**API Product Mindset**

Design APIs as products, easy consumed by developers and how easily it can be leveraged in new ways in the future

**Important – Future use and extensibility**

1. Documentation
2. Consistent design standards
3. Versioning and Security
4. Should not be overdesigned
5. Ease of consumption
6. Consumption vs Exposure

**Success Stories**

* AccuWeather
* Pitney Bowes – GeoSearch API

**Product Manager**

1. Cant manage what they cant measure
2. API usage tie up with Key Performance Indicators (KPIs)
3. How often the API is called?
4. What kind of calls are being made?
5. Are consumers changing their behavior?

**API Management Pillars**

1. Outside-In – Customer obsession
2. Time to market – Define MVPs
3. Iteration – Mature product through iteration

**API Roles**

* API Product Manager
* API Architect
* API Developer
* API Evangelist
* API Champion

**API LifeCycle**

* Design -> Devlop -> Secure -> Publish -> Scale -> Monitor -> Analyze -> Monetize – Design

Project Mindset vs Product Mindset

|  |  |  |
| --- | --- | --- |
|  | Project Mindset | Product Mindset |
| Perspective | Inside-Out | Outside-In |
| Goals | Completion | Consumption |
| Metrics | Completion, Cost, Acceptance | Consumption, Review, Customer Satisfaction |
| Requirements | Exhaustive | Iterative |
| Teams | Intermittent | Continouous |

Web API Design

Quoting Antoine de Saint-Exupery, “perfection is achieved not when there is nothing more to add, but when there is nothing more to take away.”

API design

1. Representation of resources - Definition of the fields in the resources
2. Use of Standards – HTTP headers?
3. URLs and URI templates – Query interface for locating resources
4. Required behaviors by clients – DNS caching behaviors, retry behaviors, tolerance of fields that were not previously present

Web API = representation JSON

Json = {null, string, number, Boolean}

Json <> {dates, urls}

Json = {name=property, value=entities}

The natural construct for representing relationships in http is the link

What is hypermedia as the engine of application state

Drawbacks of traditional way of writing rest apis

1. Relationship between entities are represented by not using links – you need to look for documentation rather than just look at the data; no good documentation convention for describing which property values found in representations can be plugged into which templates so some amount of guesswork is usually necessary, you need to write code to combine the ownerid property value with the template and produce a url- template processor

ATOM specification – IANA

https://apigee.com/collections#Collection

<https://apigee.com/collections#Page>

Versioning – use of PATCH

Produce json or html as output . SPAs access apis are completely opaque to search engines – single set of urls for api and spa and provide html output for each of these urls