When Python meets GraphQL

Managing contributor identities in your Open-source project

FOSDEM 2020 Python DevRoom

Bitergia

share this slide! @mghfdez

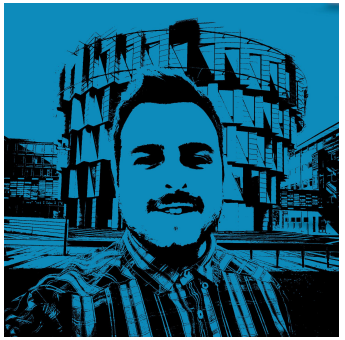
About me

My name is Miguel-Ángel Fernández

Working at Bitergia, part of the Engineering team

Software developer...

... also involved in stuff related with data and metrics



Bitergia

share this slide! @mghfdez

Bitergia

share this slide! @mghfdez

How can I measure my project?

How many contributors do we have ?

How many companies are contributing to my project?

Bitergia

share this slide! @mghfdez

It's all about identities

Tom Riddle

Affiliated to Slytherin, Hogwarts

Photo credit: julioolliveiraa

It’s all about identities

Lord Voldemort

Working as a freelance (dark) wizard

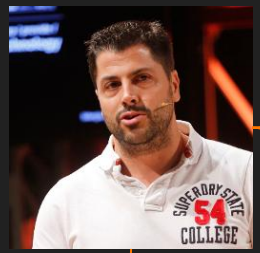
Photo credit: James Seattle

Wait… they are the same person!

Photo credit: julioolivеiraa

Photo credit: James Seattle

A little bit more complex



git

Manrique López <jsmanrique@bitergia.com>
Jose Manrique López de la Fuente <jsmanrique@gmail.com>
Manrique López <jsmanrique@gmail.com>

jsmanrique

Mailman

jsmanrique@gmail.com
jsmanrique@bitergia.com
correo@jsmanrique.es

PHABRICATOR

jsmanrique@bitergia.com

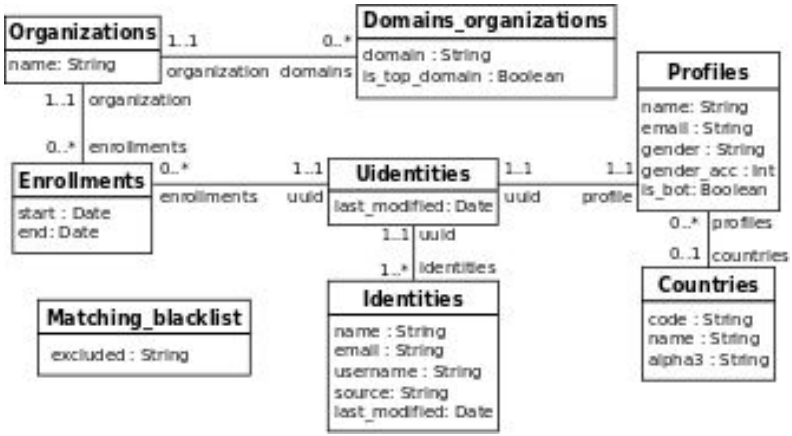
jsmanrique

02/2005 - 12/2010 CTIC
01/2010 - 12/2012 Andago
01/2013 - 06/2013 TapQuo
07/2013 - 12/2015 freelance (ASOLIF, CENATIC)
07/2013 - now Bitergia

Bitergia

Bitergia

“For I'm the famous **Sorting Hat**.
(...)
So put me on and you will know
Which house you should be in... ”



Bitergia

share this slide! @mghfdez

SortingHat: Wizardry on Software Project Members

Photo credit: James Seattle

Merge identities!

Lord Voldemort

Tom Riddle

Affiliate this person!

Complete the profile!

