

**W4156**

**API Economy & RESTful  
API Design**

# Moving On

We are going to move on from OO

The next mini-arc will be on Architecture

- What is Architecture?
- Architectural Styles
- How do you Architect a system?
- Scalability

Before that we are going to discuss APIs and REST as a precursor

# Agenda

- ❑ Defining APIs
- ❑ API Motivations
- ❑ REST & RESTful APIs
- ❑ Designing RESTful APIs
- ❑ Supporting Technology (Swagger)

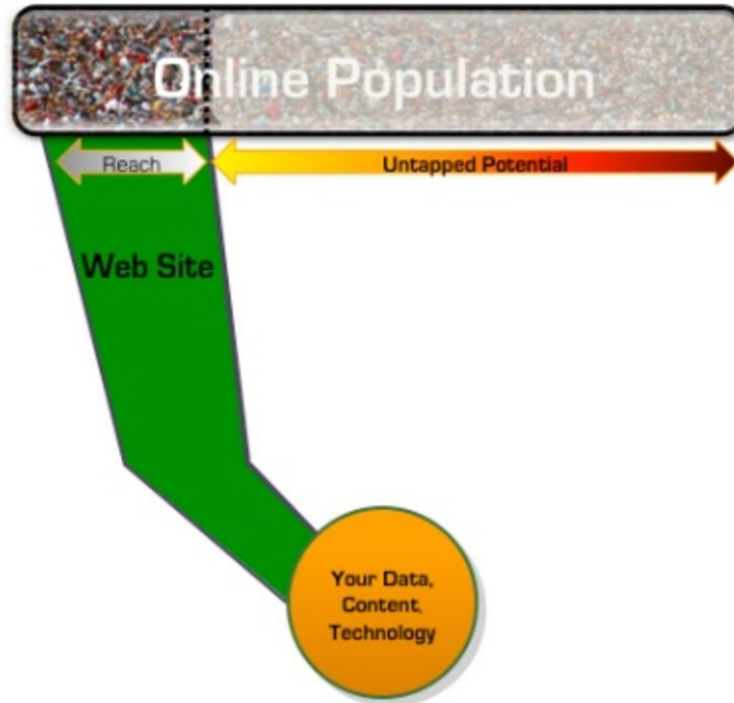
# API

**Application Programming Interface (API)** is a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components.

1. *The precision required to agree the contract between two pieces of functionality is non-trivial.  
What can you do? When can I call it? What will the result be? What is the object I get back? What states do you and I transition through? What happens in the event of failure?  
Example: Can you ask Ewan to book a flight?*
2. *Some people may initially associate with the interface between code within a process but also used to define the wider interface between components. I consider levels of abstraction*
  - *Code:Code*
  - *Microservice:Microservice within an application*
  - *Application:Application*
  - *Client:Business*

