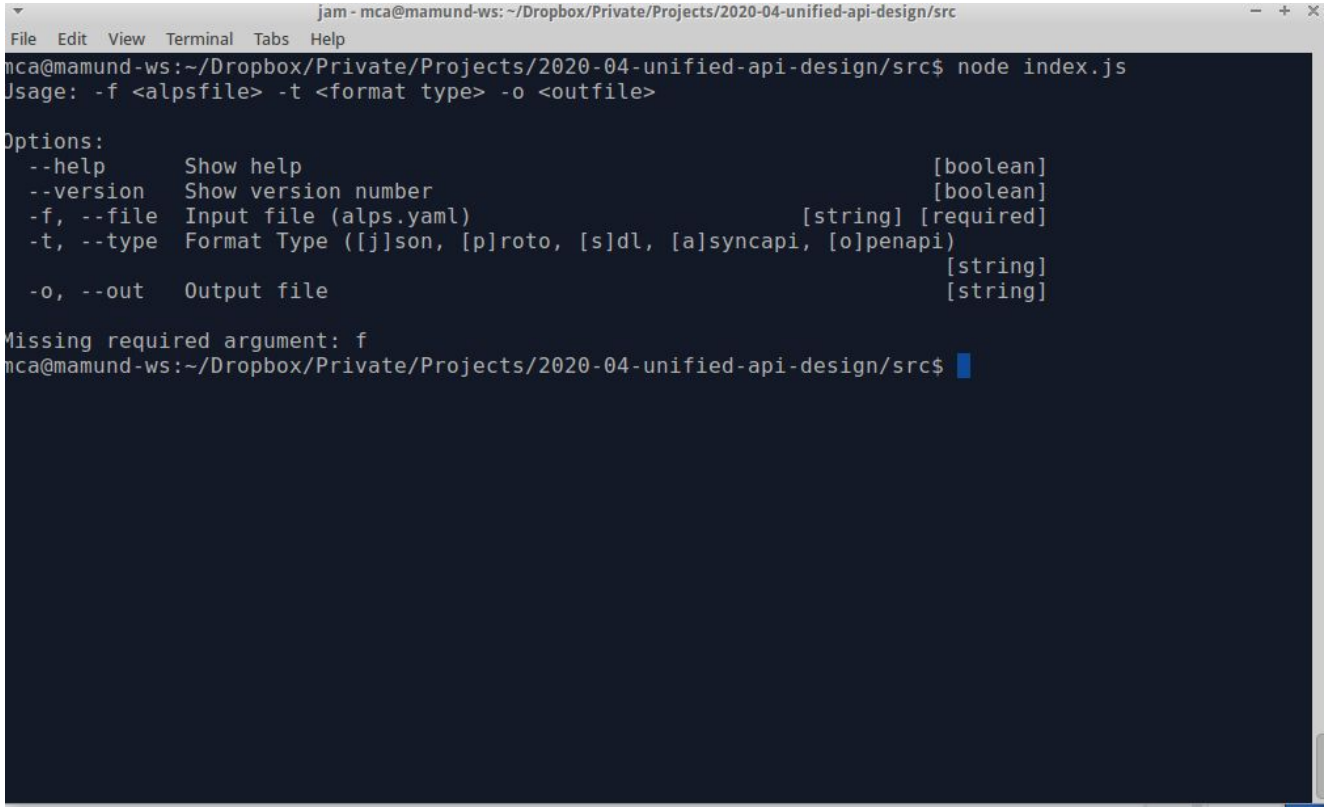


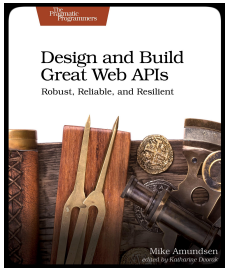
A Method for Unified API Design



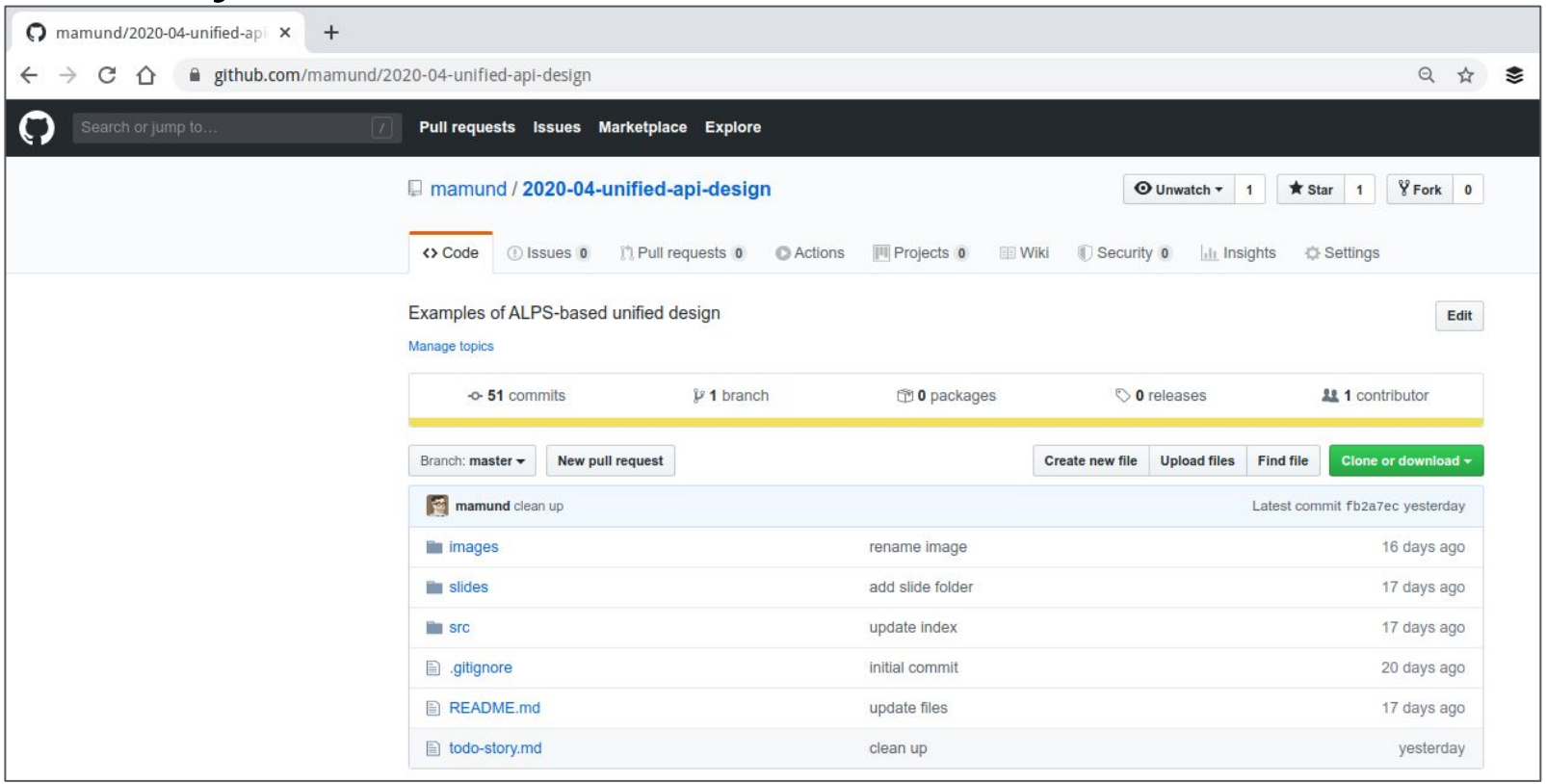


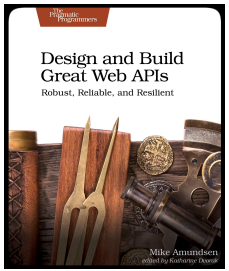
Unified Utility for ALPS Conversion