Name: Tom
Gender: Male

Email: tom@dark.wiz

Bitergia

share this slide! @mghfdez

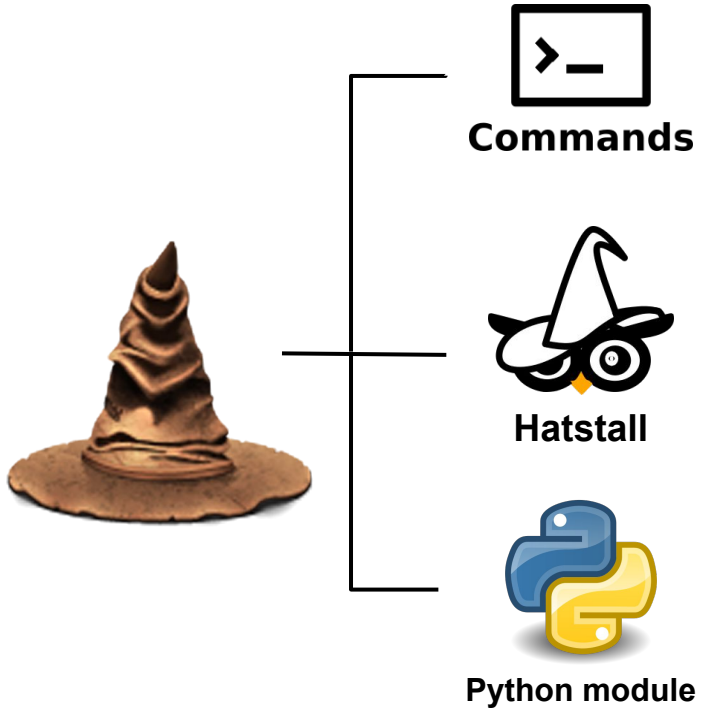
Boosting SH integration

Main idea: building a robust API

Easy to integrate with external apps

Flexible, easy to adapt

Ensure consistency

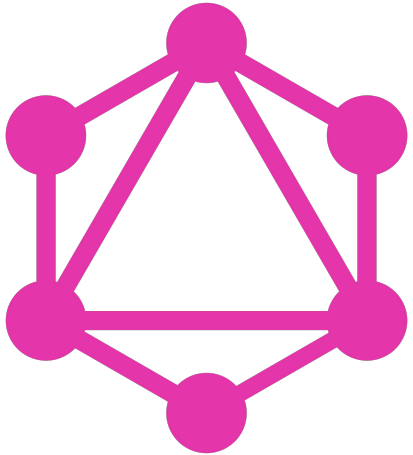


GraphQL is...

... A query language, transport-agnostic
but typically served over HTTP.

... A specification for client-server communication:
It doesn't dictate which language to use, how the data should be stored or which clients to support.

... Based on graph theory: nodes, edges and connections.



REST vs GraphQL

```
/unique_identities/<uuid>/identities

/unique_identities/<uuid>/profile

/unique_identities/<uuid>/enrollments

/organizations/<org_name>/domains
```

```
query {
    unique_identities(uuid:"<uuid>") {
        identities {
            uid
        }
        profile {
            email
            gender
        }
        enrollments {
            organization
            end_date
        }
        domains {
            domain_name
        }
    }
}
```

Bitergia

Comparing approaches: REST

Convention between server and client

Overfetching / Underfetching

API Documentation is not tied to development

Multiple requests per view

Comparing approaches: GraphQL

Strongly typed language

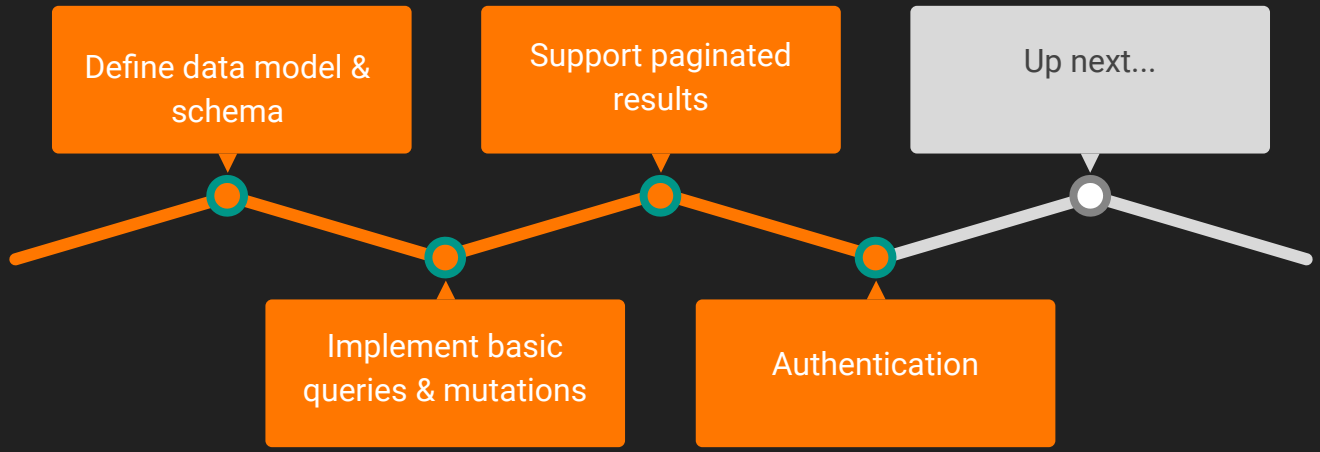
The client defines what it receives

The server only sends what is needed

One single request per view

Summarizing ...

