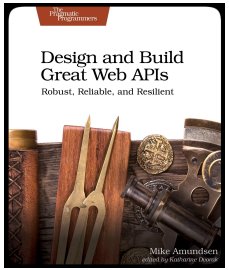
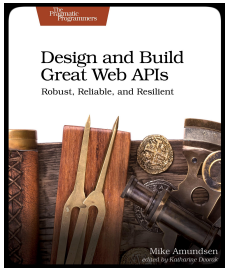
# Building Great Web APIs

## Part Two

@mamund
Mike Amundsen

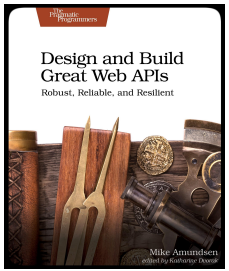


# Welcome Back

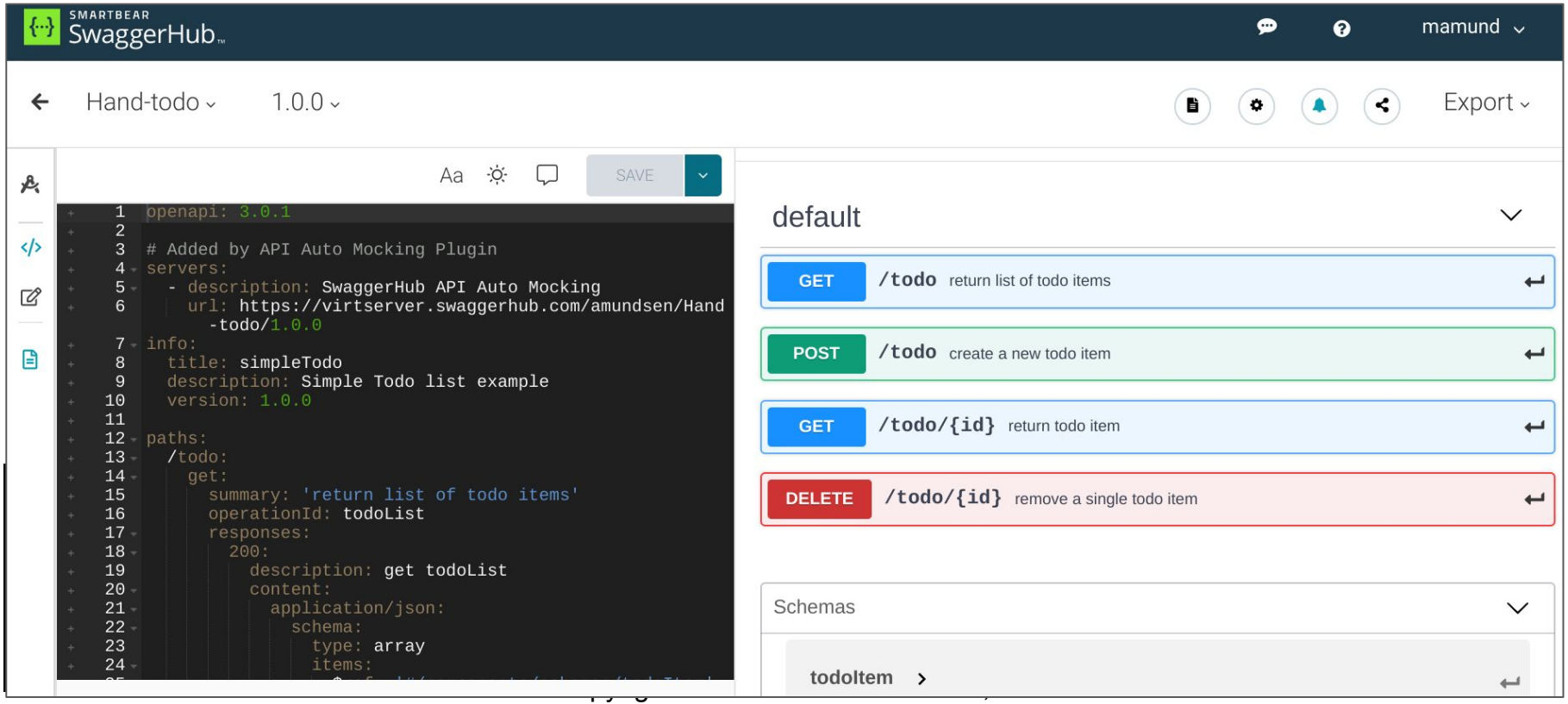


# Three-Phase API Implementation

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

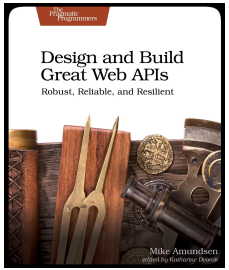


# Prototyping with OpenAPI



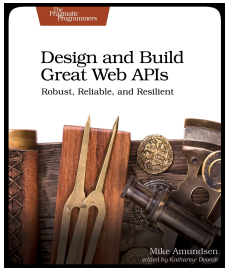
# Building with NodeJS/Express & DARRT

```
mca@penguin:~/projects/api-tool-kit/api-starter-kit/darrt$ tree
.
├── actions.js
├── data.js
├── lib
│   ├── component.js
│   ├── ejs-helpers.js
│   ├── storage.js
│   └── utils.js
├── representation.js
├── representors
│   ├── app-json.js
│   ├── forms-json.js
│   ├── links-json.js
│   ├── prag-json.js
│   └── text-csv.js
├── resources.js
└── transitions.js

2 directories, 14 files
mca@penguin:~/projects/api-tool-kit/api-starter-kit/darrt$
```

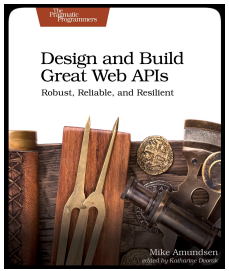


# Overnight Assignment for API Design

- Start from an ALPS Description
- Using the API Starter Kit…
  - Update `data.js`, `actions.js`, & `resources.js` as needed
- Use `npm run dev` to validate the API Project

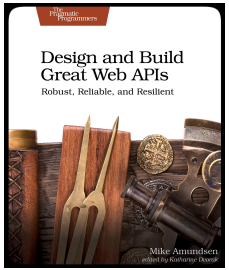


# Assignment: Review



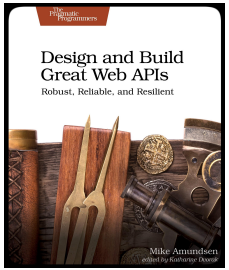
# ALPS API Description

```
#########################################################
# TODO .....: Application-Level Profile Semantics doc
# Author ...: Mike Amundsen (@mamund)
# Date .....: 2020-04-13
#########################################################

alps:
  version: '1.0'
  doc:
    value: ALPS document for BigCo Company API

  # metadata
  name: Company API
  id: http://alps.io/profiles/mamund/company
  root: http://api.example.org/company

  descriptor:
    # properties
    - id: id
      type: semantic

    - id: status
      type: semantic
      text: 'suspended, pending, active, closed'
```



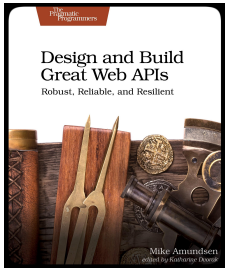
# DARRT : data.js

```
// this service's message properties
exports.props = [
  'id',
  'companyName',
  'streetAddress',
  'city',
  'stateProvince',
  'postalCode',
  'country',
  'telephone',
  'email',
  'status',
  'dateCreated',
  'dateUpdated'
];

// required properties
exports.reqd = ['id','companyName','email','status'];

// enumerated properties
exports.enums = [
  {status:['pending','active','suspended','closed']}
];
```



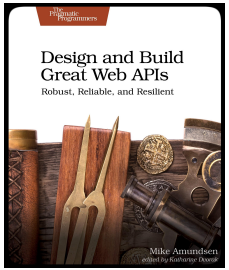
# DARRT : actions.js

```
/*****************************************
// bigco, inc
// company action elements
// 2020-02-01 : mamund
 *****************************************/

var component = require('./lib/component');
var data = require('./data');

/*****************************************
// actions for the company service
// home, create, list, filter,
// read, update, status, remove
 *****************************************/

module.exports.home = function(req,res) {}
module.exports.create = function(req,res) {}
module.exports.list = function(req,res) {}
module.exports.filter = function(req,res) {}
module.exports.read = function(req,res) {}
module.exports.update = function(req,res) {}
module.exports.status = function(req,res) {}
module.exports.remove = function(req,res) {}

