

# Designing APIs with OpenAPI

Mike Amundsen  
@mamund



**Mike Amundsen**  
**@mamund**

# Designing APIs with OpenAPI

- API Design Assets
- Prototyping Your API Design with OpenAPI
- Building with Open OpenAPI
- Defining OpenAPI Paths
- Modifying Your API in Production



***Let's get started ...***



# API Design Assets

# Design Assets

*Design assets are any digital resources that are created or used in the design process.*

-- Uxcel



<https://app.uxcel.com/glossary/assets>

# API Design Assets

- API Story
- API Diagram
- API Vocabulary



# Person Service API

---

## Purpose

---

The Person Service allows users to keep track of important people.

## Action

---

The Person Service API supports the following actions:

- Create Person
- Get Person List
- Filter Person List
- Remove Person
- Get Person
- Update Person
- Update Person Status
- Remove Person

## Data

---

The service keeps tracks of important persons. For each person there is a unique `id` for each person record stored. Each record also contains values for `givenName`, `familyName`, `telephone`, and `email`. Each record also has an internal `status` value.

## Rules

---

- When a new record is created the client application can establish the value for `id` or allow the service to assign one. If the client attempts to apply an `id` value that is already in use, the new service will reject the record.
- The only valid values for `status` are `active` and `inactive`. Any other value will be rejected by the service.
- The following fields are required when storing updated or new records: `id`, `givenName`, `familyName`, `status`