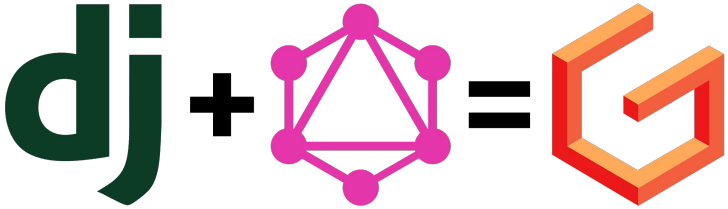
Implementing process



Implementation: Graphene-Django

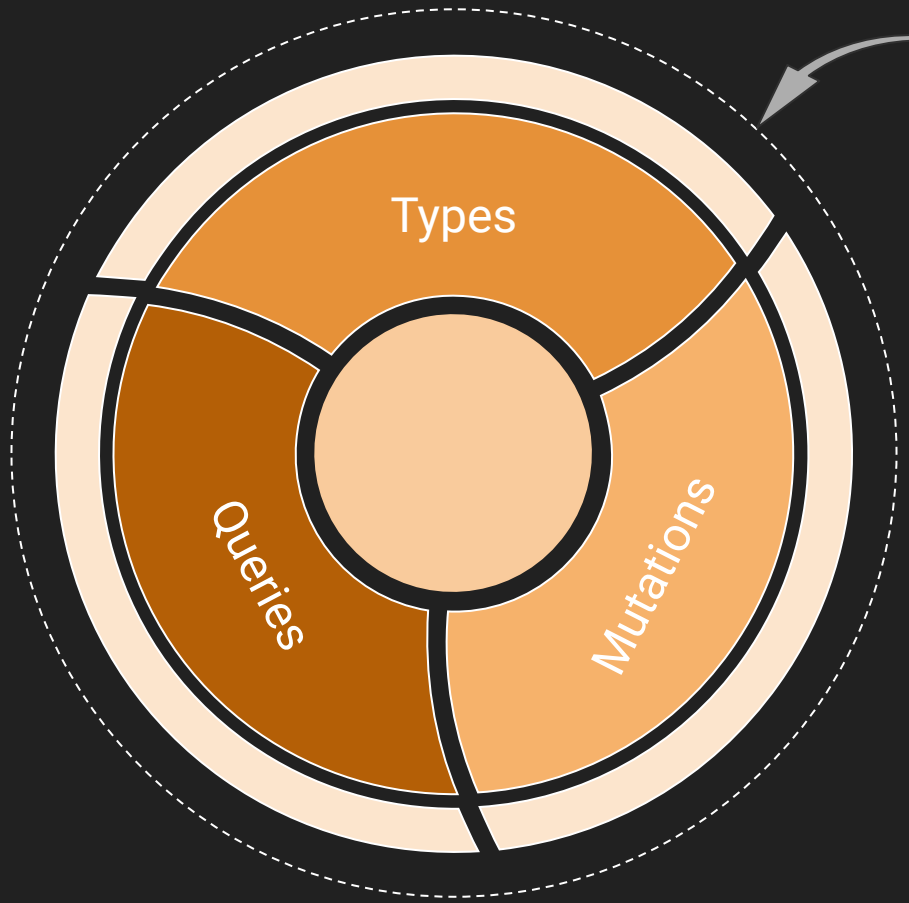
Graphene-Django is built on top of Graphene.

It provides some additional abstractions that help to add GraphQL functionality to your Django project.



