Designing a RESTful API

Shawn Wildermuth
MICROSOFT MVP, INSTRUCTOR AND FILMMAKER

@shawnwildermuth https://wilderminds.com

Agenda

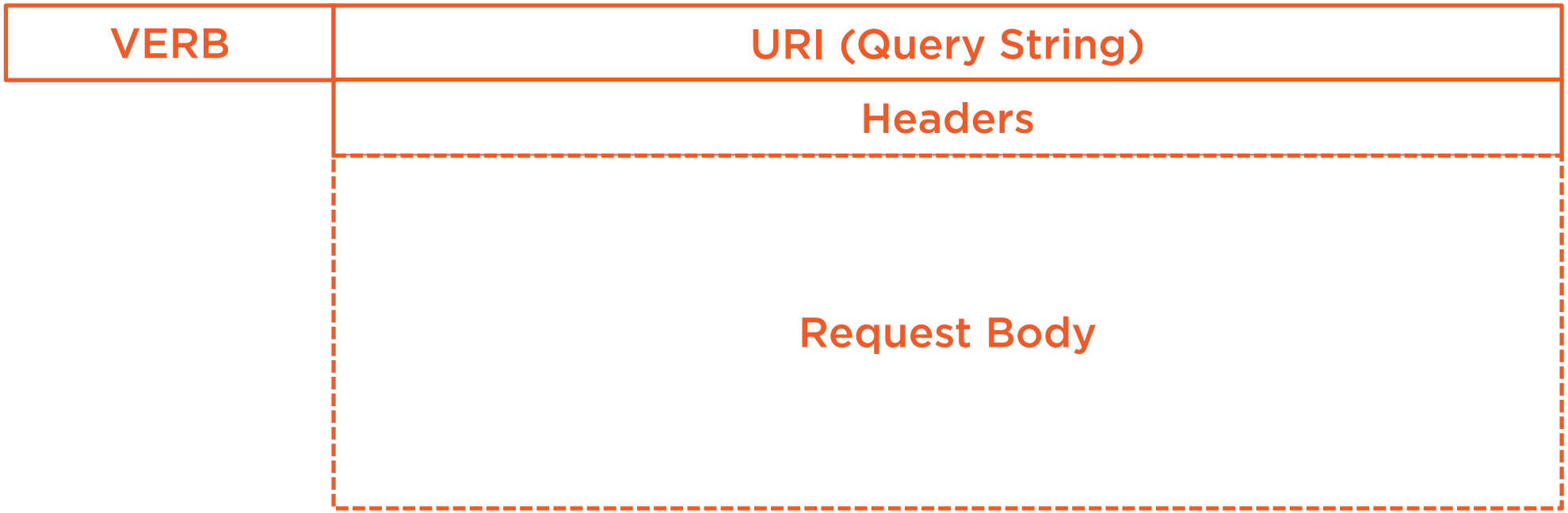
Designing a RESTful API

- Designing the API First
- Nouns and Verbs
- URI Design
- Status Codes
- Designing Results