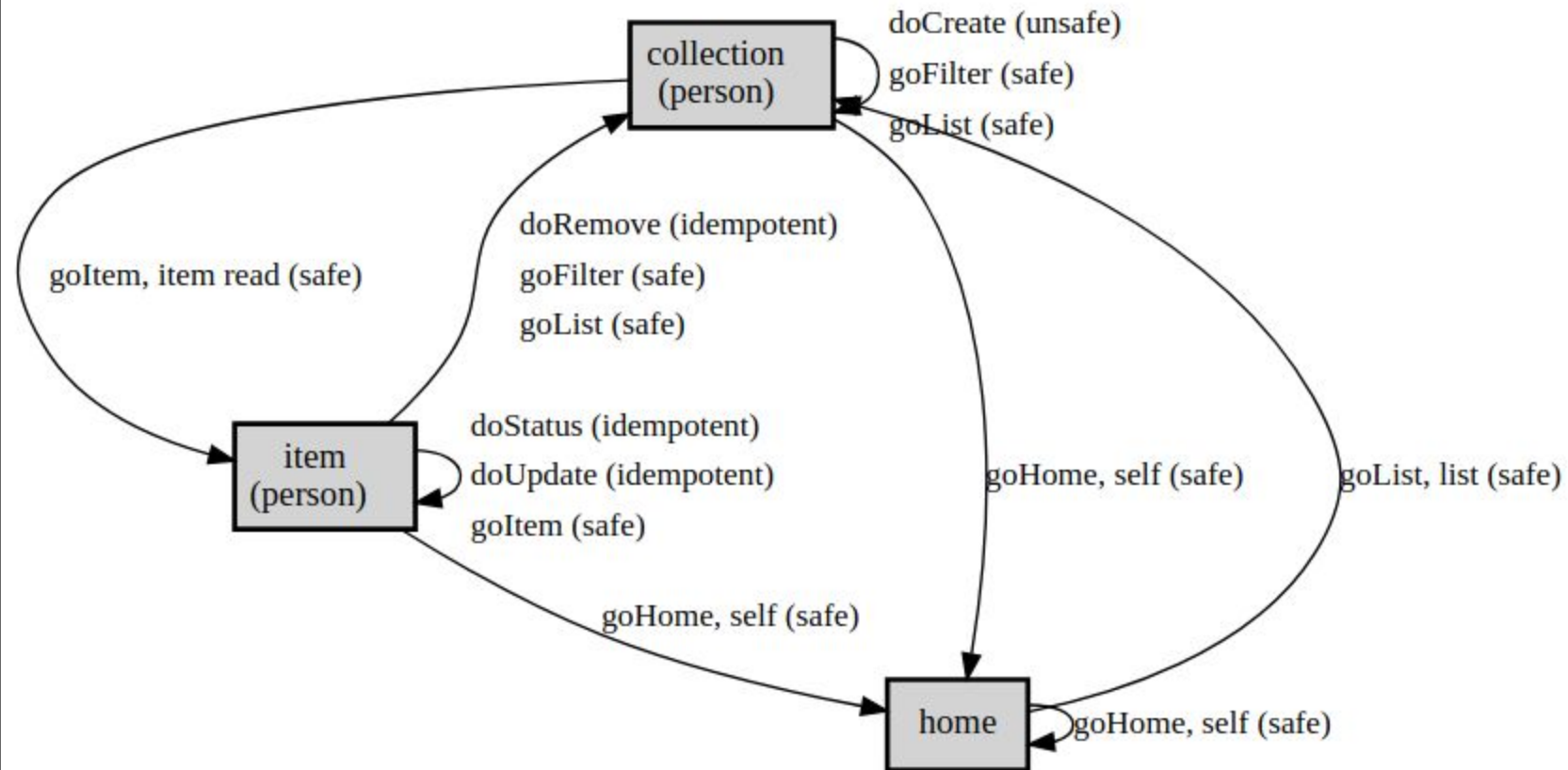
## Processing

---

*NONE*



## Person Service API



# API Person Service Vocabulary

## Resources

- home (semantic), Home (starting point) of the person service
- collection (semantic), List of person records
- item (semantic), Single person record

## Actions

- doCreate (unsafe), Create a new person record
- doRemove (idempotent), Remove an existing person record
- doStatus (idempotent), Change the status of an existing person record
- doUpdate (idempotent), Update an existing person record
- goFilter (safe), Filter the list of person records
- goHome (safe), Go to the Home resource
- goItem (safe), Go to a single person record
- goList (safe), Go to the list of person records

## Containers

- person (semantic), The properties of a person record

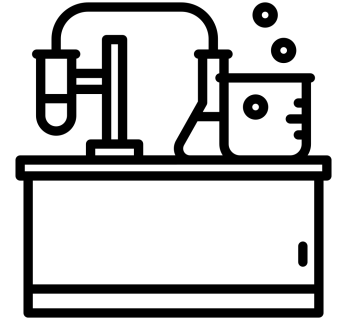
## Properties

- id (semantic), Id of the person record
- familyName (semantic), The family name of the person
- givenName (semantic), The given name of the person
- email (semantic), Email address associated with the person
- telephone (semantic), Telephone associated with the person
- status (semantic), Status of the person record (active, inactive)

# API Design Assets : Review

- API Story
  - Shared, accessible, repeatable
- API Diagram
  - Display the interactions
- API Vocabulary
  - The "magical words" of the API



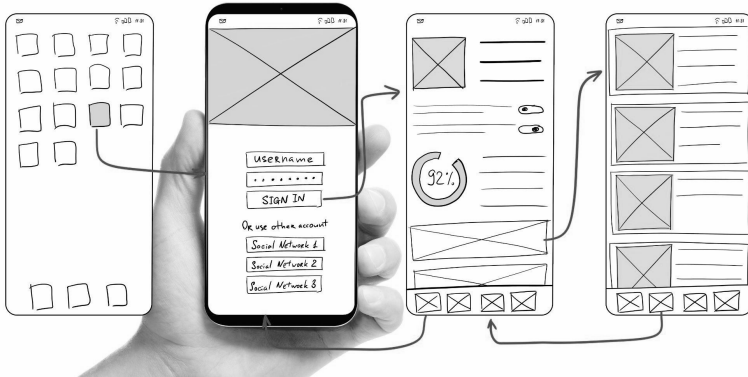
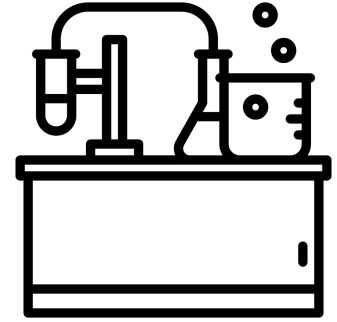