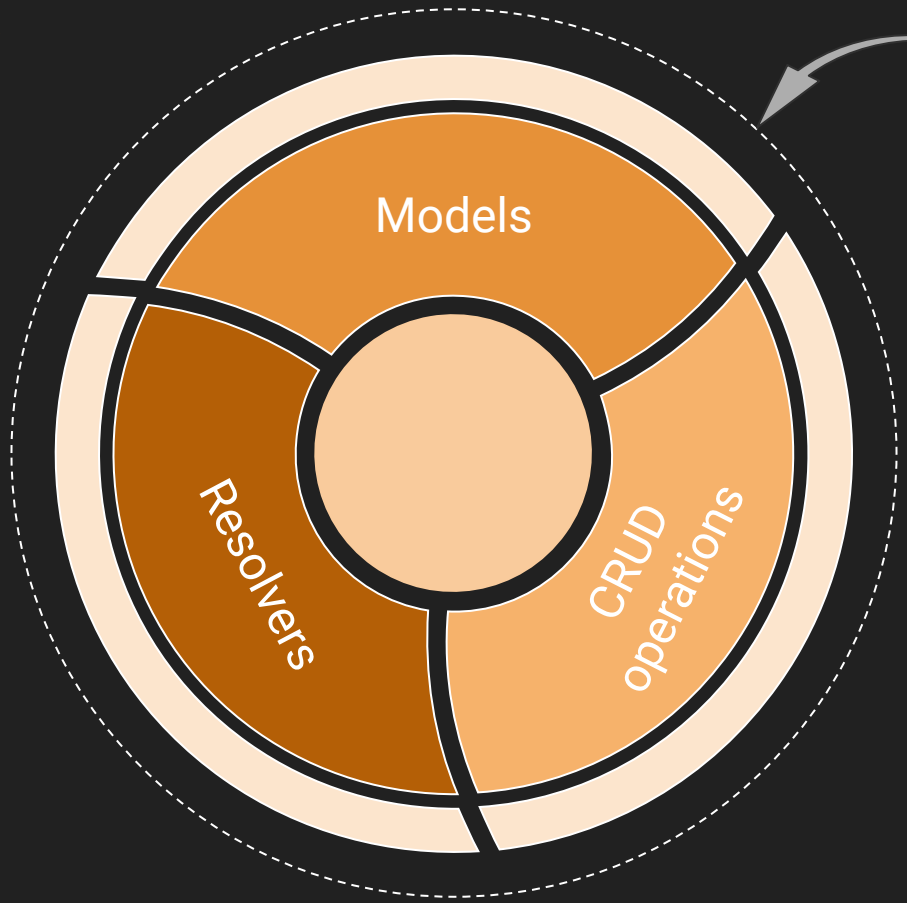
Picture credit: Snippedia

Schema



GraphQL
Schema

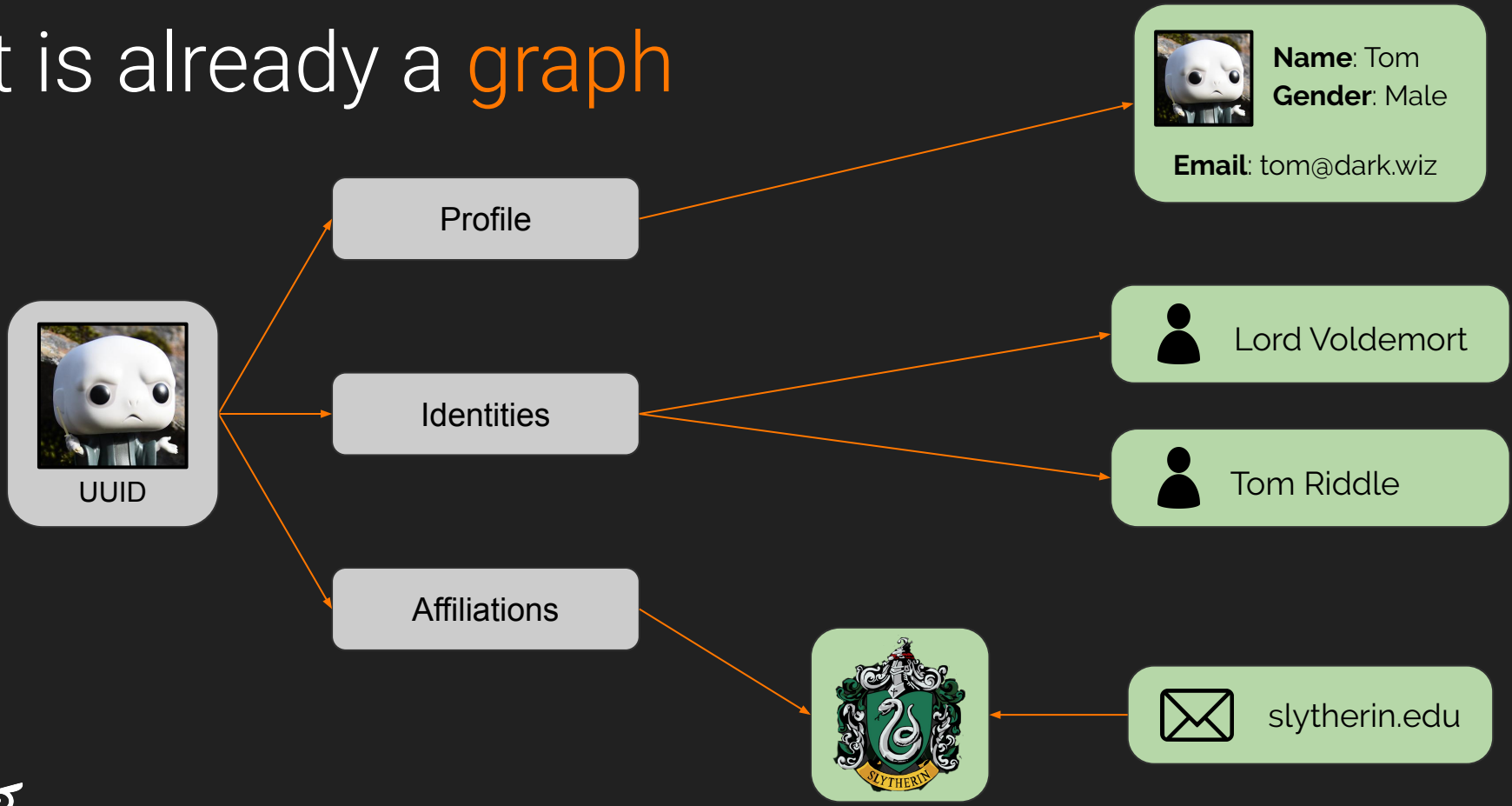
Schema.py



GraphQL Schema:
Graphene-Django

Bitergia

It is already a graph



Bitergia

share this slide! @mghfdez

(Basic) Recipe for building queries

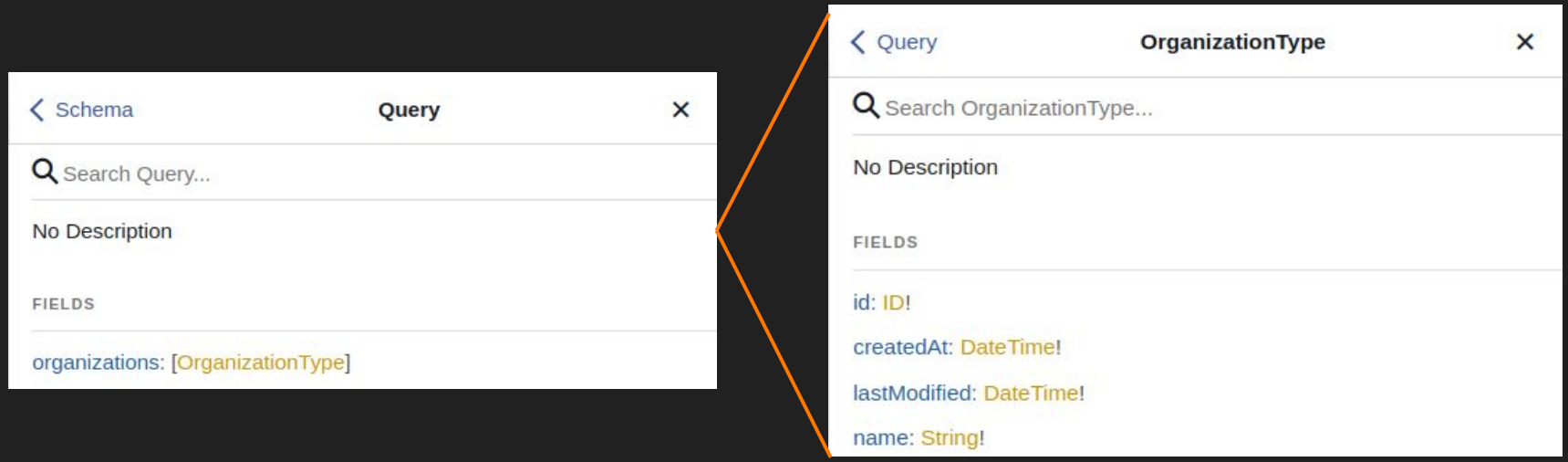
```python
class Organization(EntityBase):
    name = CharField(max_length=MAX_SIZE)

    class Meta:
        db_table = 'organizations'
        unique_together = ('name',)

    def __str__(self):
        return self.name
```

models.py

```python
class OrganizationType(DjangoObjectType):
    class Meta:
        model = Organization


class SortingHatQuery:

    organizations = graphene.List(OrganizationType)

    def resolve_organizations(self, info, **kwargs):
        return Organization.objects.order_by('name')
```

schema.py

Documentation is already updated!



