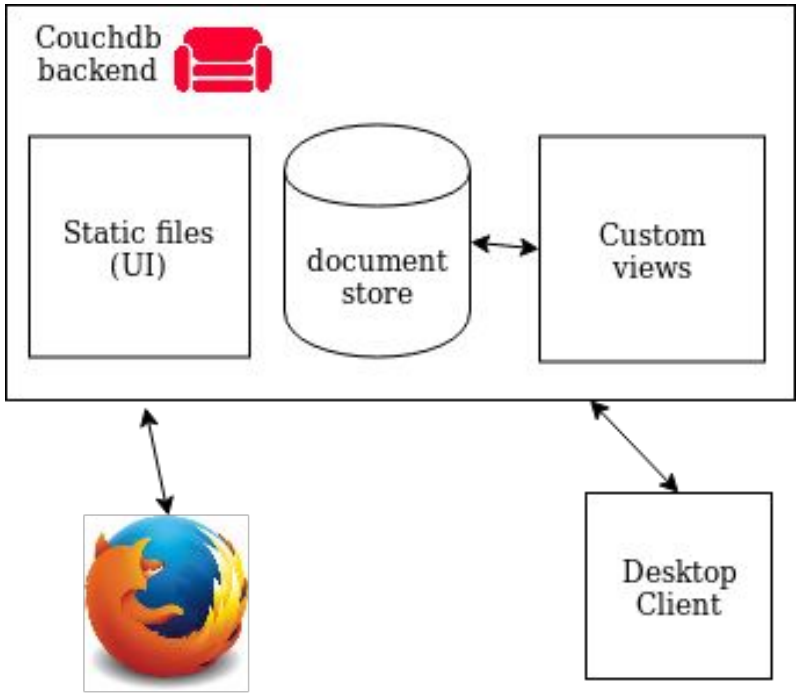
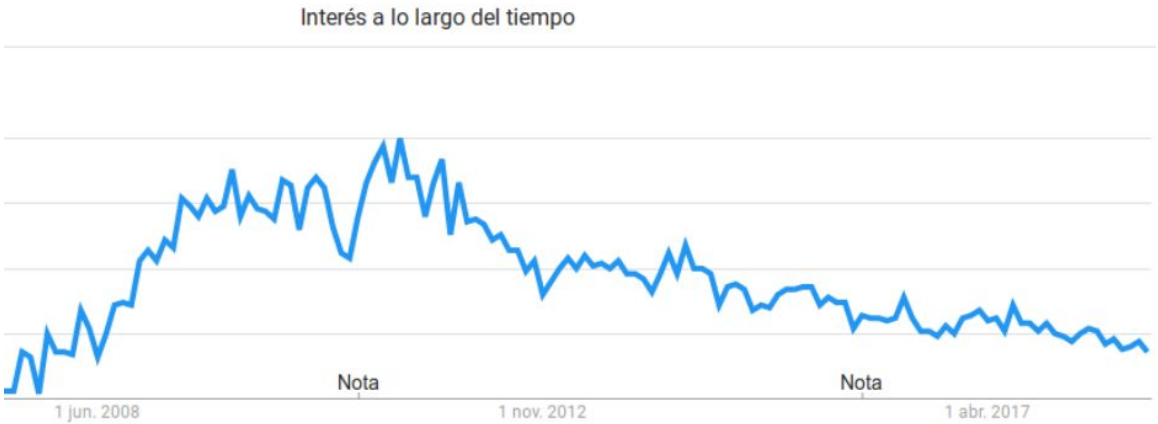
Sobreviviendo al síndrome del segundo sistema con Python

Arquitectura inicial

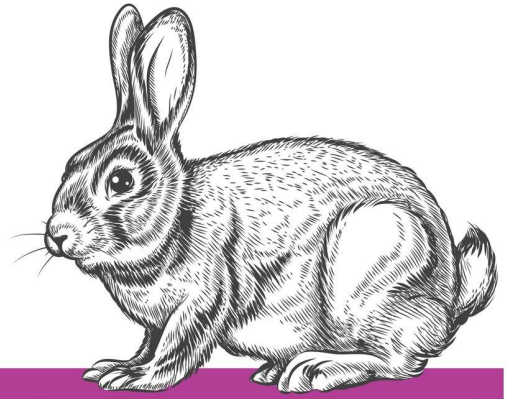


Ventajas de CouchDB:

- Nos daba un backend con un api rest
- Ayudó a tener un producto en poco tiempo
- Replicación



Depending on a vague popularity contest



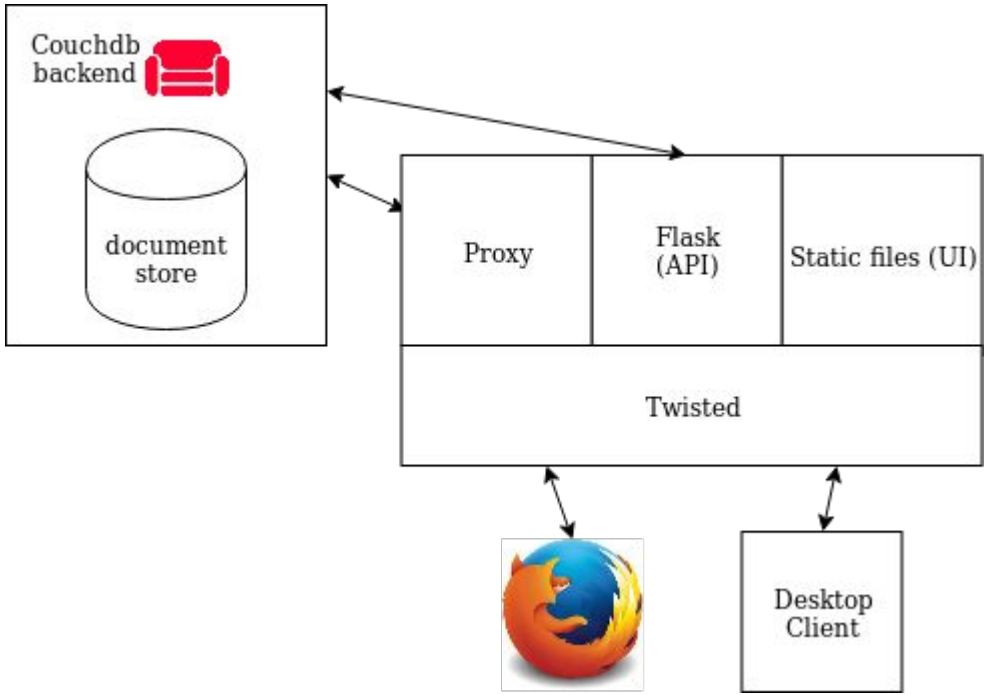
Choosing Based on GitHub Stars

You Only Live Once

O RLY?

