



Hello!

I am Pete Smaluck

I am a Frontend dev at Unata.

You can find me at:

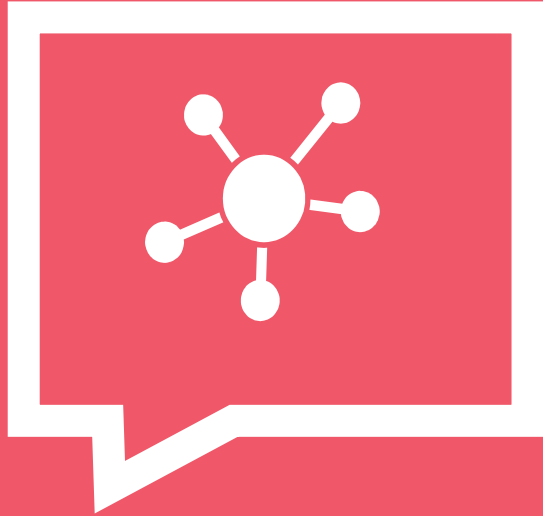
pete@unata.com



API Relationship Advice.. From a frontend dev

Agenda

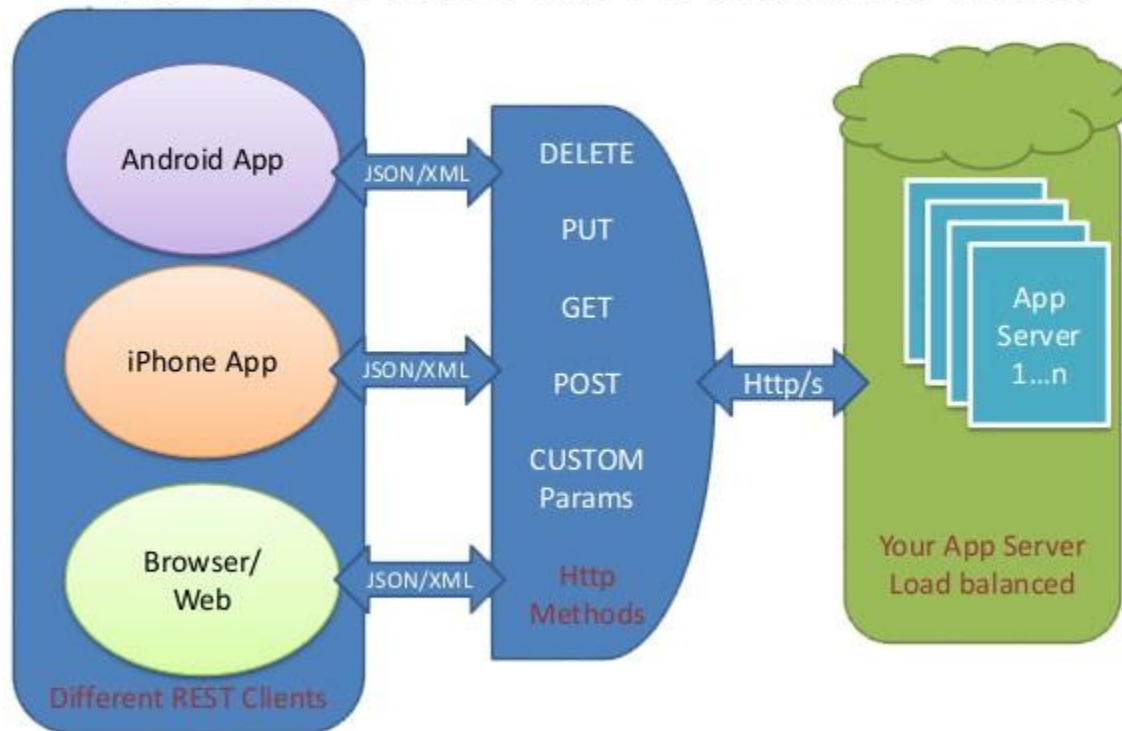
1. What is an API?
2. Backend and frontend traits
3. Building features together
4. Focusing on strengths
5. Communication
6. Succeeding together



WTF is an API?