Bitergia

(Basic) Recipe for building mutations

```python
class AddOrganization(graphene.Mutation):
    class Arguments:
        name = graphene.String()

    organization = graphene.Field(lambda: OrganizationType)

    def mutate(self, info, name):
        org = add_organization(name)

        return AddOrganization(
            organization=org
        )
```

```python
class SortingHatMutation(graphene.ObjectType):
    add_organization = AddOrganization.Field()
```

schema.py

Bitergia

(Basic) Recipe for building mutations

```python
@django.db.transaction.atomic
def add_organization(name):

    try:
        org = add_organization_db(name=name)
    except ValueError as e:
        raise InvalidValueError(msg=str(e))
    except AlreadyExistsError as exc:
        raise exc

    return org
```

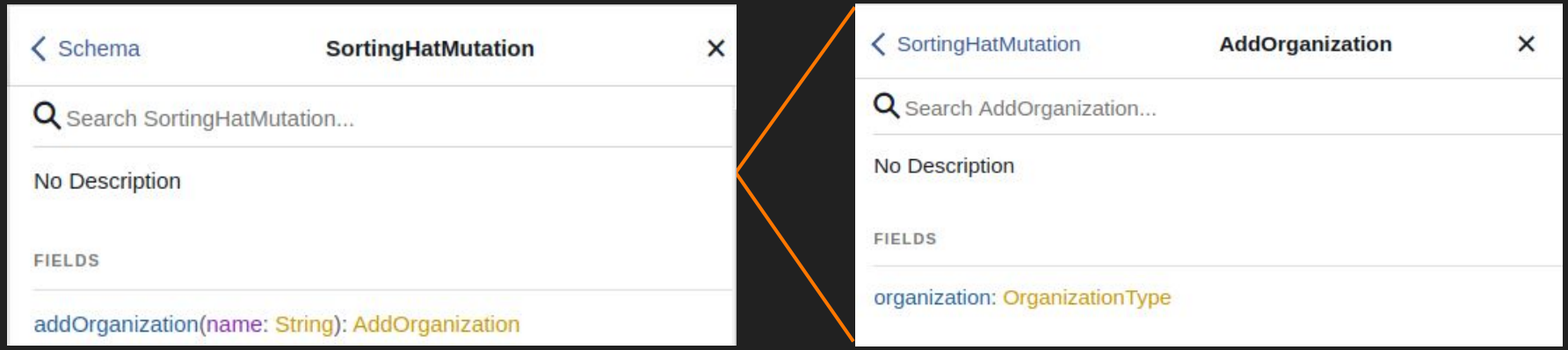
api.py

```python
def add_organization(name):

    validate_field('name', name)
    organization = Organization(name=name)

    try:
        organization.save()
    except django.db.utils.IntegrityError as exc:
        _handle_integrity_error(Organization, exc)

    return organization
```

db.py

Bitergia

Documentation is already updated… again!



Bitergia

share this slide! @mghfdez

About pagination

How are we getting the cursor?

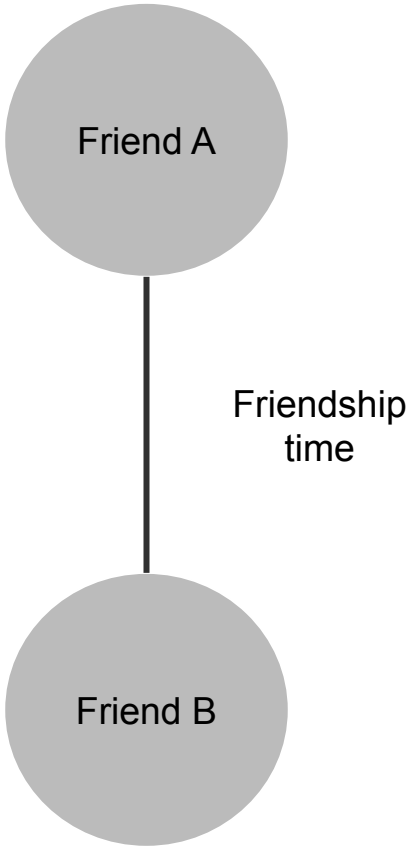
It is a property of the connection,
not of the object.

```
identities(first:2 offset:2)

identities(first:2 after:$uuid)

identities(first:2 after:$uuidCursor)
```

Edges and connections

Information that is specific to the edge, rather than to one of the objects.