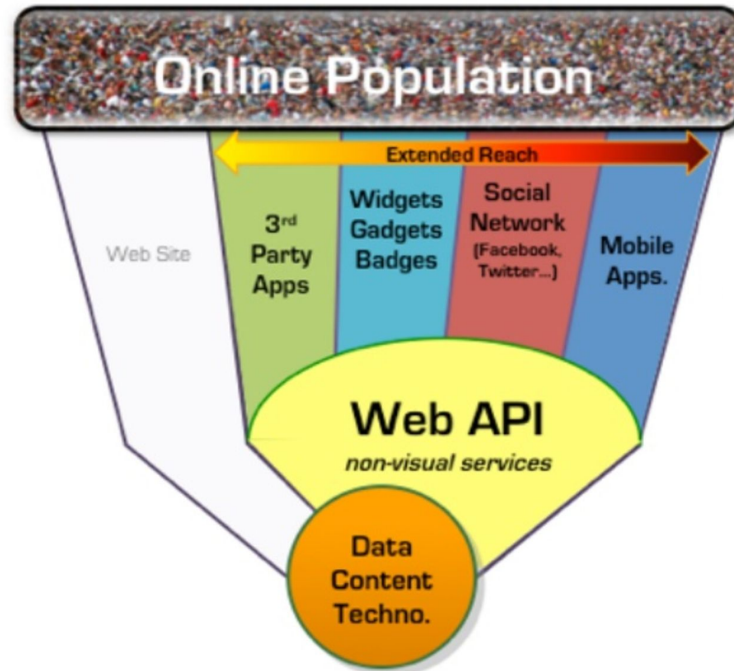
# Motivation 1: Commercial

# API Economy: Changing Delivery Channels

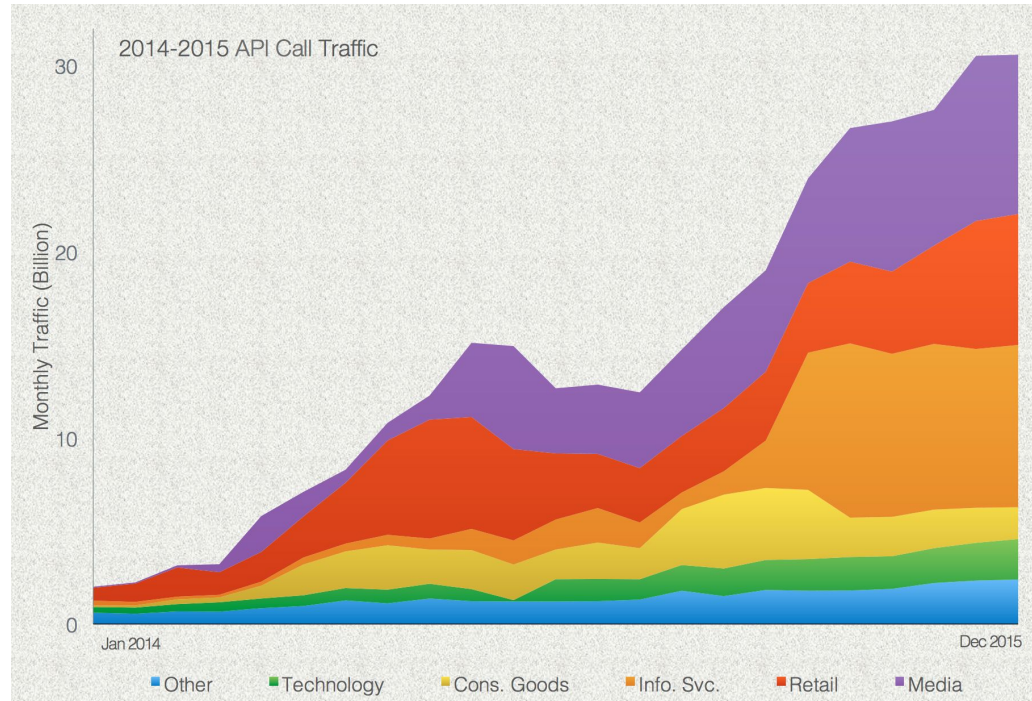


Source: Mulesoft the [API Economy](#)