# Prototyping Your API Design

# What is an API Prototype?

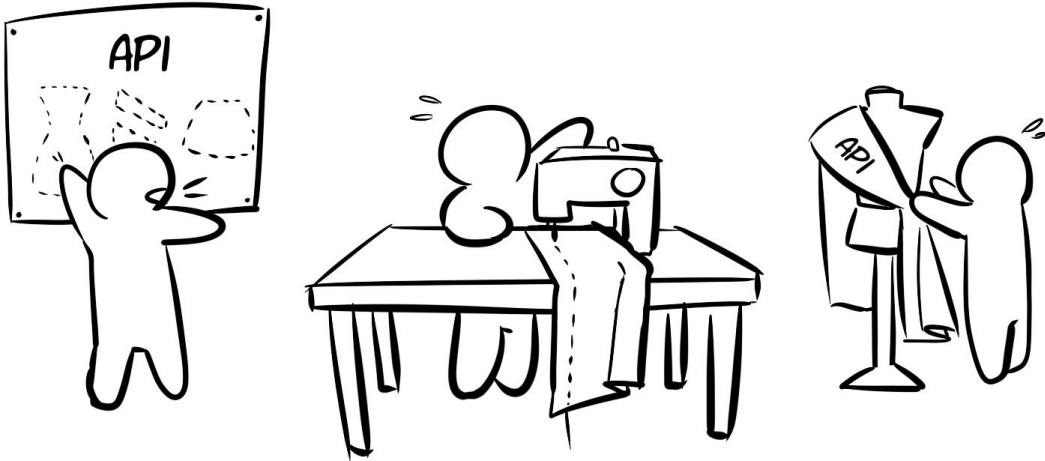
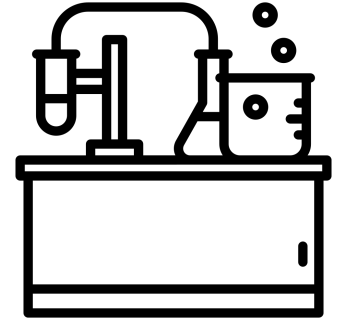
*A first, typical or preliminary model of something, especially a machine, from which other forms are developed or copied.*

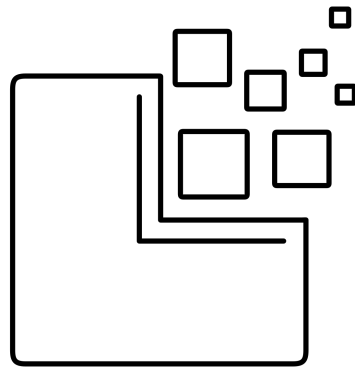
*-- Oxford Dictionary*



# Why use prototypes?

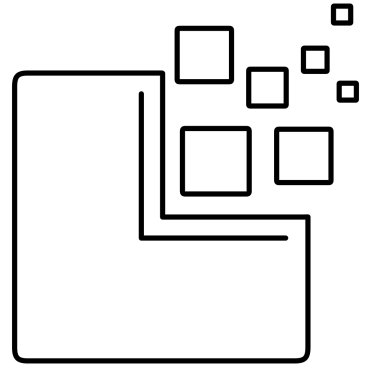
- Inexpensive experiments
- Explore the details
- Prototypes are made to be tested





# Building with OpenAPI

# OpenAPI Basic Info



- Contains the basic shared information about your API
- Key elements to include are:
  - version
  - title
  - description
- Other elements:
  - servers
  - paths
  - components