@ThePracticalDev

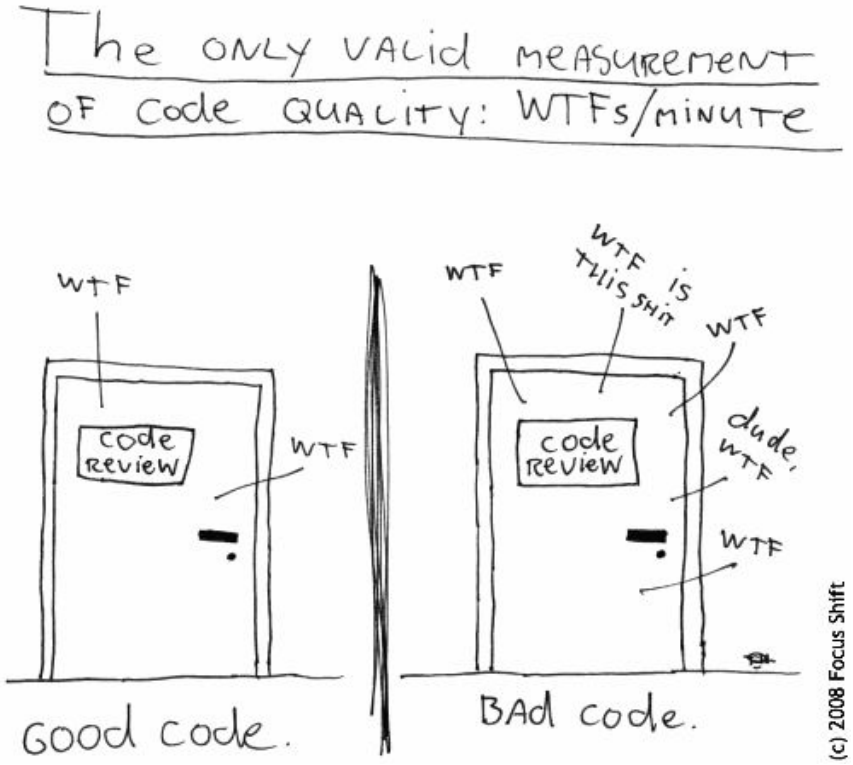
Arquitectura con backend python



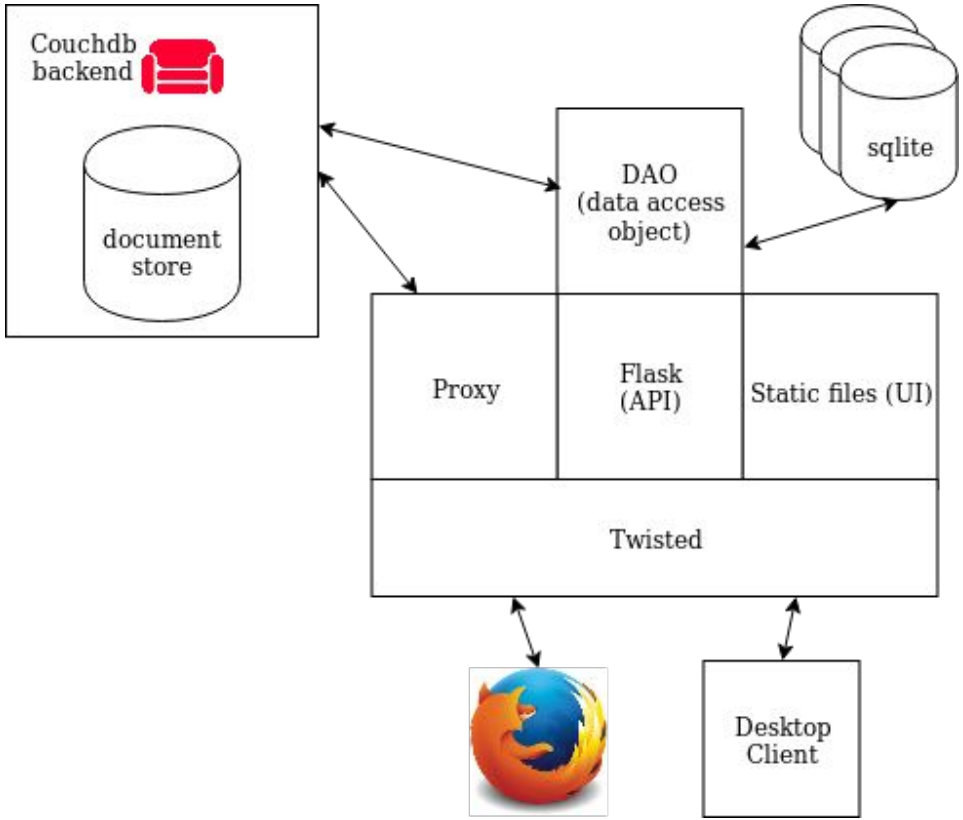
Se empezó a usar CouchDB sólo como DB (mantuvimos compatibilidad con las vistas viejas)

Deuda técnica acumulada

- Se terminó usando foreign keys con una DB que no lo soporta.
- CouchDB versión 2 no es backwards compatible.
- Documentos con distinto esquema de datos (de versiones viejas)