Incase you didn't know..

REST API Architecture



Combining our Efforts

```
"teams" : {  
  "home" : {  
    "runs" : 8,  
    "hits" : 12,  
    "errors" : 0  
  },  
  "away" : {  
    "runs" : 6,  
    "hits" : 8,  
    "errors" : 0  
  }  
}
```

FINAL		R	H	E
	Blue Jays (9-19, 5-11 Away)	6	8	0
	Yankees (17-9, 12-3 Home)	8	12	0

Challenges..

- Separate business concerns
- Different skill sets
- Working in isolation for extended periods of time
- For frontends: not fully understanding the complexity of the backend
- Implementing features in parallel

1.

Considering our differences

How do backends and frontends differ?



“

*A great relationship is about two things.
First appreciating similarities, and second,
respecting the differences.*

- Unknown

Character Traits

Frontends

UX and the design
implementation

Performance

Beauty across devices

UI/Dialog/Buttons

Data structures

Backends

The storage of data

Data structures

Calculations

Performance

Keeping low response times

Full Stack Devs

Enjoy the single life...

2.

Focusing on each others' strengths

Frontends working with the app data

**Frontend devs work with
the app's data...**

When it is appropriate

1. To take a load off the server

- Payloads sent to the server can often help to take the load off server calculations

2. For faster feedback to the user



- Sometimes we do tricks to mimic data change prior to receiving and API response

When it is NOT appropriate

1. Too much data!
2. Important business logic
3. Personally Identifiable Information (PII) concerns
4. Permissions logic

Some tips for working with data on FE

1. Using Lodash/Underscore/RxJS/Redux.
2. How is the state of the app being handled? It's nice when the backend handles the state of the app.
3. If there is data manipulation it's best if the patterns can be repeated, so we can build a generic *service* to handle similar cases.

Generic Services

Are we building the frontend in such a way to be able to switch APIs tomorrow?

Service layers help accomplish this.

3.

Practicing Patience

When frontend development moves ahead of backend work.



Ideal situation..

API development is completed prior to the UI development.

But this rarely happens.

Features are often built in parallel.