There are specifications like Relay



Implementing pagination

We are taking our own approach without reinventing the wheel

It is a hybrid approach based on offsets and limits, using *Paginator* Django objects

Also benefiting from edges & connections

Query

```
{
  organizations(
    page: 1
    pageSize: 3
  ){
    entities{
      name
    }
    pageInfo{
      page
      pageSize
      numPages
      hasNext
      hasPrev
      startIndex
      endIndex
      totalResults
    }
  }
}
```

QUERY VARIABLES

Result


```
{
  "data": {
    "organizations": {
      "entities": [
        {
          "name": "Test"
        },
        {
          "name": "Test 2"
        },
        {
          "name": "Test 3"
        }
      ],
      "pageInfo": {
        "page": 1,
        "pageSize": 3,
        "numPages": 2,
        "hasNext": true,
        "hasPrev": false,
        "startIndex": 1,
        "endIndex": 3,
        "totalResults": 5
      }
    }
  }
}
```


Bitergia

```
{
  organizations(
    page: 1
    pageSize: 3
  ){
    entities{
      name
    }
    pageInfo{
      page
      pageSize
      numPages
      hasNext
      hasPrev
      startIndex
      endIndex
      totalResults
    }
  }
}
```

```
"pageInfo": {
  "page": 1,
  "pageSize": 3,
  "numPages": 2,
  "hasNext": true,
  "hasPrev": false,
  "startIndex": 1,
  "endIndex": 3,
  "totalResults": 5
}
```

Django objects

Query results

Pagination info

```python
class AbstractPaginatedType(graphene.ObjectType):

    @classmethod
    def create_paginated_result(cls, query, page=1,
page_size=DEFAULT_SIZE):
        paginator = Paginator(query, page_size)
        result = paginator.page(page)

        entities = result.object_list

        page_info = PaginationType(
            page=result.number,
            page_size=page_size,
            num_pages=paginator.num_pages,
            has_next=result.has_next(),
            has_prev=result.has_previous(),
            start_index=result.start_index(),
            end_index=result.end_index(),
            total_results=len(query)
        )

        return cls(entities=entities, page_info=page_info)
```

Bitergia

share this slide! @mghfdez

Returning paginated results

```
class OrganizationPaginatedType(AbstractPaginatedType):
  entities = graphene.List(OrganizationType)
  page_info = graphene.Field(PaginationType)


class SortingHatQuery:

  def resolve_organizations(...)

    (...)

    return OrganizationPaginatedType.create_paginated_result(query,
                                                             page,
                                                             page_size=page_size)
```

Bitergia

Authenticated queries

It is based on JSON Web Tokens (JWT)

An existing user must generate a token which has to be included in the Authorization header with the HTTP request

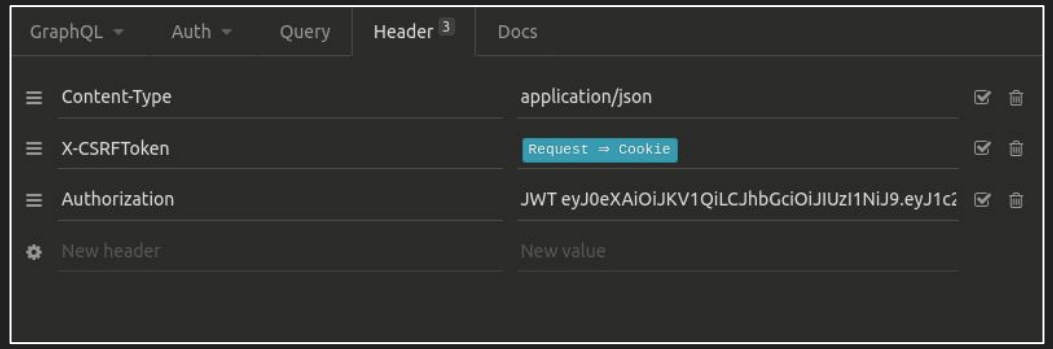
This token is generated using a mutation which comes defined by the *graphene-jwt* module

Testing authentication

Use an application capable of setting up headers to the HTTP requests

Heads-up!

Configuring the Django CSRF token properly was not trivial



Insomnia app

Testing authentication

```
from django.test import RequestFactory

def setUp(self):

    self.user = get_user_model().objects.create(username='test')
    self.context_value = RequestFactory().get(GRAPHQL_ENDPOINT)
    self.context_value.user = self.user


def test_add_organization(self):

    client = graphene.test.Client(schema)
    executed = client.execute(self.SH_ADD_ORG, context_value=self.context_value)
```

Bitergia

Bonus: filtering

```
class OrganizationFilterType(graphene.InputObjectType):
  name = graphene.String(required=False)


class SortingHatQuery:

  organizations = graphene.Field(
      OrganizationPaginatedType,
      page_size=graphene.Int(),
      page=graphene.Int(),
      filters=OrganizationFilterType(required=False)
  )

  def resolve_organizations(...):
        # Modified resolver
```