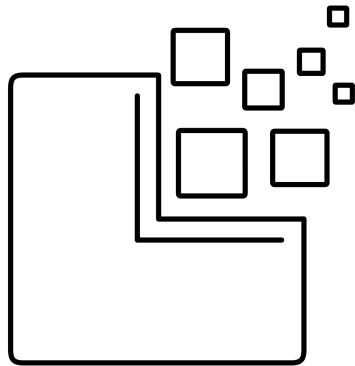
```
1 openapi: 3.0.0
2 info:
3   version: '1.0'
4   title: Person API
5   description: An example API for the "Designing APIs with OpenAPI" workshop
6 servers:
7   # Added by API Auto Mocking Plugin
8   - description: SwaggerHub API Auto Mocking
9     url: https://virtserver.swaggerhub.com/amundsen/Person-API/1.0
10
11 paths: {}
12 components: {}
```



# The OpenAPI Component Parameters

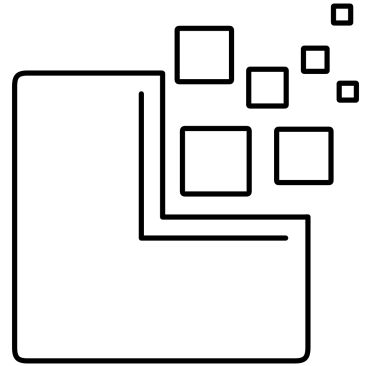
- Important properties:
  - name, in, required, schema/type
- Recommend properties:
  - Description
  - example

```
parameters:  
  nameParam:  
    name: name  
    in: query  
    description: "Used to filter the list of persons"  
    required: false  
    schema:  
      type: string  
      example: "fred"
```



# The OpenAPI Component Schemas

- Important properties:
  - Type (object/array)
  - Properties (type, description, example)

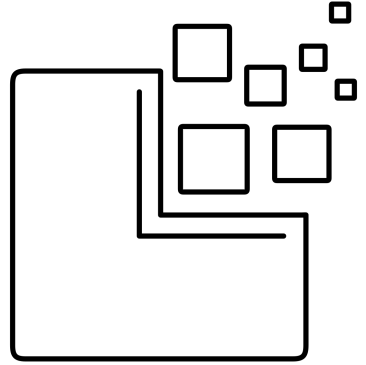


```
PersonItem:
  type: object
  properties:
    id:
      type: string
      description: "Unique identifier for each record"
      example: "q1w2e3r4"
    name:
      type: string
      description: "Name of the person"
      example: "Morton Muffley"
    email:
      type: string
      description: "Default email for this person"
      example: "morton.muffley@example.org"
PersonCollection:
  type: array
  items:
    $ref: '#/components/schemas/PersonItem'
```

# The OpenAPI Component Responses

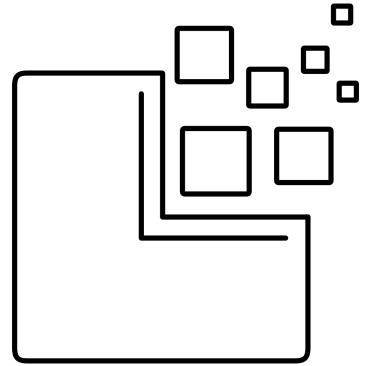
- Important properties:
  - Content (media type)
  - Schema (use \$ref)
- Also:
  - Include description

```
responses:  
  GenericError:  
    description: "generic error occurred"  
    content:  
      application/json:  
        schema:  
          $ref: "#/components/schemas/Error"
```



# The OpenAPI Component RequestBodies

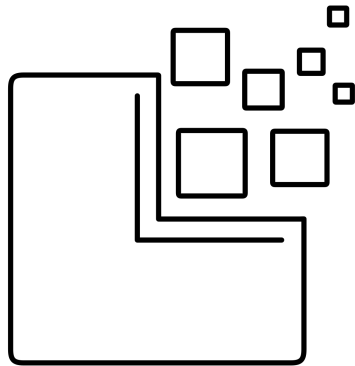
- Use to support arguments in POST/PUT requests
- Important properties:
  - Description, required, content, schema
- Note: you can support more than one content element