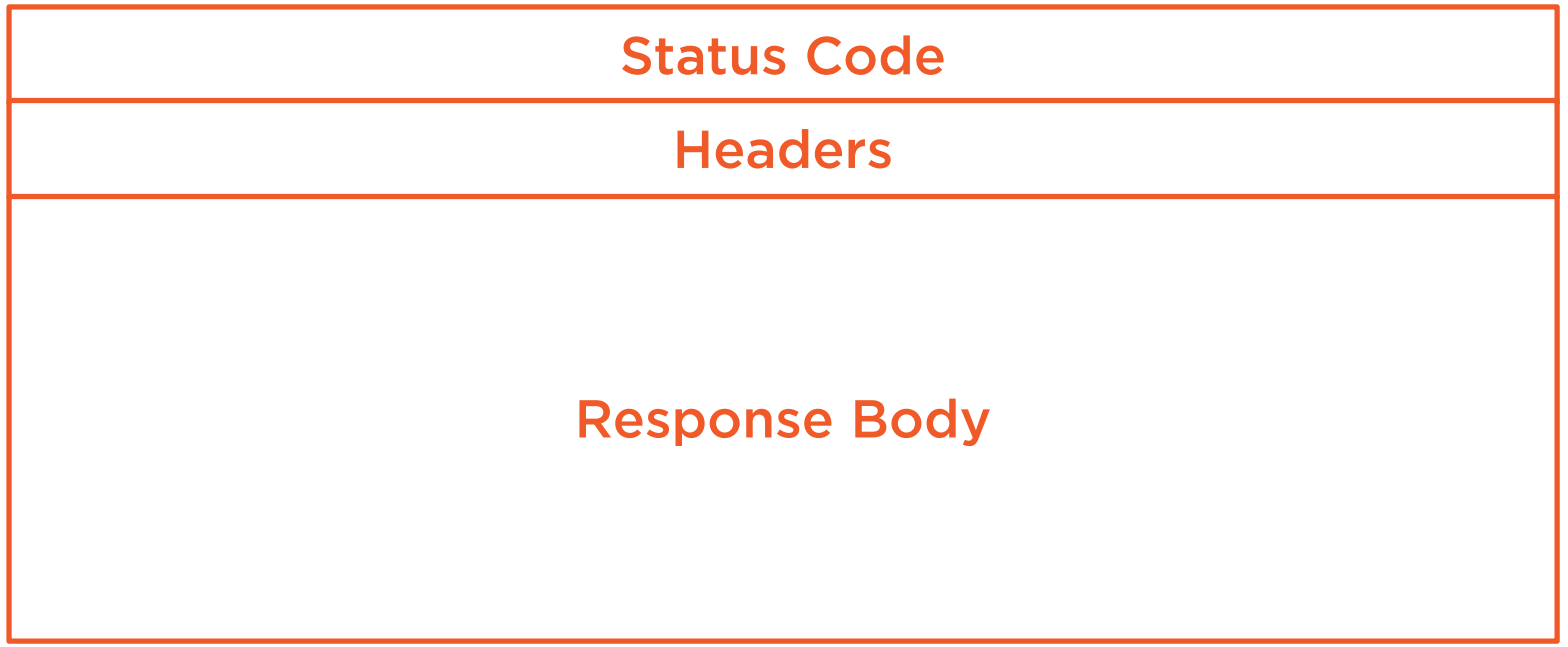
Design Your API First

- Can’t fix an API after publishing
- Too easy to add ad-hoc endpoints
- Helps understand the requirements
- Well designed API can mature

REST APIs Have Several Parts



REST APIs Have Several Parts



In Designing APIs, What Are URIs?

URIs are just paths to Resources

- E.g. api.yourserver.com/people

Query Strings for non-data elements

- E.g. format, sorting, searching, etc.

API Design

- Nouns are Good, Verbs are Bad

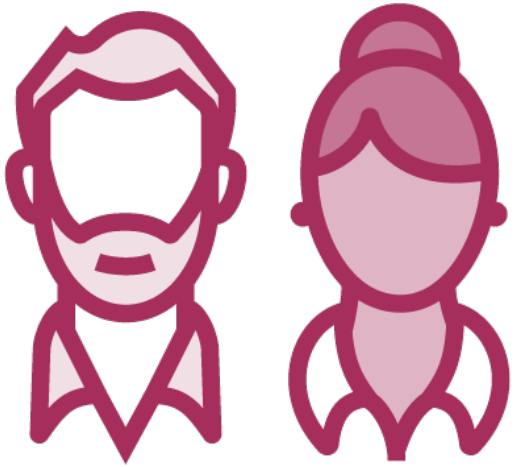
```
/getCustomers
/getCustomersByName
/getCustomersByPhone
/getNewCustomers
/verifyCredit
/saveCustomer
/updateCustomer
/deleteCustomer
...
```

API Design

- Prefer Nouns

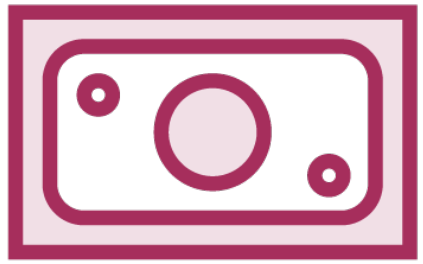
```
/customers
/invoices
/products
/employees
...
```

What Are “Resources”?