# API Economy



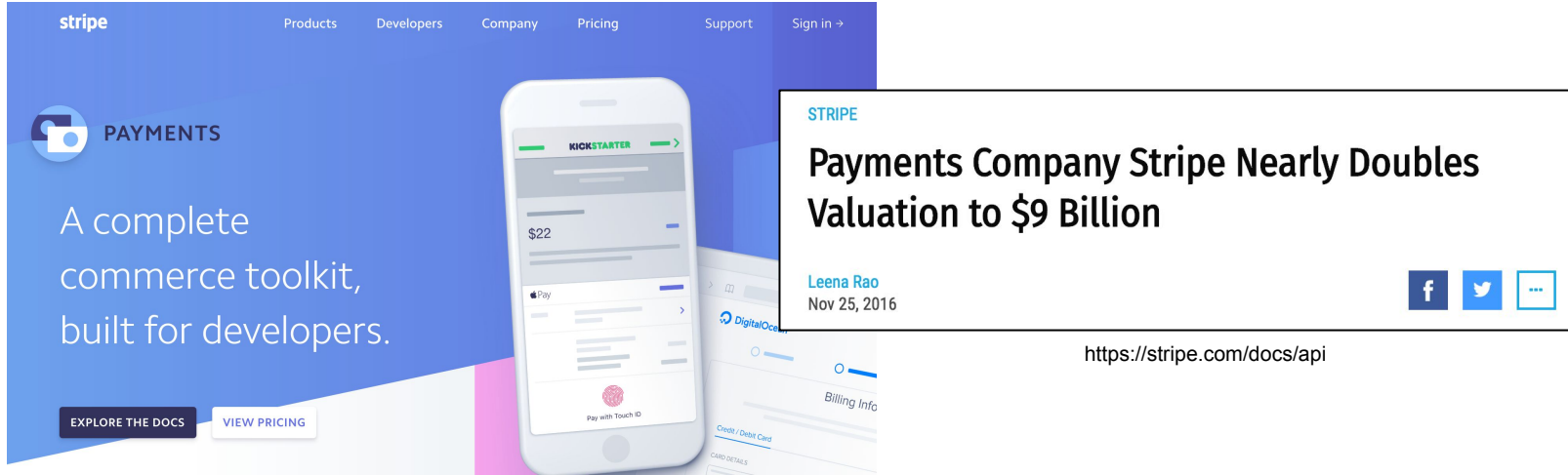
# API Growth



Source: Apigee State of APIs Report 2016



# Sometimes API *is the product*



The image shows a screenshot of the Stripe website. The top navigation bar includes links for Products, Developers, Company, Pricing, Support, and Sign In. The main header features the Stripe logo and the word "PAYMENTS". Below this, a large blue section contains the text "A complete commerce toolkit, built for developers." and two buttons: "EXPLORE THE DOCS" and "VIEW PRICING". In the center, there is a smartphone displaying a Kickstarter payment interface with a \$22 amount and a "Pay with Touch ID" button. To the right, a white box contains a news article snippet titled "Payments Company Stripe Nearly Doubles Valuation to \$9 Billion" by Leena Rao, dated Nov 25, 2016. Social media icons for Facebook, Twitter, and a generic share icon are present. Below the article box, the URL <https://stripe.com/docs/api> is displayed.

stripe Products Developers Company Pricing Support Sign In →

**PAYMENTS**

A complete commerce toolkit, built for developers.

EXPLORE THE DOCS VIEW PRICING

**STRIPE**

**Payments Company Stripe Nearly Doubles Valuation to \$9 Billion**

Leena Rao  
Nov 25, 2016

<https://stripe.com/docs/api>

today, according to [programmableweb.com](http://programmableweb.com). [Salesforce.com](http://Salesforce.com) generates 50% of its revenue through APIs, [Expedia.com](http://Expedia.com) generates 90%, and [eBay](http://eBay), 60%. [Salesforce.com](http://Salesforce.com) has a

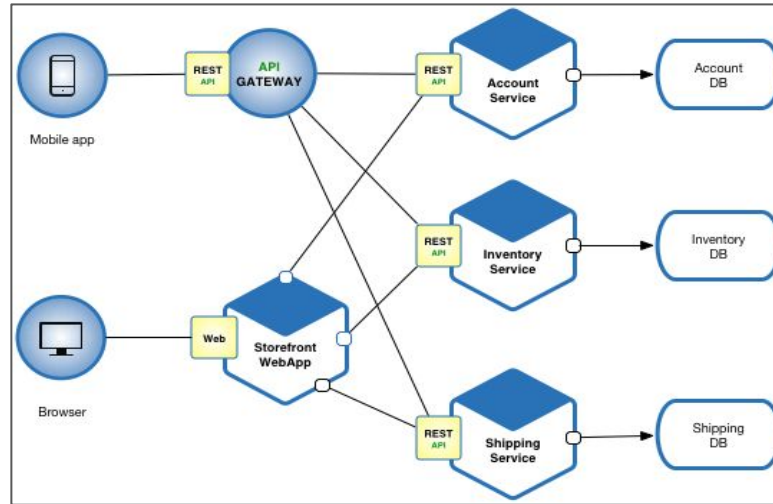
# Delivery

We need to deliver our business service to our customers/users.