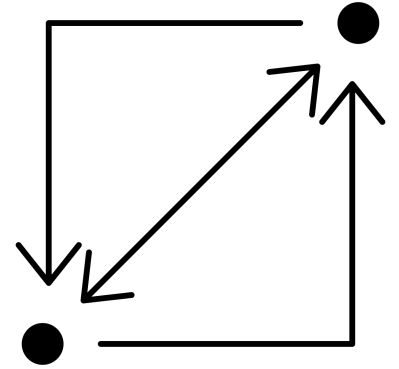


```
requestBodies:  
  AddPerson:  
    description: "body for a new Person record"  
    required: true  
    content:  
      application/json:  
        schema:  
          $ref: "#/components/schemas/PersonItem"  
      application/x-www-urlencoded:  
        schema:  
          $ref: "#/components/schemas/PersonItem"
```

# Leveraging OpenAPI Components

- The OpenAPI Components Section
  - Organize your OpenAPI definition
  - Improve the consistency of your API design
- Parameters, Schemas, Responses, & RequestBodies
  - Parameters for inline queries
  - Schemas for shared objects (item/collection)
  - Responses for general use responses (e.g. errors)
  - requestBodies for POST/PUT requests (e.g. addItem)

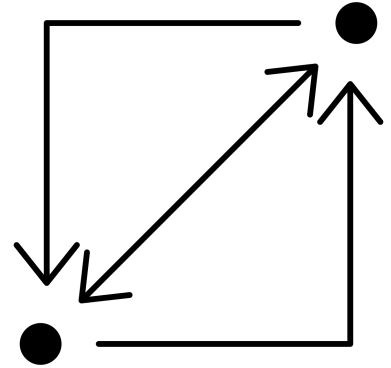




# Defining OpenAPI Paths

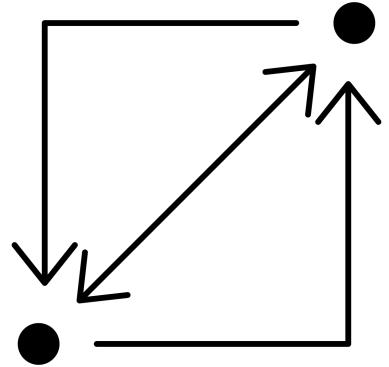
# The OpenAPI Paths Section

- Paths are the 'heart' of OpenAPI documents
- One top-level entry for each resource/URL
- One or more HTTP methods for each resource
- One or more `parameters`, `requestBodies`, `responses`



# Elements of a Path Definition

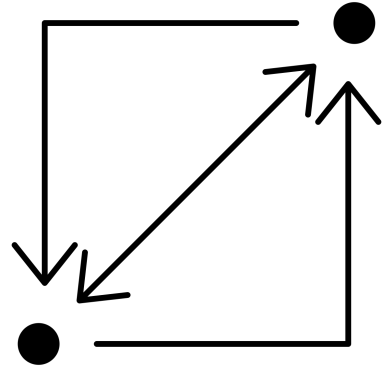
```
### home ###  
/:  
  get:  
    operationId: home  
    summary: Use this to retrieve the home page for the onboarding API  
    tags:  
      - onboarding  
  
    parameters:  
      - $ref: "#/components/parameters/eTag"  
  
    responses:  
      '200':  
        $ref: "#/components/responses/reply"  
  
    default:  
      $ref: "#/components/responses/error"
```