People

Invoices



Payments

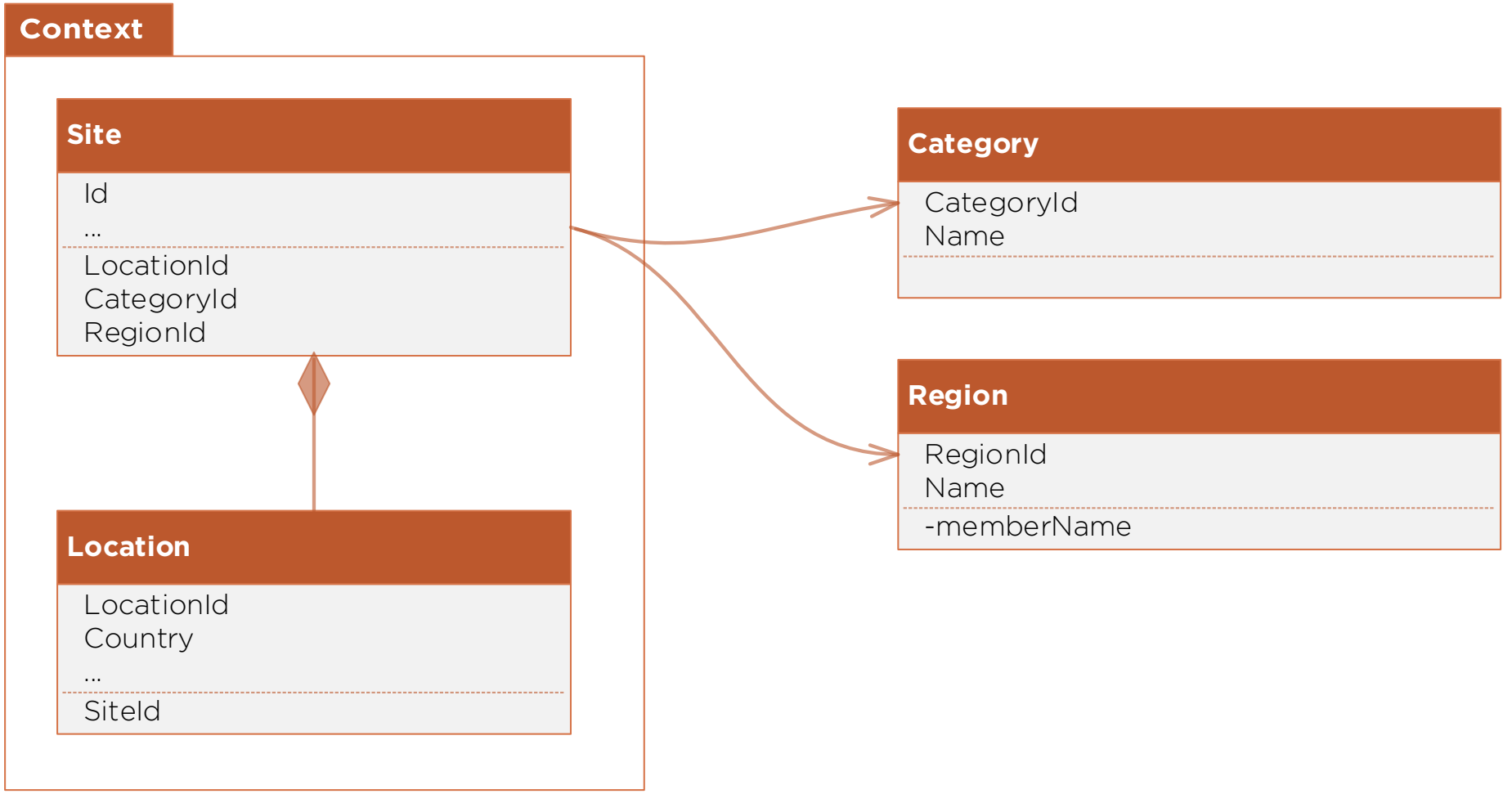
Products

Are Resources Just Entities?

Resource

Context

The Resources



Identifiers in URIs

- Use unique identifiers
- Does not have to be ‘primary keys’

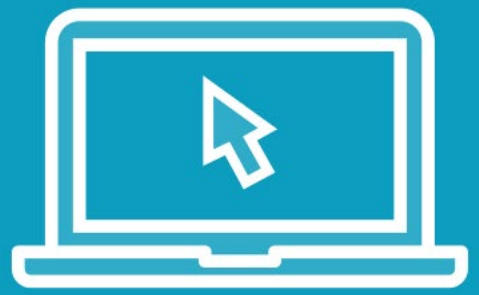
```
/sites
/sites/1
/sites/stone-henge
/sites/uk-101
...
```

Query Strings

- Use for non-resource properties

```
/sites?sort=name
/sites?page=1
/sites?format=json
...
```

Demo



Using URIs

GET

- Retrieve a resource

POST

- Add a new resource

PUT

- Update an existing resource

PATCH

- Update a resource with changes

DELETE

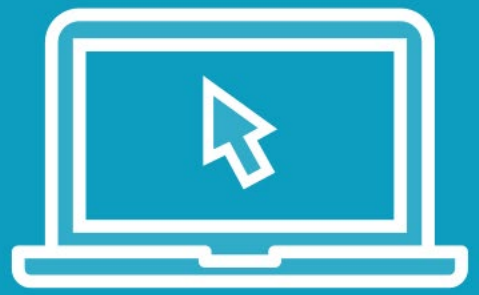
- Remove the existing resource

Verbs and URIs

How do I do Verbs then?