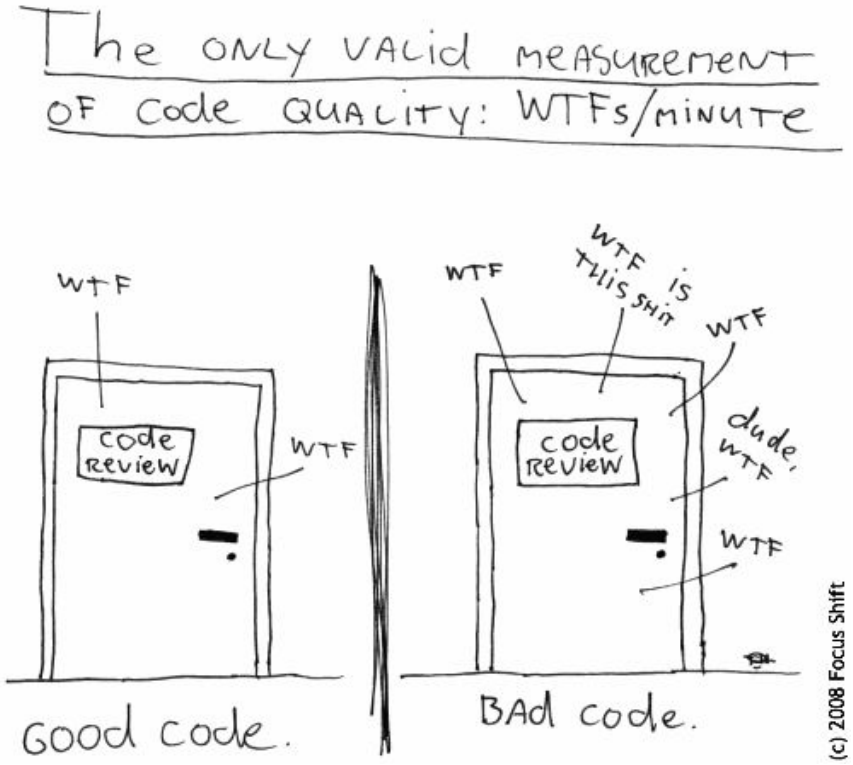
Arquitectura final (antes del refactor)




Mmmm..
Mira como te mira Conan..

Deuda técnica acumulada

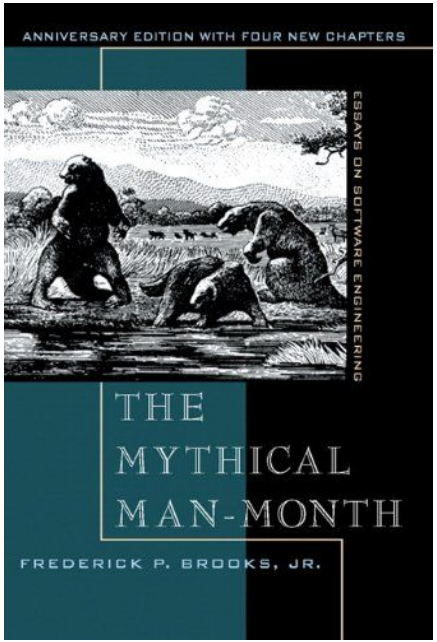
- Uso de SQLite para arreglar problemas de performance.
- Podían existir datos en CouchDB y no en sqlite o viceversa.
- Solo se podía correr una instancia del server.



Proceso de refactor

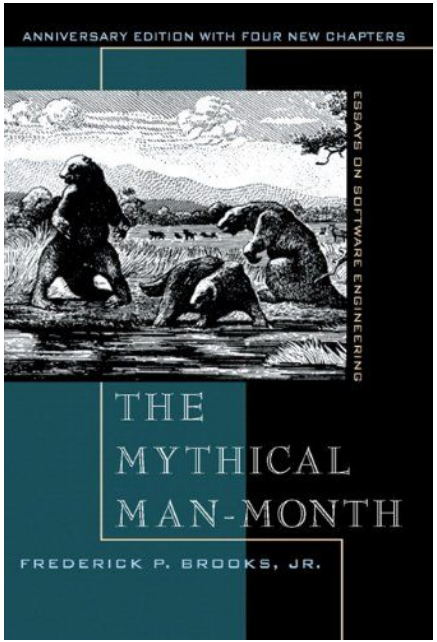
Second System Syndrome

“The first time you use a new technology or build a new type of system, you know that you're a beginner, so you tend to be naturally conservative.“



Second System Syndrome

“The general tendency is to over-design the second system, using all the ideas and frills that were cautiously sidetracked on the first one. “



El plan

- Reemplazar CouchDB por otra base de datos.
- Mantener compatibilidad hacia atrás en las respuestas de la API
- Cambiar la menor cantidad de código javascript (frontend).
- Sin deadline.