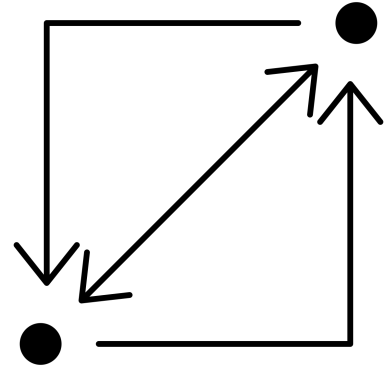
# Elements of a Path Definition

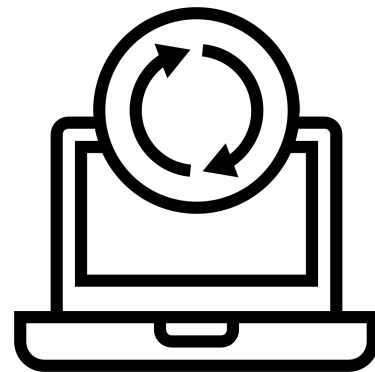
```
### start ###  
/start:  
  post:  
    operationId: startOnboarding  
    summary: Use this to start process of onboarding a customer  
    tags:  
      - onboarding  
  
    parameters:  
      - $ref: "#/components/parameters/ifNoneMatch"  
  
    requestBody:  
      $ref: "#/components/requestBodies/start"  
  
    responses:  
      '201':  
        $ref: "#/components/responses/created"  
  
    default:  
      $ref: "#/components/responses/error"
```



# Elements of a Path Definition

```
### work in progress record ###  
/wip/{identifier}:  
  get:  
    operationId: wipItem  
    summary: Use this to return a single onboarding record  
    tags:  
      - onboarding  
  
    parameters:  
      - $ref: "#/components/parameters/identifier"  
      - $ref: "#/components/parameters/eTag"  
  
    responses:  
      '200':  
        $ref: "#/components/responses/reply"  
  
    default:  
      $ref: "#/components/responses/error"
```

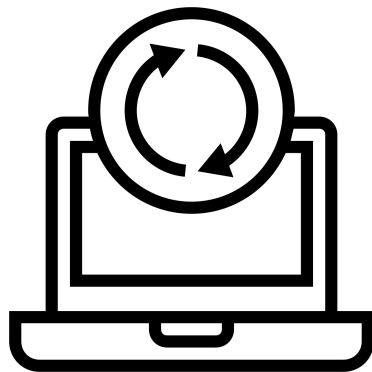




# Modifying Your API in Production

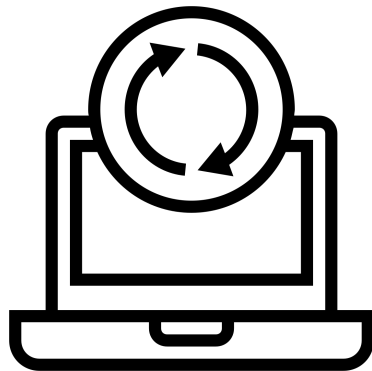
# Modifying Your API in Production

- Beyond Versioning
- Three Rules for Modifying Production APIs
- Testing and Deploying Modified APIs



# Modifying Your API in Production : Review

- Beyond Versioning
  - First, do no harm
- Three Rules for Modifying Production APIs
  - Take nothing away, don't redefine, additions are optional
- Testing and Deploying Modified APIs
  - Don't replace tests, support reversible releases, salesforce vs AWS



# Review

# Designing APIs with OpenAPI : Review

- API Design Assets
- Prototyping Your API Design with OpenAPI
- Leveraging OpenAPI Components
- Defining OpenAPI Paths
- Modifying Your API in Production