Resource	GET (read)	POST (create)	PUT (update)	DELETE (delete)
/customers	Get List	Create Item	Update Batch	Error
/customers/123	Get Item	Error	Update Item	Delete Item

Demo



Using Verbs

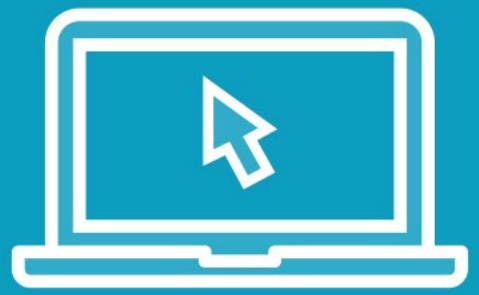
Idempotent

adj: operation that can be applied multiple times without changing the result

Idempotency in REST

- Operations result in same side effect
  - GET, PUT, PATCH and DELETE
- POST is not idempotent

Demo



Understanding Idempotency

Designing Results

```
{
  "id": 1557,
  "name": "Aasivissuit",
  "yearInscribed": 2018,
  "url": "https://...",
  "imageUrl": "https://...",
  "descriptionMarkup":
     "Located inside the Arctic",
  "states": "Denmark",
  "location": {...},
  "category": {...},
  "region": {...}
}
```

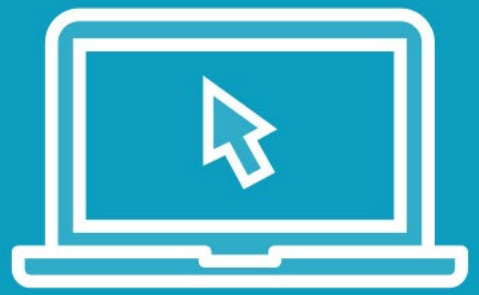
Designing Results

- Members Names
  - Shouldn't Expose Server Details
    - E.g. Ruby, Java or .NET
    - I Prefer camelCasing
    - If objectionable...
    - ...at least be consistent

Designing Collections

```
{
  "totalResults": 245,
  "nextPageUrl": "/api/sites?page=2",
  "results": [{
    "id": 1557,
    "name": "Aasivissuit",
    "yearInscribed": 2018,
    "url": "https://...",
    "imageUrl": "https://...",
    ...
  }]
}
```

Demo



Designing Your Results

Decide Formats During Design

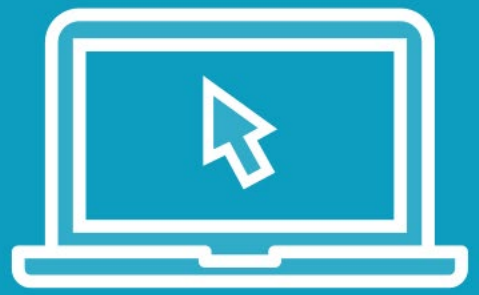
```
// Abide by Accept Header:
Accept: application/json, text/xml

// Return sane default (usually JSON)
Content Type: application/json

// Prefer not to use query strings for formats
/api/customer?format=json // <- Antipattern
```

Common Formats:

- JSON: application/json
- XML: text/xml
- JSONP: application/javascript*
- RSS: application/xml+rss
- ATOM: application/xml+atom

* Requires callback parameter

Demo



Formatting Results

Hypermedia

- Allows results to be self-describing
- Allows programmatic navigation
- Adds complexity

Hypermedia

```
{
  "results": [{
    "id": 1557,
    "name": "Aasivissuit",
    "yearInscribed": 2018,
    "url": "https://...",
    "_links": {
      "self": "/api/site/1",
      "region": "/api/region/us",
      "relatedSites": "/api/region/us/sites"
    }
    ...
  }]
}
```

Hypermedia can be helpful…but pragmatism means that most projects don’t need it.

What We've Learned

APIs should be based on simple URIs that are understandable

Match verbs to intended operations to improve usability of our APIs

Designing an API means designing it all, not just URIs

Coming Up:
Handling More Complex Scenarios