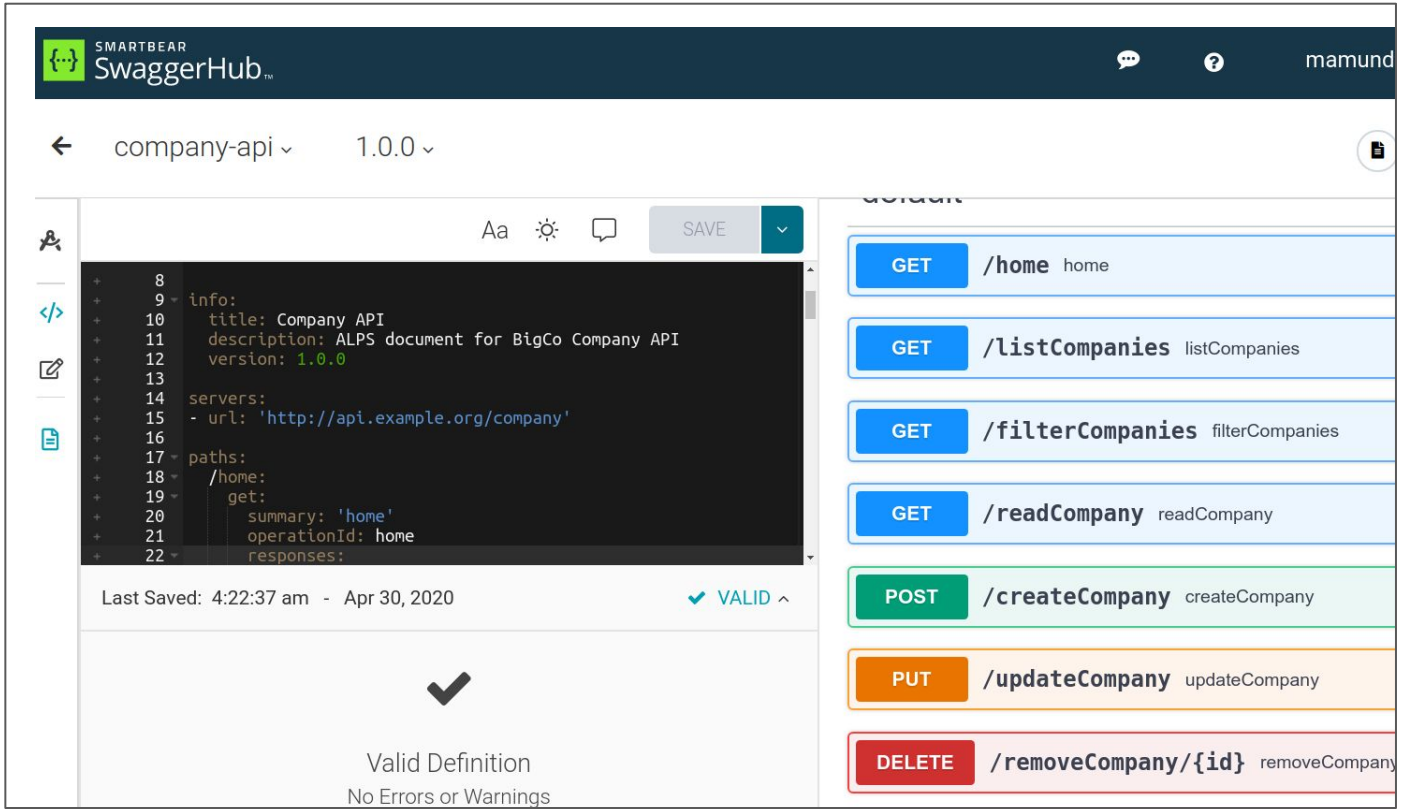


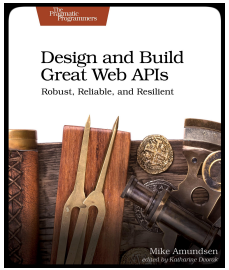
Unified Utility for ALPS Conversion



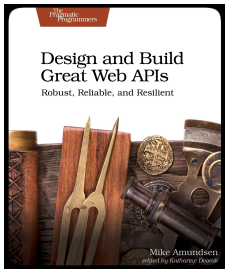


Company OpenAPI



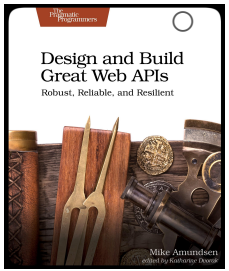


API Design: The Big Picture



Write out your API Story

- APIs start with a story
  - "We need…"
  - "Our customers requested…"
  - "I have an idea…"
- Stories are shared understanding
  - Our brains are wired for stories, not data
  - Stories are accessible
  - Stories are repeatable