We want to support channels where there is viable revenue: desktop, web,  
mobile or API

## Motivation 2: Engineering

# Internal APIs / Decomposition



<http://microservices.io/patterns/microservices.html>

When we *choose* to decompose our application into distinct components some form of API is necessary to integrate those components

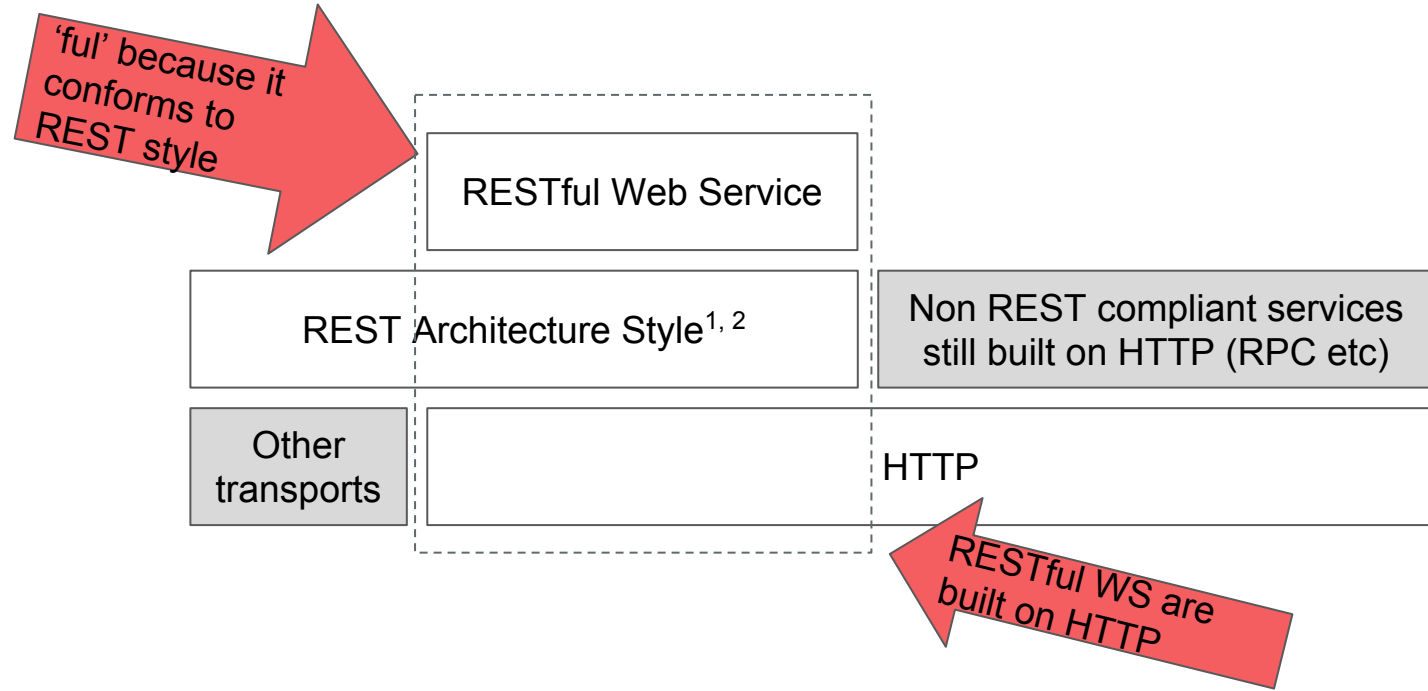
**REST and RESTful?**

# REST Architectural Style

*“Representational State Transfer (**REST**) is an **architectural style** that specifies **constraints**, such as the uniform interface, that if applied to a **web service** induce **desirable properties**, such as performance, scalability, and modifiability, that enable **services** to work best on the **Web**.”*

*“the motivation for developing REST was to create an architectural model for how the Web should work, such that it could serve as the guiding framework for the Web protocol standards. REST has been applied to describe the desired Web architecture”*

# REST and RESTful



<sup>1</sup> Note REST is an architectural style not a technology or framework. Diagram represents 'RESTful service built in that style)

<sup>2</sup> REST is a style. As such a REST compliance system could be build over other transports. In practice REST is almost always in the context of HTTP and RESTful web services

Elements of REST Architectural Style		Description
<b>Resources</b>		Everything is a resource with a resource identifier
<b>Representations</b>		The state of each resource has a representation (JSON, XML)
<b>Constraints</b>	Client-Server	Client Server Architecture
	Stateless	Servers are stateless. State on client side and transmitted as part of request
	Cacheable	Responses must be cacheable or identify if not cacheable
	Layered	Client can not be able to tell if connected directly or through intermediaries
	Uniform Interface	Irrespective of how either <i>side</i> is implemented they communicate over a <i>uniform</i> interface. (tradeoff of specialism/optimization vs independence, simplicity)
	Code on Demand (optional)	Server can enrich client behavior by providing scripts to run locally



# REST Benefit

*Why go to all this hassle/precision of defining this style?*

1. Can not communicate unless we have common communication framework!
2. Architectural style has several advantageous properties
  - Performance
  - Scalability
  - Reliability
  - Simplicity
  - Visibility
  - Interoperability
  - Independence

# Why do I want you to understand REST vs jumping into RESTful API design?

Engineers who do not understand the underlying constraints of REST and design APIs which *violate* those constraints will end up with APIs that are ugly, don't work or don't scale .....