Eligiendo un motor de base de datos

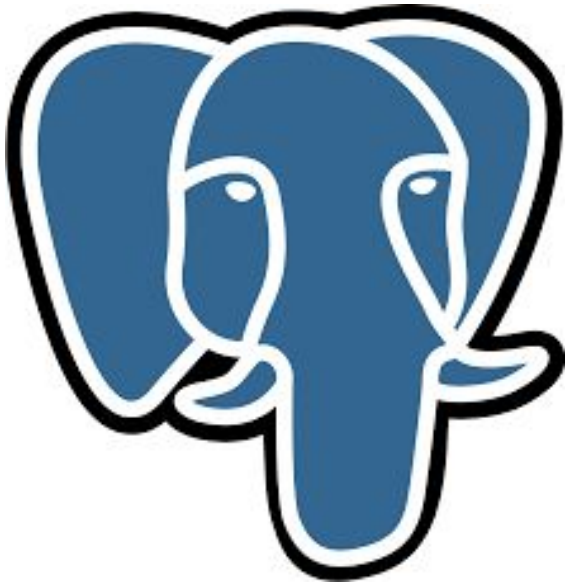
- SQLite
  - Problema de locking
  - No requiere servicio corriendo
- PostgreSQL
  - Implementa mejor el estándar SQL
  - Estricto con los tipos de datos, respeta restricciones
- MariaDB
  - No es tan estricto con tipos de datos
  - Comunidad dividida (MariaDB/MySQL)
  - Insert de strings en columnas de tipo integer
  - Select 1000/0 -> NULL
- MongoDB
  - Tiene foreign keys/joins (?)
  - Sin esquema

Para más información:
http://bit.ly/video-mariadb-problemas

NoSQL vs SQL

- Con CouchDB cada developer hacia lo que quería y no había un schema bien definido.
- Introducir un bug tenía mucho impacto, por ejemplo calcular mal el hash de un objeto
- En SQL no se pueden tener datos inválidos (con MySQL quizá si).

Ganador: PostgreSQL

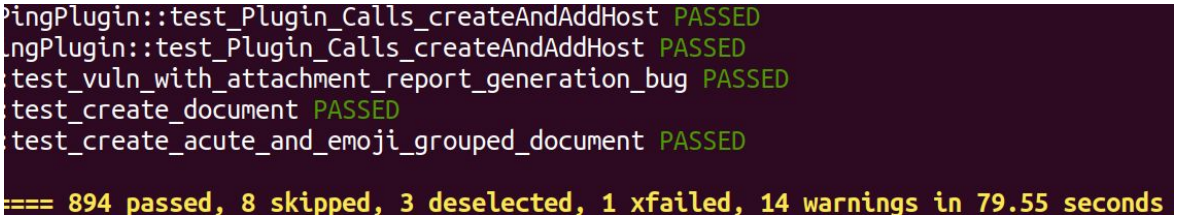


Armado del esquema e integración con Python

- Le dedicamos alrededor de un mes.
- Se validó con todos.
- Teníamos la ventaja de conocer los modelos en detalle (no era un app de cero).
- Usamos SQLAlchemy como ORM.
- Crear un script que importe los datos de CouchDB a PostgreSQL

Testing

- Flask permite testear muy fácilmente los endpoints de la api
- Usamos principalmente tests impuros (le pegan a la DB)
- Se agregó un CI
- Para los tests usamos:
  - pytest
  - factoryboy
  - hypothesis



Ejemplo de un test

```
59    def test_create_service(self, test_client, host, session):
60        session.commit()
61        data = {
62            "name": "ftp",
63            "description": "test. test",
64            "owned": False,
65            "ports": [21],
66            "protocol": "tcp",
67            "status": "open",
68            "parent": host.id
69        }
70        res = test_client.post(self.url(), data=data)
71        assert res.status_code == 201
72        service = Service.query.get(res.json['_id'])
73        assert service.name == "ftp"
74        assert service.port == 21
75        assert service.host is host
76
```