Write out your API Story

13 lines (9 sloc) | 1.09 KB — Raw | Blame | History

Company Story at BigCo, Inc.

Purpose

We keep track of companies for BigCo, Inc.

Data

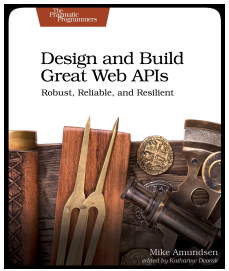
Data we include in a company record includes company name, street address, city, state/province, postal code, country, telephone, and email. Each record has a status value (pending, suspended, active, closed). We also track the date/time the record was created and the last date/time it was updated. We keep copies of the records, even after they have been deleted.

Actions

Typical work on the company records include getting the list of company records, reading a single record, creating, updating, and deleting records. You can also update the status of a single record. Finally, you can get a filtered list of all redords (support for filtering by status, by country, by state, and by company name).

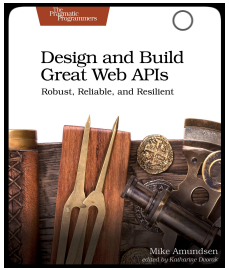
Processing

Each company has a unique identifier in the system. Right now that is a combination of the first four letters of their company name and date the company was added to the system (in the format YYYYMMDD. We add another digit if adding that does not result in a unique identifier in the system.