```

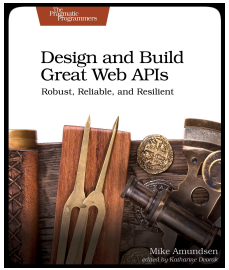


# DARRT : resources.js

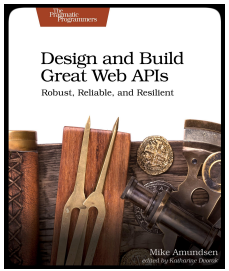
```
/*****************************************
// bigco, inc
// company resources
 *****************************************/

var express, router, bodyParser, actions, representation,
  transitions, utils, templates, forms, metadata;

init();

// shared metadata for this service
metadata = [
  {name: "title", value: "BigCo Company Records"},
  {name: "author", value: "Mike Amundsen"},
  {name: "release", value: "1.0.0"}
];

router.get('/',function(req,res){...});
router.post('/', function(req,res){...});
router.get('/list/',function(req,res){...});
router.get('/filter/', function(req,res){...});
router.get('/:id', function(req,res){...});
router.put('/:id', function(req,res){...});
router.delete('/:id', function(req,res){...});
router.patch('/status/:id', function(req,res){...});
```



# BREAK

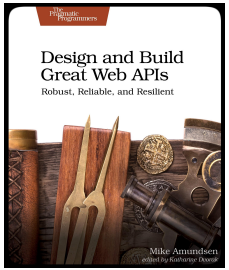


# Validating the API



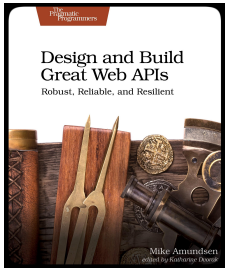
# Validating the API

- Compose simple "curl tests" to validate the API
- Test each endpoint
- Confirm "happy path" tests (200 OK)
- Include "sad path" tests (400 Bad Request)

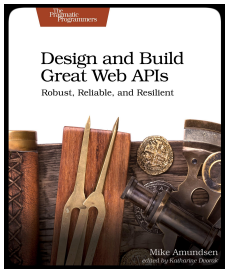


# Validating the API

```
# company SRTs
# 2020-03 mamund

# happy
# these should return 200
http://localhost:8484/
http://localhost:8484/list/
http://localhost:8484/filter?status=active
http://localhost:8484/ -X POST -d id=q1w2e3r4&status=pending&companyName=MikeCo&er
http://localhost:8484/q1w2e3r4 -X PUT -d streetAddress=123%
20Main&city=Byteville&stateProvince=MD&postalCode=12345&country=USA&telephone=2345
http://localhost:8484/status/q1w2e3r4 -X PATCH -d status=active
http://localhost:8484/q1w2e3r4 -X DELETE

# sad
# these should return 400
http://localhost:8484/xxxx -X GET
http://localhost:8484/12345 -X DELETE
http://localhost:8484/ -X POST -d id=12345
http://localhost:8484/ -X POST -d id=12345&companyName=BadRec,%20Inc.
http://localhost:8484/ -X POST -d id=12345&companyName=BadRec,%20Inc.&email=badrec
http://localhost:8484/ -X POST -d id=12345&companyName=BadRec,%
20Inc.&email=badrec@example.org&status=broken
http://localhost:8484/ -X POST -d id=12345&companyName=BadRec,%
20Inc.&email=badrec@example.org&status=pending
http://localhost:8484/ -X POST -d id=12345&companyName=BadRec,%
20Inc.&email=badrec@example.org&status=pending
```