Primary examples

- Designing APIs that mix nouns and verbs
- Not designing APIs that handle failure (idempotent)
- Designing APIs which use GET to implement

# RESTful Web Services / APIs

# (Aside - Paradigm)



You have a good command of the OO paradigm.

OO (objects, data, methods and interactions)

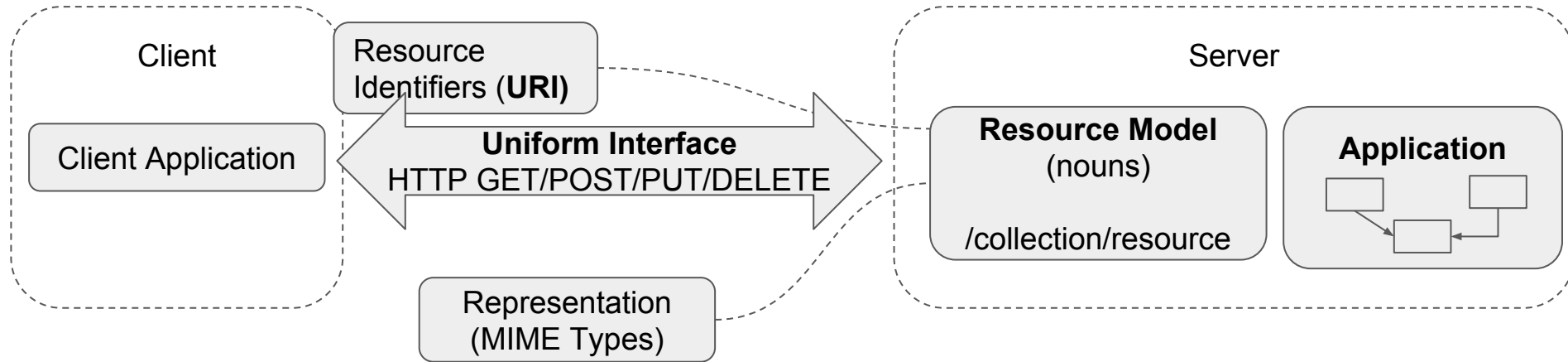
REST is a ***different paradigm*** (resources, identifiers, verbs)

***However***, at a very abstract level paradigms are:

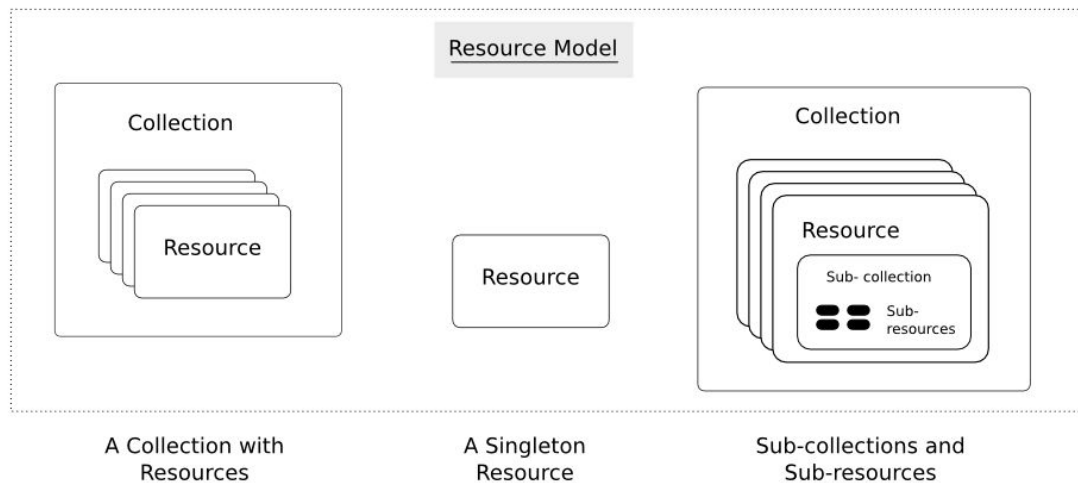
1. Problem to be solved
2. Facilities/constructs of the language/architecture
3. Way of thinking
4. Process to analyze problems within a paradigm