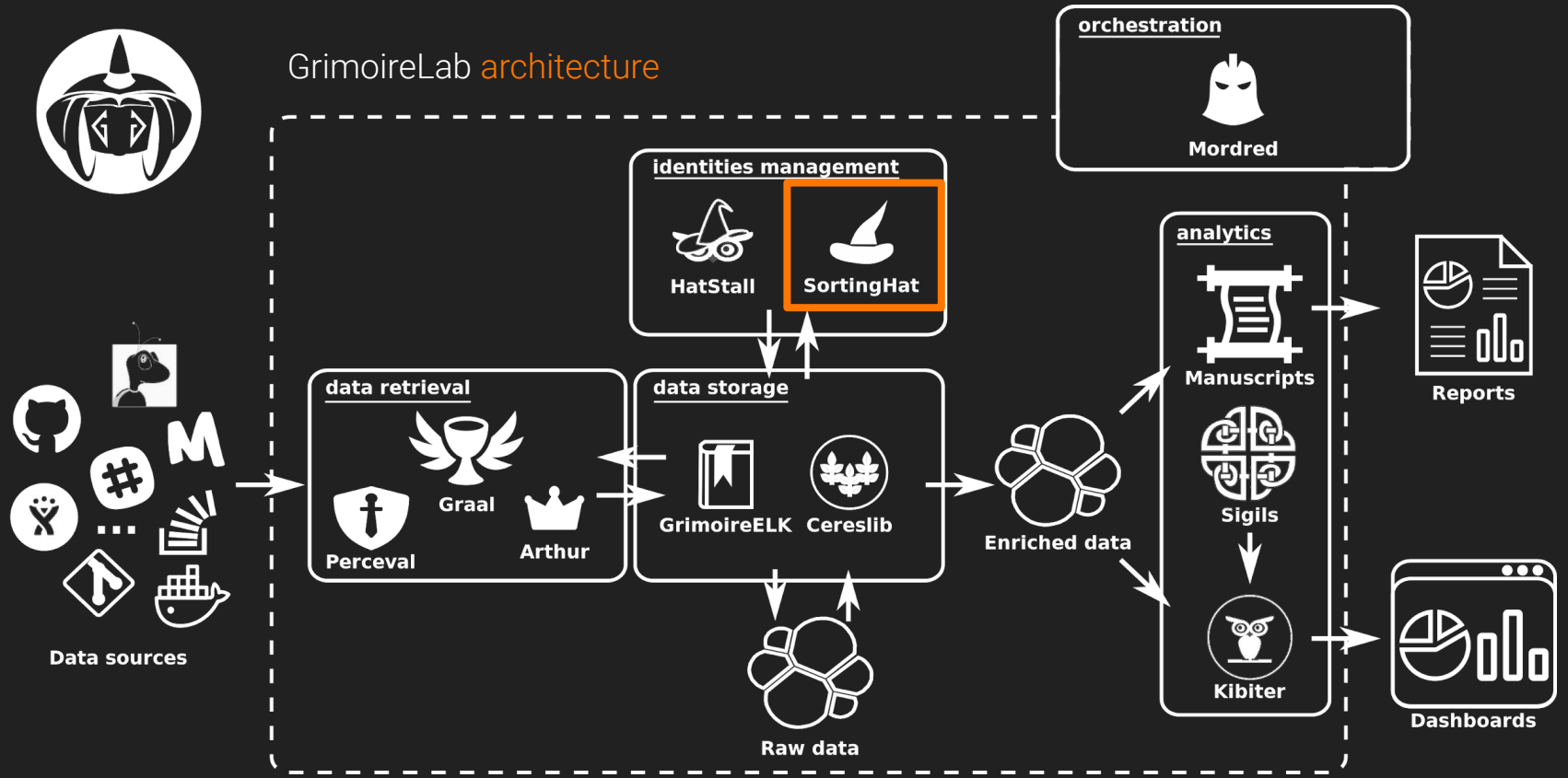
Bitergia

(some) Future work

Implementing a command line & web Client

Limiting nested queries

Feedback is welcome!



Bitergia

share this slide! @mghfdez

Let’s go for some questions

Twitter @mghfdez

Email mafesan@bitergia.com

GitHub mafesan

FLOSS enthusiast & Data nerd

Software Developer @ Bitergia

Contributing to CHAOSS-GrimoireLab project

Bitergia

share this slide! @mghfdez