GraphQL and Django

Arianne Dee - 7Geese

OKRs

- Objectives (Goals)
- Key Results (How you are measuring your goals)

- Learn something new every day (Objective)
- Spend at least 10 mins every day reading articles, watching videos, or following tutorials (Key Result 1)
- Attend 3 technical meet ups (Key Result 2)

The Django way

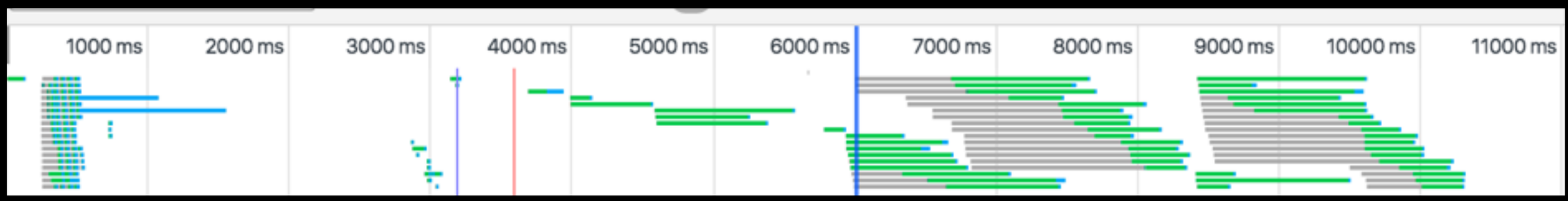
```
# in views.py

def index(request, template_name='main/main.html'):
    user = UserProfile.objects.get(id=request.user.id)
    objectives = Objective.objects.filter(creator=user)
    context = {‘user’:user, ‘objectives’: objectives}

    return render_to_response(template_name, context)
```

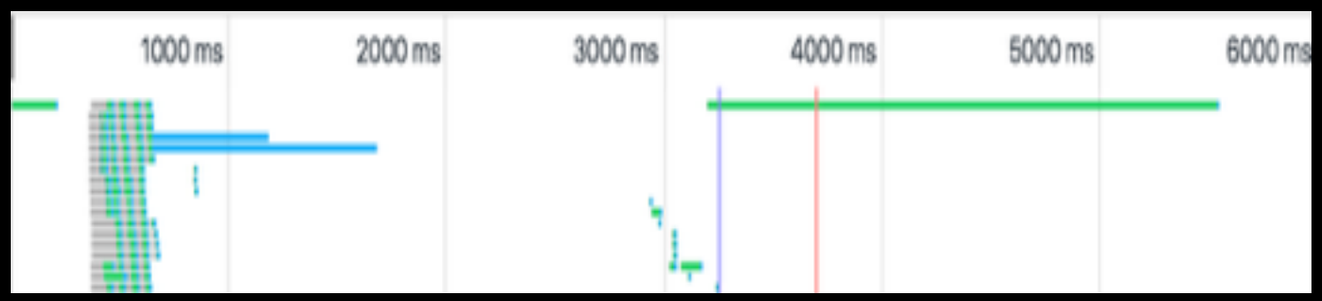
The REST API way

- /api/userprofiles/me/
- /api/userprofiles/{pk}/objectives/
- /api/userprofiles/{pk}/objectives/{pk}/keyresults/
- /api/objectives/?krs=True



The GraphQL way

- Created at Facebook
  - Has been used in production since 2012
    - Data fetching for React (through Relay)
- Made public in Jan 2015



How?

- **Server**: publish a schema that describes the data that's available

- **Client**: call the endpoint with a JSON-like query in either the POST data or GET query param

You also get

- Types
- Fragments (partials)
- Aliases
- Variables
- Enums
- Comments
- Mutations (put, patch)
- Subscriptions

Graphene

- Python library for defining schemas and parsing and processing queries
- http://graphene-python.org/
- Integrations for Django and SQLAlchemy
- Create schema that defines nodes
- pip install graphene-django

Define nodes

```
from graphene_django import DjangoObjectType

class KeyResultNode(DjangoObjectType):
    class Meta:
        model = KeyResult


class ObjectiveNode(DjangoObjectType):

    class Meta:
        model = Objective
```

Define queries

```
import graphene

class Query(graphene.ObjectType):
    objectives = graphene.List(ObjectiveNode)

    def resolve_objectives(self):
        return Objective.objects.all()


schema = graphene.Schema(
    query=Query
)
```

Hook it up

```
# in settings.py

INSTALLED_APPS += [
    'graphene_django',
]

# Where your Graphene schema lives
GRAPHENE = {
    'SCHEMA': ‘app.objectives.schema.schema’
}
```

Hook it up

```
# in urls.py

from graphene_django.views import GraphQLView

urlpatterns += [
    url(r'^graphql', GraphQLView.as_view(graphiql=True)),
]
```

That’s great, but can you actually use it in the real world?

Real world considerations

- query validation
- custom fields
- filters
- pagination
- authorization
- interfaces and inheritance

Custom fields

```
import graphene
from django.contrib.auth.models import User


class UserNode(DjangoObjectType):
    say_hello = graphene.String(
        in_french=graphene.Argument(graphene.Boolean()))

    def resolve_say_hello(self, args, context, info):
        in_french = args.get('in_french', False)

        if in_french:
            return 'Bonjour {}!'.format(self.first_name)
        else:
            return 'Hello {}!'.format(self.first_name)

    class Meta:
        model = User
```

Authorization

Apply authorization on the resolvers

```
class Query(ObjectType):
    objectives = graphene.List(ObjectiveNode)

    def resolve_objectives(self, args, context, info):
        if not context.user.is_authenticated():
            return Objectives.objects.none()
        else:
            return context.user.objectives.all()
```

But no clear way to apply authorization on every resolver

Filters and pagination

- Need Relay - comes with Graphene

```
{
  objectives (name_contains: ‘awesome’) {
    count
    edges {
      node {
        name
        progress
      }
      pageInfo {
        endCursor
        hasNextPage
      }
    }
  }
}
```

Filters

pip install django-filters

```
# import django_filters

class ObjectiveFilter(django_filters.FilterSet):
    name = django_filters.CharFilter(lookup_type='icontains')

    class Meta:
        model = Objective
        fields = ['name']


class Query(graphene.ObjectType):
    objectives = DjangoFilterConnectionField(ObjectiveNode,
filterset_class=ObjectiveFilter)
```

Can’t currently use django-filters without relay

Real world concerns

- Denial of service attacks
- Performance
- Immature tooling

Security - DoS

- An attacker could construct a query which is very expensive to execute (e.g. lots of nested relations)

- Could create explicit list of allowed queries, each with their own URL

- Could perform an initial cost analysis of a query before executing it, and set a cost threshold

Performance

- No select_related, prefetch_related when performing GraphQL queries
- Queries with connections will perform lots of db queries
- You can override the resolvers to improve performance where necessary

Immature tooling

- Graphene has been public since Sept 2015. It's pretty good, but it has some rough edges
- Just released v1.0 on Sept 26, 2016
- It's the only viable GraphQL library (we've found) for Python
- Doesn't currently contain features found in other languages (batch querying, cost analysis)

Some Links

- graphql.org
- Zero to GraphQL (video)
- Intro to GraphQL (blog post)
- Graphene is now production ready (blog post)

Thank you!

Questions/comments:
arianne@7geese.com