RESTful API Design		Description
<b>Resources</b>		Concepts modelled as <i>resources</i> (book, order, etc) accessible via URI http://foo.com/api/order
<b>Representations</b>		Resource state defined as JSON (less so XML)
<b>Constraints</b>	Client-Server	HTTP client server architecture (browser/application -> web server)
	Stateless	No state held on server
	Cacheable	Responses must be cacheable or identify if not cacheable (GETs can be cached)
	Layered	Local Caches, CDN, Caches, Web Servers
	Uniform Interface	HTTP Verbs (GET, PUT, POST, DELETE)
	Code on Demand	Client side javascript

# RESTful Abstract Model



# Resources



<http://restful-api-design.readthedocs.io/en/latest/resources.html>

# Designing RESTful Web APIs



# RESTful Design Process

We have a problem we have encountered before. How do we turn a problem domain into a RESTful API design of resources/nouns and actions?

Data oriented design vs process oriented  
<http://todayisapotato.github.io/rest-design-talk/#/6>

# Worked Example

Ewan has too many bikes and wants to build a system to keep track of them.

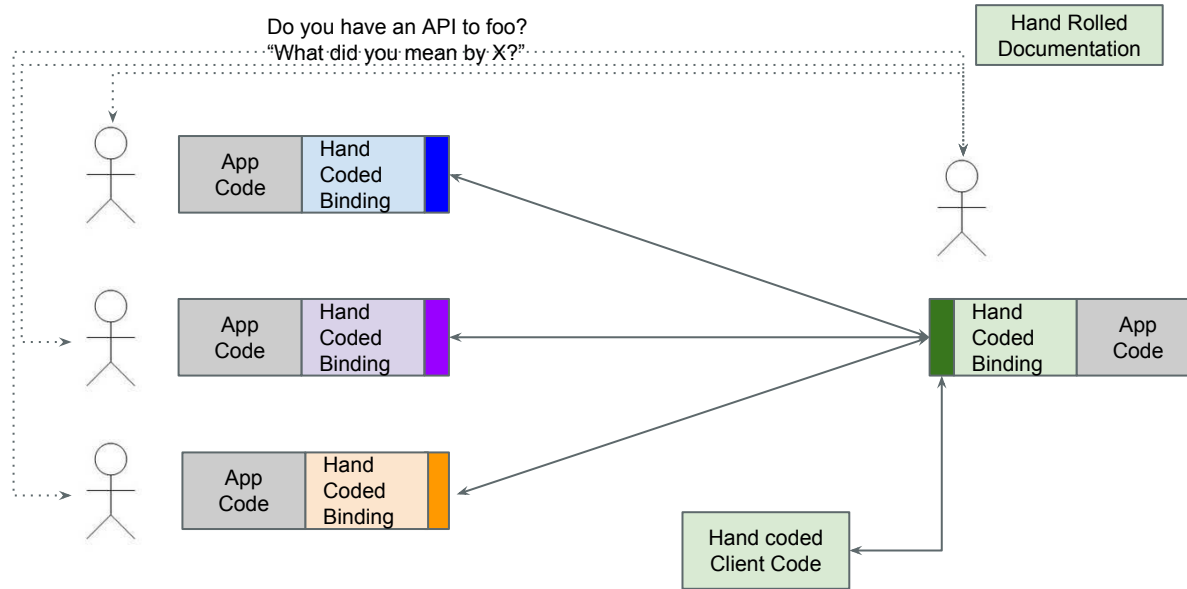
- What are the use-cases?
- What are the nouns?
- What are the verbs?
- What are their representations?