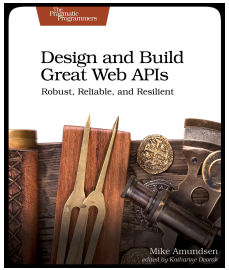


# Exercise: Write Validation Calls

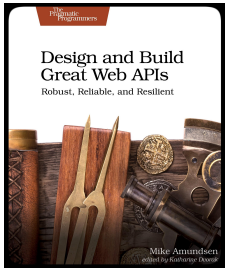


# Writing Validating Calls

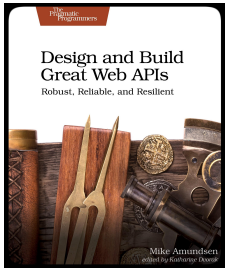
- Use curl
- Write out a 'happy path' call for each endpoint
- Write out a 'sad path' call to confirm data.js rules



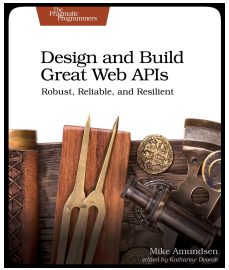
# Exercise: Stand-Up



# BREAK

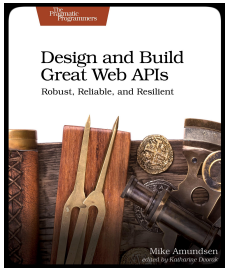


# Deploying APIs



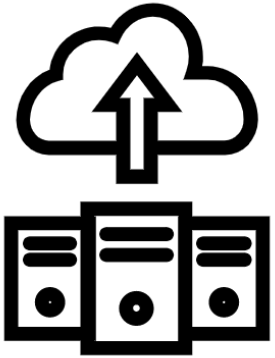
# Deploying APIs

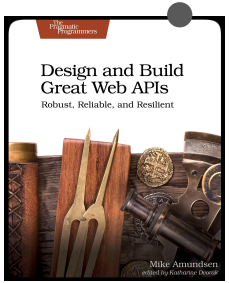
- Challenges of Deployment
- Git-based Deployment
- Using **Heroku**



# Deploying APIs - Challenges

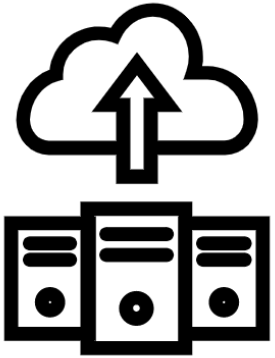
- Deploying your app can be complicated
- Compatibility
  - Hardware
  - OS
  - Platform
  - Framework
  - Dependencies

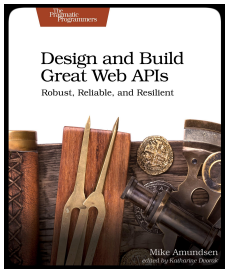




# Deploying APIs - DevOps

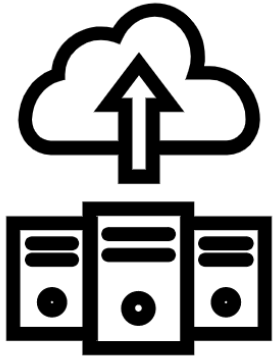
- DevOps was created to help with all this
- Developers & Operators working together
- Started as a hashtag on twitter #DevOps
- Series of small conferences started in 2009
- Emphasis on automation to improve reliability

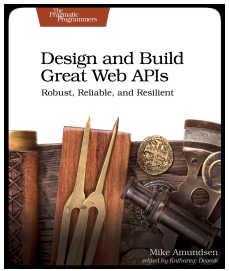




# Deploying APIs - Tools

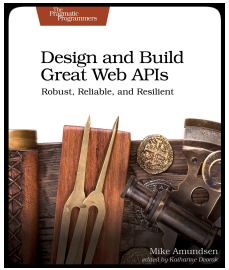
- Build tools
- CI/CD pipeline
- Docker (containers)
- Kubernetes (deployment orchestration)





# Deploying APIs - Using Heroku

- Cloud platform (2007)
- Originally just for Ruby/Rails projects
- Now supports Java, NodeJS, Python, Go, Clojure, Scala
- Acquired by Salesforce in 2011
- Full platform w/ marketplace ecosystem
- Heroku uses proprietary container tech (Dynos)