Resolutions

1. Mock backends

- Node.js - Allows FE's to work independently and offline

2. Virtual Machines

- Vagrant, Docker - allows FE's to use a partially complete API

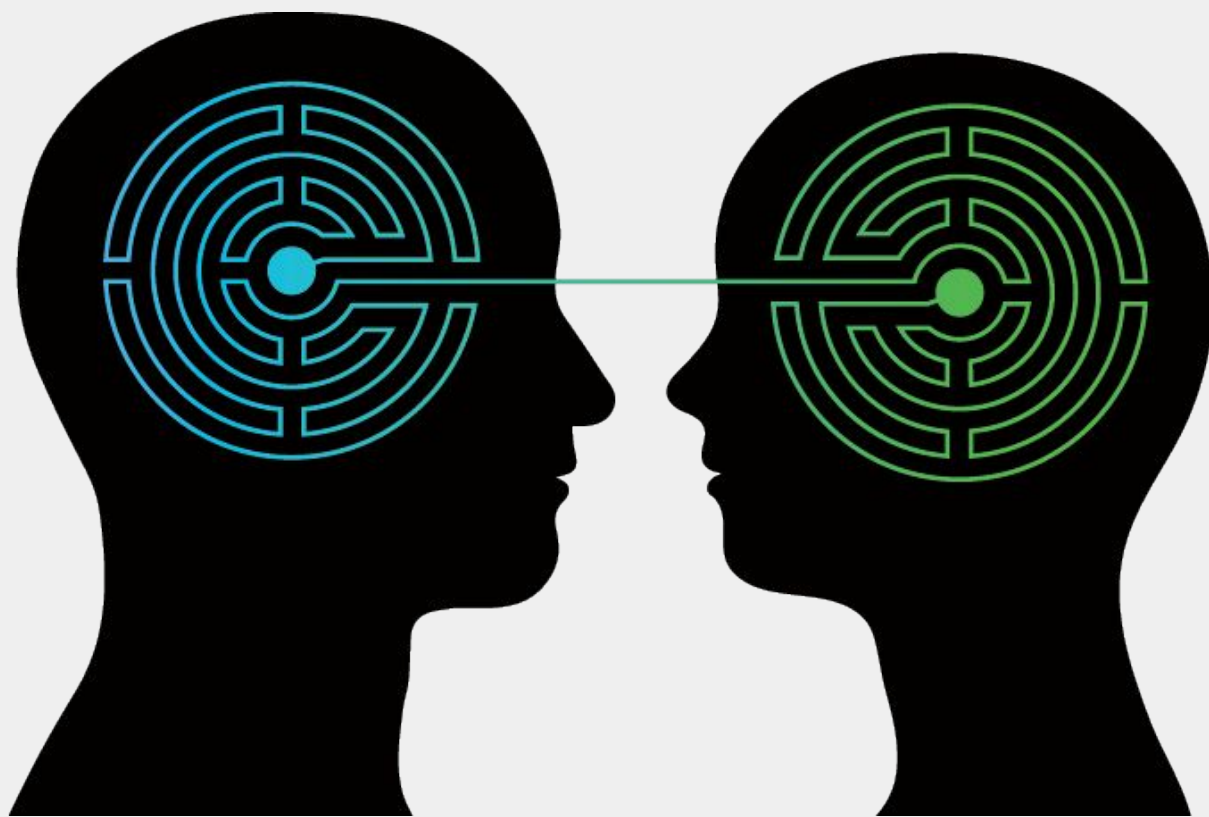
3. Some type of temporary backend

4.

Clear Communication



Documentation and Discussion



Keeping documentation up to date!

It's hard to succeed without great docs!

Documentation living within the code. Having them separate often leads to stale documentation

PUT /cart

Add/update a list of items in to the cart. If an item's quantity is 0, the item will be deleted by the server.

Error 40001: Missing fields

Error 40002: Invalid field values

Error 40100: Not authenticated

Error 40401: Resource not found

Example request:

```
PUT /cart HTTP/1.1
Accept: application/json
Content-Type: application/json

{
  "items": [
    {
      "item_type": "store_product",
      "store_product": {"href": "/store_products/1"},
      "allow_substitutions": true,
      "comment": null,
      "order_by_weight": false,
      "quantity": 1,
      "unit_price": "5.00",
      "subtotal": "5.00"
    }
  ]
}
```

Talking?

Talking to each other is also helpful. Who knew?
At Unata we use a pod structure to enable this.

During feature planning, both sides can agree on
a rough draft of the API.

5.

**We Succeeded
Together.**



Well designed endpoints allow for easier development on any device.

What makes a good endpoint..

- Follows web standards
- Hides complexity from the user
- Has proper versioning of the API
- Developer friendly to read and understand
 - Intuitive to users

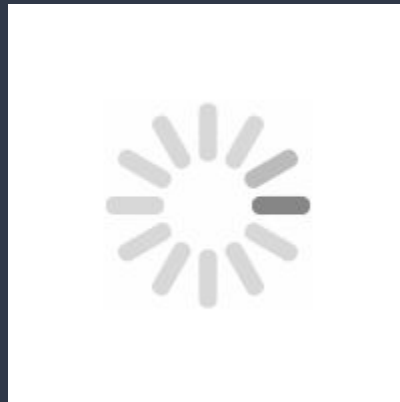
For Instance

GET /order/12

It's intuitive!

It retrieves order 12!

Quicker API response times improve user experience.



Headaches that are worth it

Sometimes Request Payloads seem a bit odd, but there is often good reasons ..

- aid the backend by limiting computation complexity (ie. shopping carts)
- Allows people to edit the same cart on different devices while maintaining state

Summary

1. Understanding our differences
2. Focusing on team members' strengths
3. Techniques when teams develop features in parallel
4. Find ways to communicate through documentation and dialog
5. A well built app depends on both teams.
6. Sometimes headaches are worth it in the long run.

Thanks!

Any questions?

You can find me at:
pete@unata.com