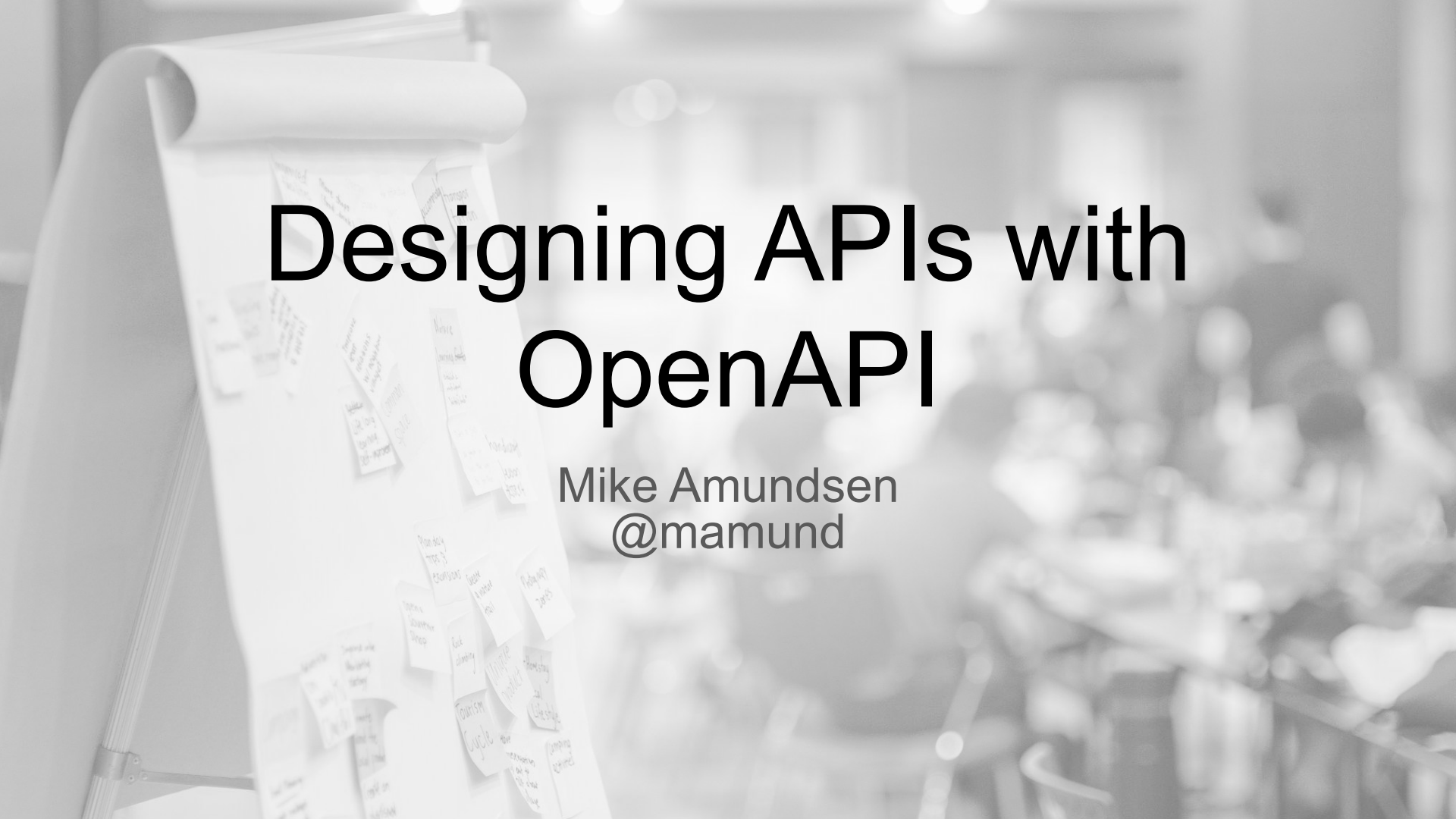
***One more thing ...***



*"Good design is actually a lot harder to notice than poor design, in part because good designs fit our needs so well that the design is invisible,"*