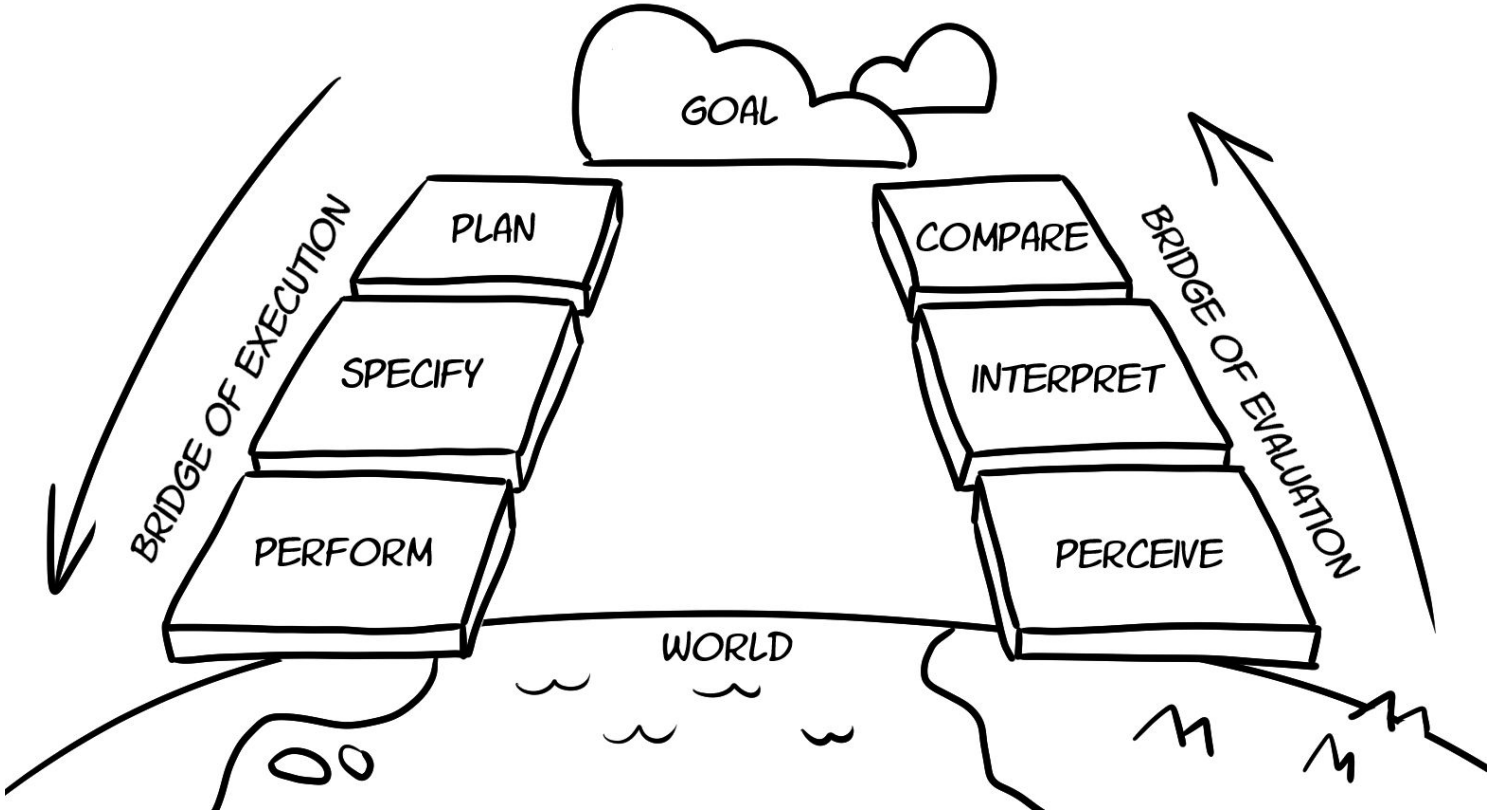


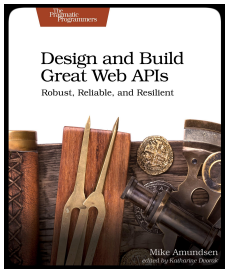
Model Your API Solution

- Models are how we see the world
  - Donald Norman's Lifecycle
  - It's always a circle, not a line
  - The RPW Loop
- Models are how we translate the story
  - Data
  - Actions
  - Workflows