# Simple Worked Example

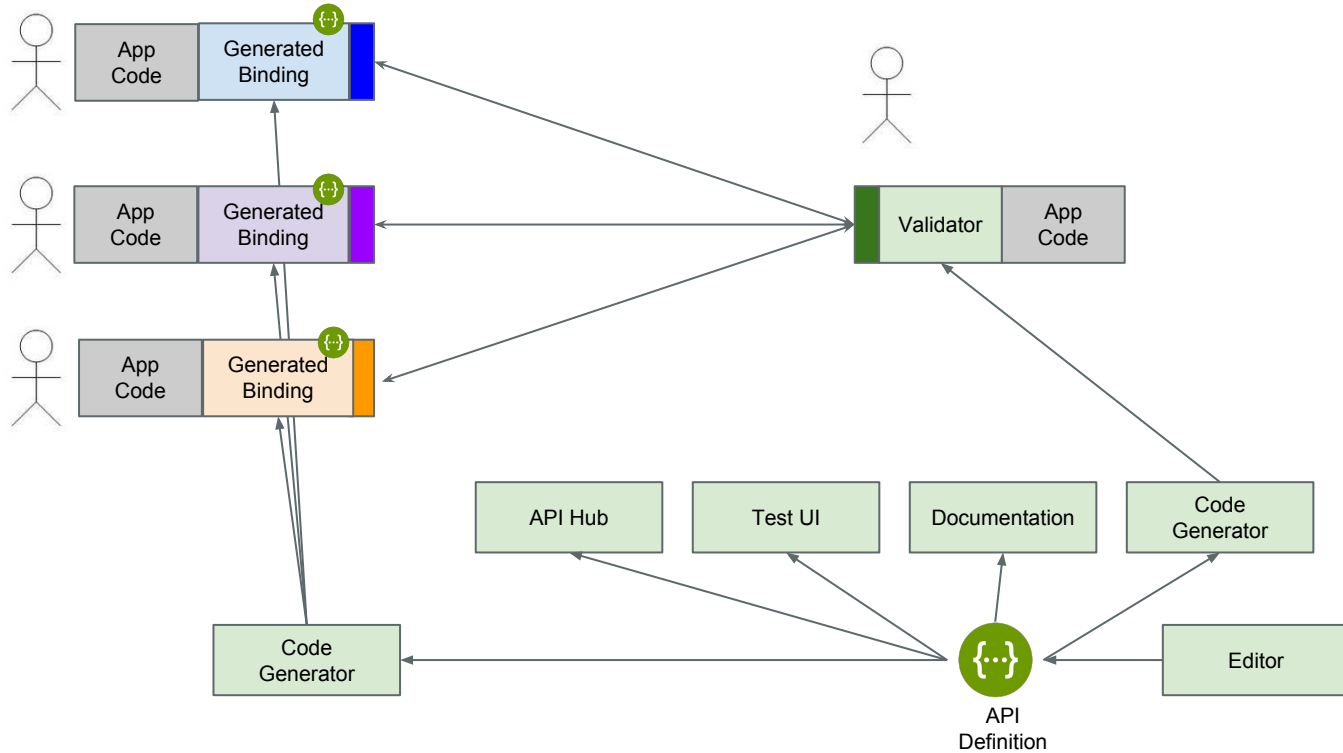
Resource	POST (create)	GET (read)	PUT (update)	DELETE (delete)
/cycles (collection)	Create a new bicycle	List all bicycles	Bulk update cycles	Delete all cycles
/cycles/1 (resource)	Error	Show the unicycle	If cycle exists update else error	Delete unicycle

# Technology Support

# A lot of repeated code to build each API ...



# Or as software engineers we can automate ...



# Pop Quiz

Question	Answer
APIs are {a channel to customers, a way of decomposing components, both}	
The difference between REST and RESTful web services is {...}?	
Within RESTful design the key is to identify {objects, nouns, resources, methods}?	
The verbs within RESTful design are {...} and the mapping of those verbs to C.R.U.D is {...}?	
The relationship between Swagger and RESTful APIs is {...}?	

# Reading

Reading	Optionality
<u>RESTful thinking</u>	Required (Process Oriented)
<u>Google Resource Oriented Design</u> <ul style="list-style-type: none"><li>- Resource Oriented Design</li><li>- Resource Names</li><li>- Standard Methods</li><li>- Naming Conventions</li></ul>	Required
<u>REST in simple terms</u>	Recommended