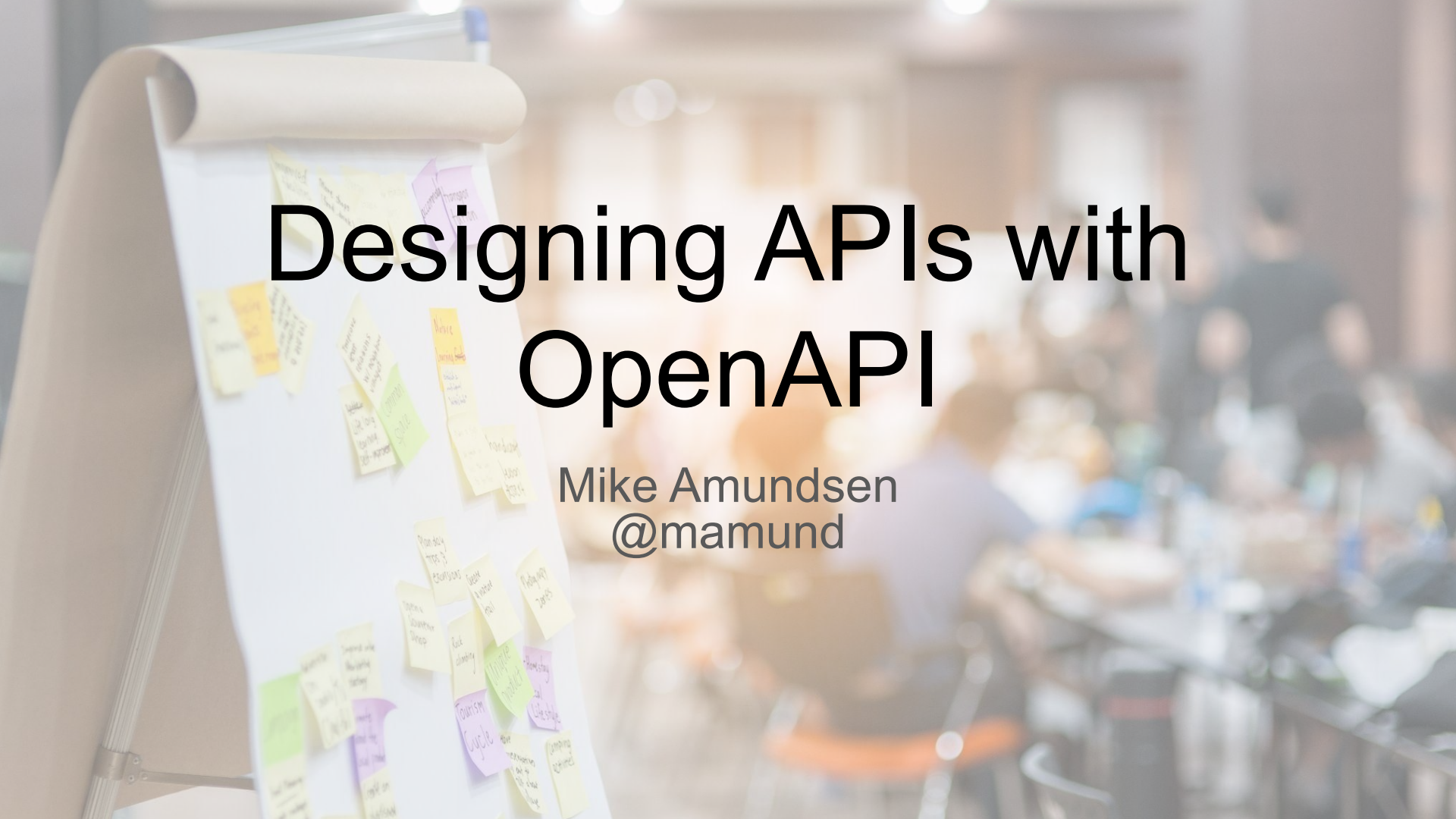
*—Don Norman  
The Design of Everyday Things*





# Designing APIs with OpenAPI

Mike Amundsen  
@mamund



# Designing APIs with OpenAPI

Mike Amundsen  
@mamund