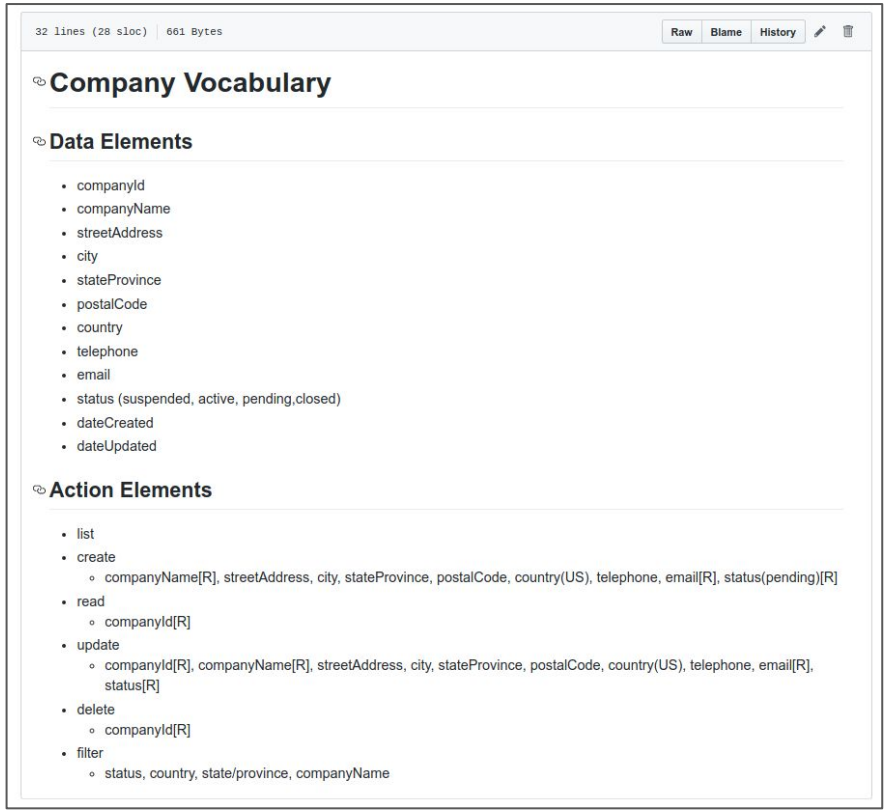


Norman's Lifecycle





Models are how we see the world



32 lines (28 sloc) | 661 Bytes — Raw | Blame | History

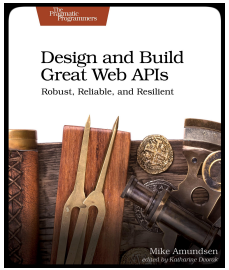
Company Vocabulary

Data Elements

- companyId
- companyName
- streetAddress
- city
- stateProvince
- postalCode
- country
- telephone
- email
- status (suspended, active, pending,closed)
- dateCreated
- dateUpdated

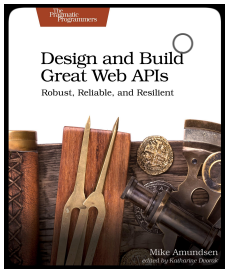
Action Elements

- list
- create
  - companyName[R], streetAddress, city, stateProvince, postalCode, country(US), telephone, email[R], status(pending)[R]
- read
  - companyId[R]
- update
  - companyId[R], companyName[R], streetAddress, city, stateProvince, postalCode, country(US), telephone, email[R], status[R]
- delete
  - companyId[R]
- filter
  - status, country, state/province, companyName

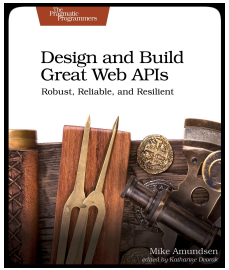


Normalize your API Model

- APIs have their own "story" but share the same "terms"
- API Properties
  - firstname -> givenName
  - lastname -> familyName
  - zipcode -> postalCode
- Shared terms mean shared understanding
  - company.status = account.status
  - user.familyName = customer.familyName



Normalize your API Model



32 lines (28 sloc) | 1.22 KB

Raw Blame History

Company Vocabulary

Data Elements

- companyId -> identifier = https://schema.org/identifier
- companyName -> legalName = https://schema.org/legalName
- streetAddress -> streetAddress = https://schema.org/streetAddress
- city -> locality = https://schema.org/addressLocality
- stateProvince -> addressRegion = https://schema.org/addressRegion
- postalCode -> postalCode = https://schema.org/postalCode
- country -> addressCountry = https://schema.org/addressCountry
- telephone -> telephone = https://schema.org/telephone
- email -> email = https://schema.org/email
- status (suspended, active, pending,closed) -> status = https://schema.org/status
- dateCreated -> dateCreated = https://schema.org/dateCreated
- dateUpdated -> dateMOdified = https://schema.org/dateModified

Diagram your API

- People "think" visually
- Diagrams reveal assumptions
- Diagrams are accessible
- Diagrams are easy to create/change/share