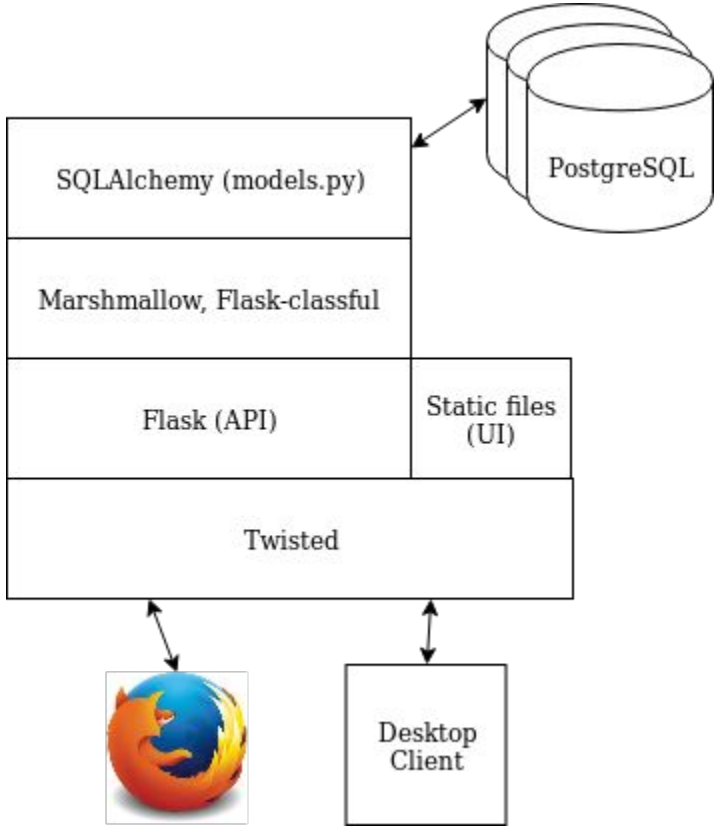
Overview de nuestro REST Framework flaskero

Librerías usadas:

- Marshmallow ❤❤❤
- Flask-Classful
- Flask-Security
- FilterAlchemy 💔
- Depot ❤

```
21
22 class LicenseSchema(AutoSchema):
23     _id = fields.Integer(dump_only=True, attribute='id')
24     end = StrictDateTimeField(load_as_tz_aware=False, attribute='end_date')
25     start = StrictDateTimeField(load_as_tz_aware=False, attribute='start_date')
26     lictype = NullToBlankString(attribute='type')
27
28     class Meta:
29         model = License
30         fields = ('_id', 'id', 'product',
31                   'start', 'end', 'lictype',
32                   'notes')
33
34
35 class LicenseView(ReadWriteView):
36     route_base = 'licenses'
37     model_class = License
38     schema_class = LicenseSchema
39
40
41 LicenseView.register(license_api)
```

Arquitectura final



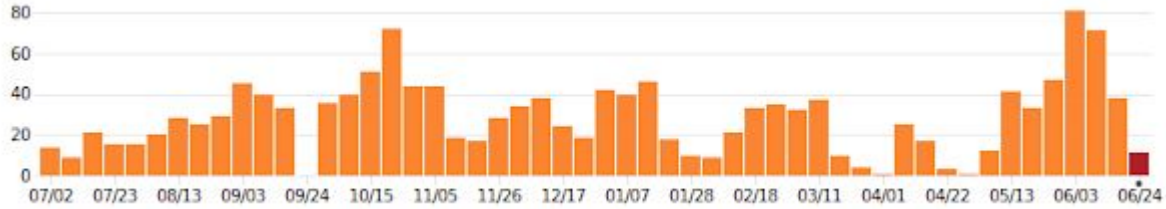
Estadísticas

29% de todos los commits corresponden a la migración.

El cambio nos llevó alrededor 10 meses full-time.

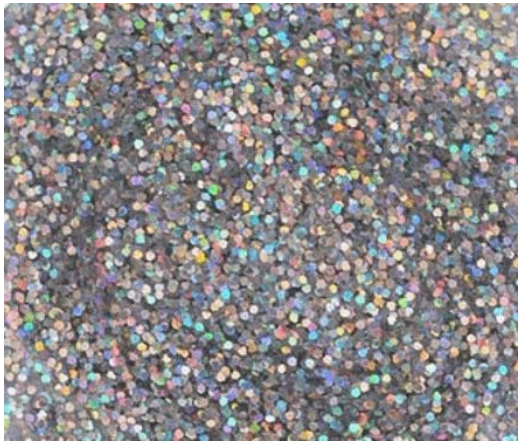
524 files changed, 32406 insertions(+), 16511 deletions(-)

↳8317 lineas de codigo nuevas corresponden a los tests



Conclusiones

- Los cambios tienen que estar bien coordinados entre todos los equipos del producto
- Poner un punto de corte
- Evitar caer en el shiny-object syndrome
- Hacer delete programming sin miedos (menos código, menos bugs)
- Se pueden hacer refactors importantes sin morir en el intento



Futuro

Soporte Python 3!
Documentación: sphinx, swagger
Nuevo frontend
Campos custom con columnas jsonb

Muchas gracias! Preguntas?

Matías Lang

@cript0nauta

Leandro Lazzaro

@llazzaro

https://github.com/infobyte/faraday/