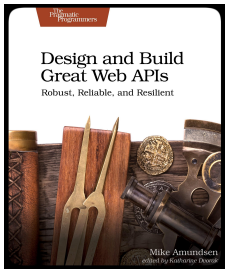


# Deploying APIs - Using Heroku

- Download CLI
  https://devcenter.heroku.com/articles/heroku-cli

- Documentation
  https://devcenter.heroku.com/articles/using-the-cli

- Create an Account (required)
  https://signup.heroku.com/



# Deploying APIs - Using Heroku

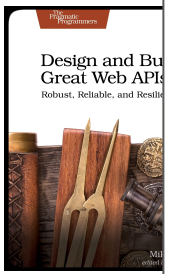
- Git deploy tutorial
  https://devcenter.heroku.com/articles/git

The `heroku create` CLI command creates a new empty application on Heroku, along with an associated empty Git repository. If you run this command from your app's root directory, the empty Heroku Git repository is automatically set as a remote for your local repository.

```
$ heroku create
Creating app... done, ⬢ thawing-inlet-61413
https://thawing-inlet-61413.herokuapp.com/ | https://git.heroku.com/thawing-inlet-61
```

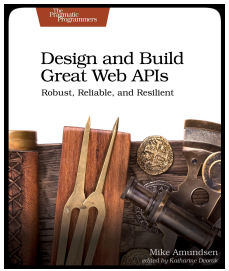
You can use the `git remote` command to confirm that a remote named `heroku` has been set for your app:

```
$ git remote -v
heroku  https://git.heroku.com/thawing-inlet-61413.git (fetch)
heroku  https://git.heroku.com/thawing-inlet-61413.git (push)
```



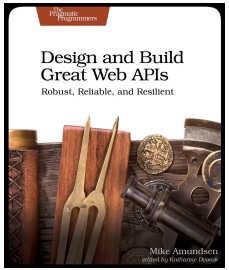

HEROKU

# Releasing Exercise

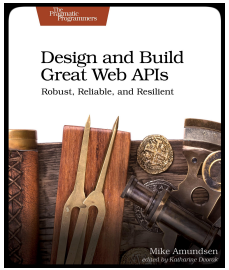


# Releasing Exercise

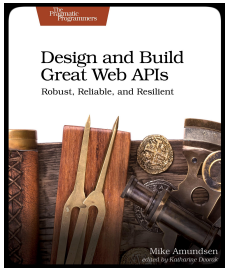
- Open command window in your project
- `heroku login`
- `heroku create`
- `git push heroku master`



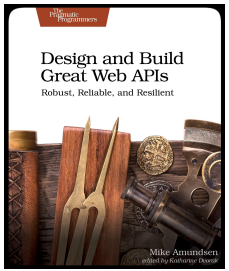
# Exercise: Review



# BREAK

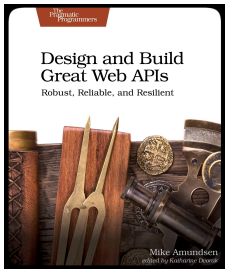


# Building APIs: Summary



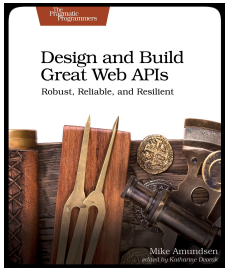
# Three-Phase API Implementation

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



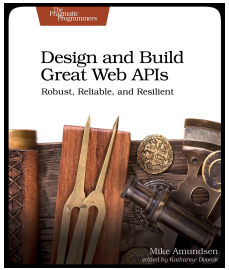
# Building APIs : DARRT

- Simple process for publishing running interfaces
- Data
- Actions
- Resources
- Representations
- Transitions



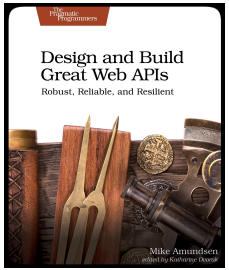
# Building APIs : DARRT

- Simple process for publishing running interfaces
- Data
- Actions
- Resources
- Representations
- Transitions

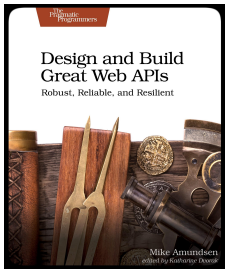


# Deploying APIs

- Challenges of Deployment
- Git-based Deployment
- Using **Heroku**



# Open Question Time



# Building Great Web APIs

## Part Two

@mamund
Mike Amundsen