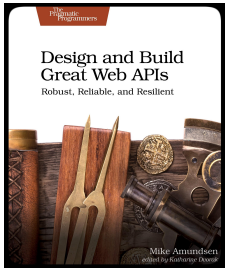
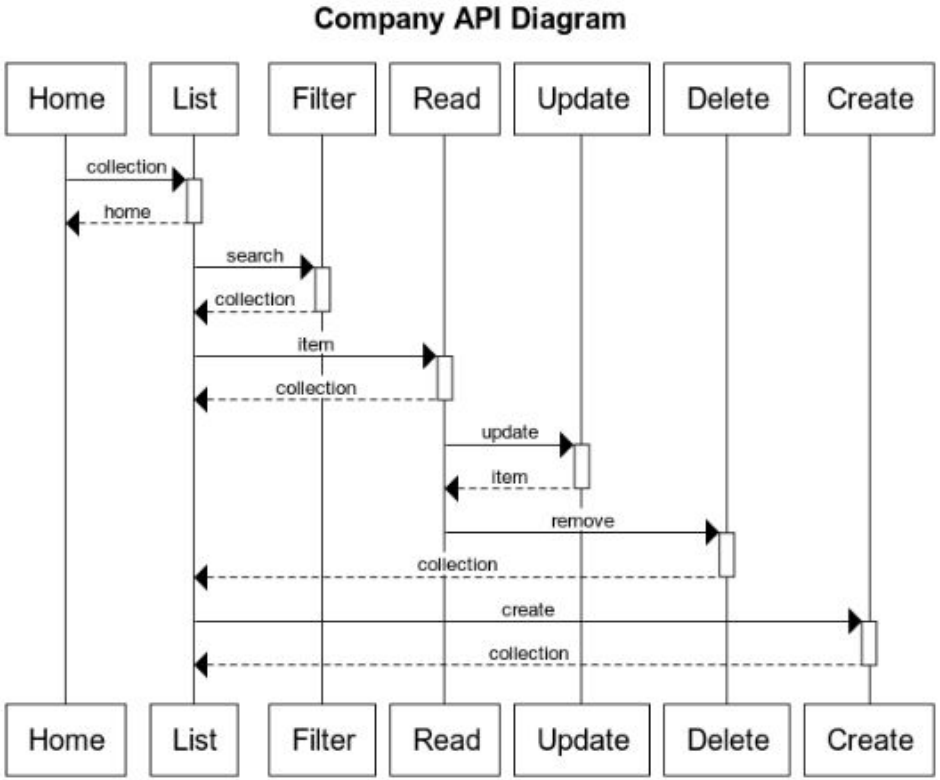
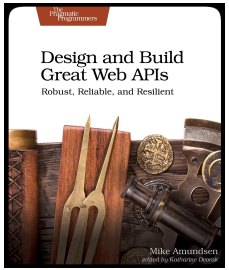


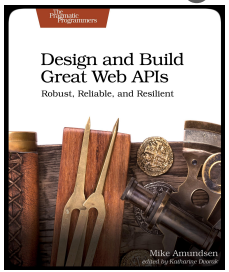
Diagram your API





Describe your API

- Application-Level Profile Semantics
  - Amundsen-Richardson-Foster (2011)
- Identifies all interface properties
  - Id, familyName, givenName, telephone, etc.
- Identifies all interface actions
  - saveCompany, setStatus, approvePayroll, etc.
- Does not include implementation details
  - URLs, schemas, methods, response codes, etc.

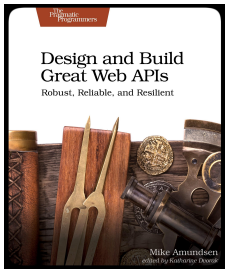


Describe your API

```
alps:
  version: '1.0'
  description: ALPS document for BigCo Credit Chec
```

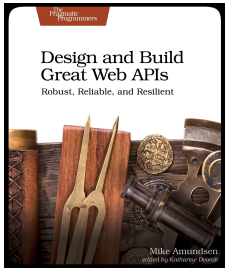
```
  - id: creditCheckItem
    type: safe
    returns: '#ratingItem'
    descriptors:
      - href: '#id'

  - id: creditCheckForm
    type: unsafe
    returns: '#ratingItem'
    descriptors:
      - href: '#id'
      - href: '#companyName'
      - href: '#ratingValue'
```

```
- id: ratingItem
  type: semantic
  descriptors:
    - id: id
      type: semantic
    - id: companyName
      type: semantic
    - id: ratingValue
      type: semantic
    - id: dateCreated
      type: semantic
    - id: dateUpdated
      type: semantic
```

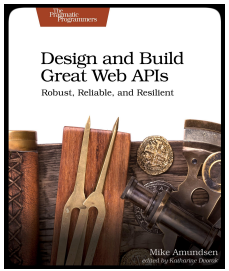


Summing it all up...

- Story
- Model
- Normalize
- Diagram
- Describe



Open Discussion Time



Designing Great APIs
Part Two

@mamund
Mike Amundsen