GraphQL in the Wild

DjangoCon 2017

Arianne Dee

GraphQL in the Wild

DjangoCon 2017

Arianne Dee

ABOUT ME

Django: 2 years

7Geese: 1.5 years

GraphQL: 1 year

Let’s talk about REST

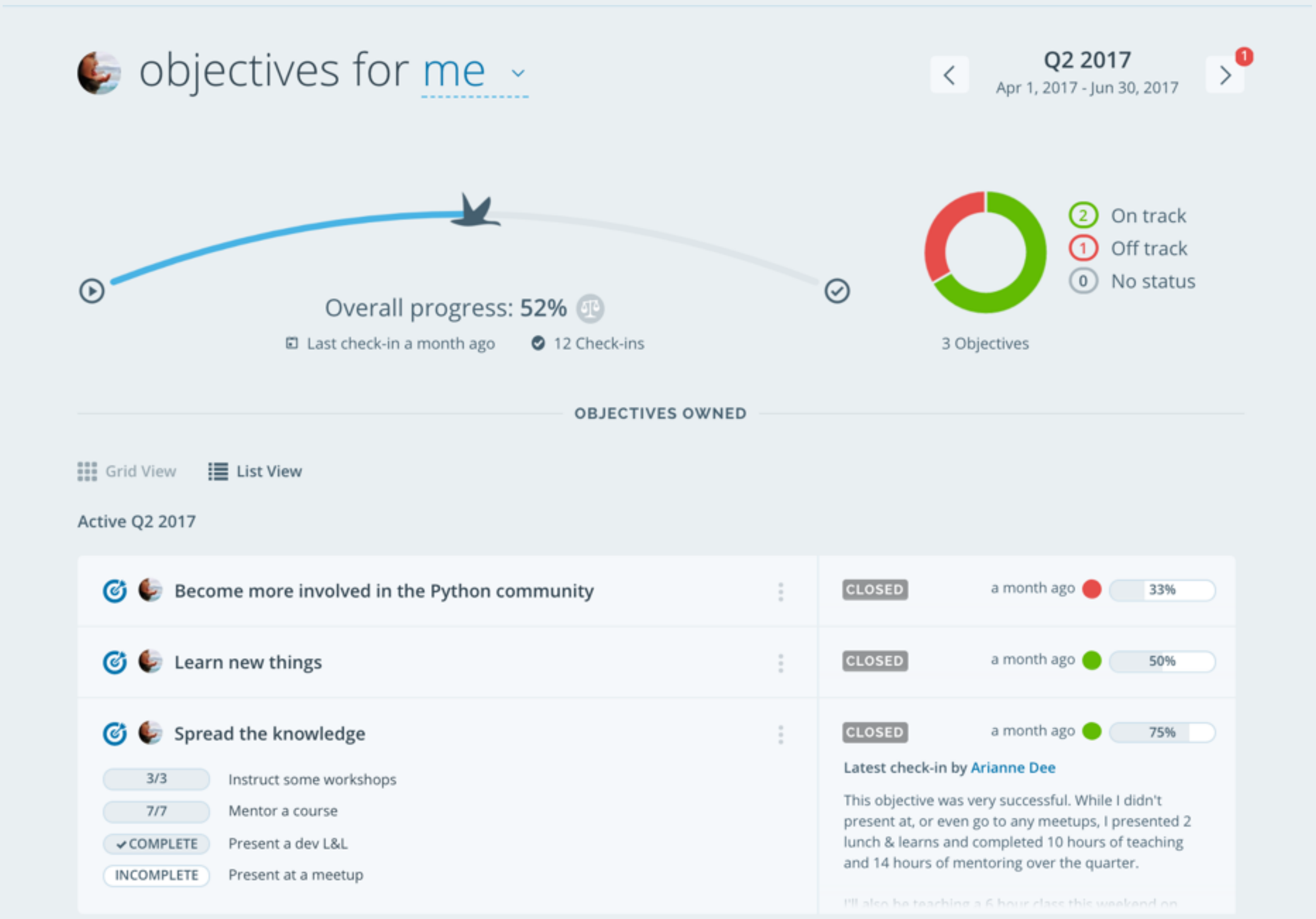
RELATED DATA

- /api/user/{pk}/
- /api/user/{pk}/resource/
- /api/user/{pk}/resource/{pk}/related_resource/

SERIALIZATION

- # User resource fields: 34
- # Important fields: 4
- Excess fields: 30/34 or 88%

Challenge #1:
Dashboards


objectives for me

Q2 2017
Apr 1, 2017 - Jun 30, 2017

Overall progress: **52%**

Last check-in a month ago · 12 Check-ins

OBJECTIVES OWNED

Grid View · List View

Active Q2 2017

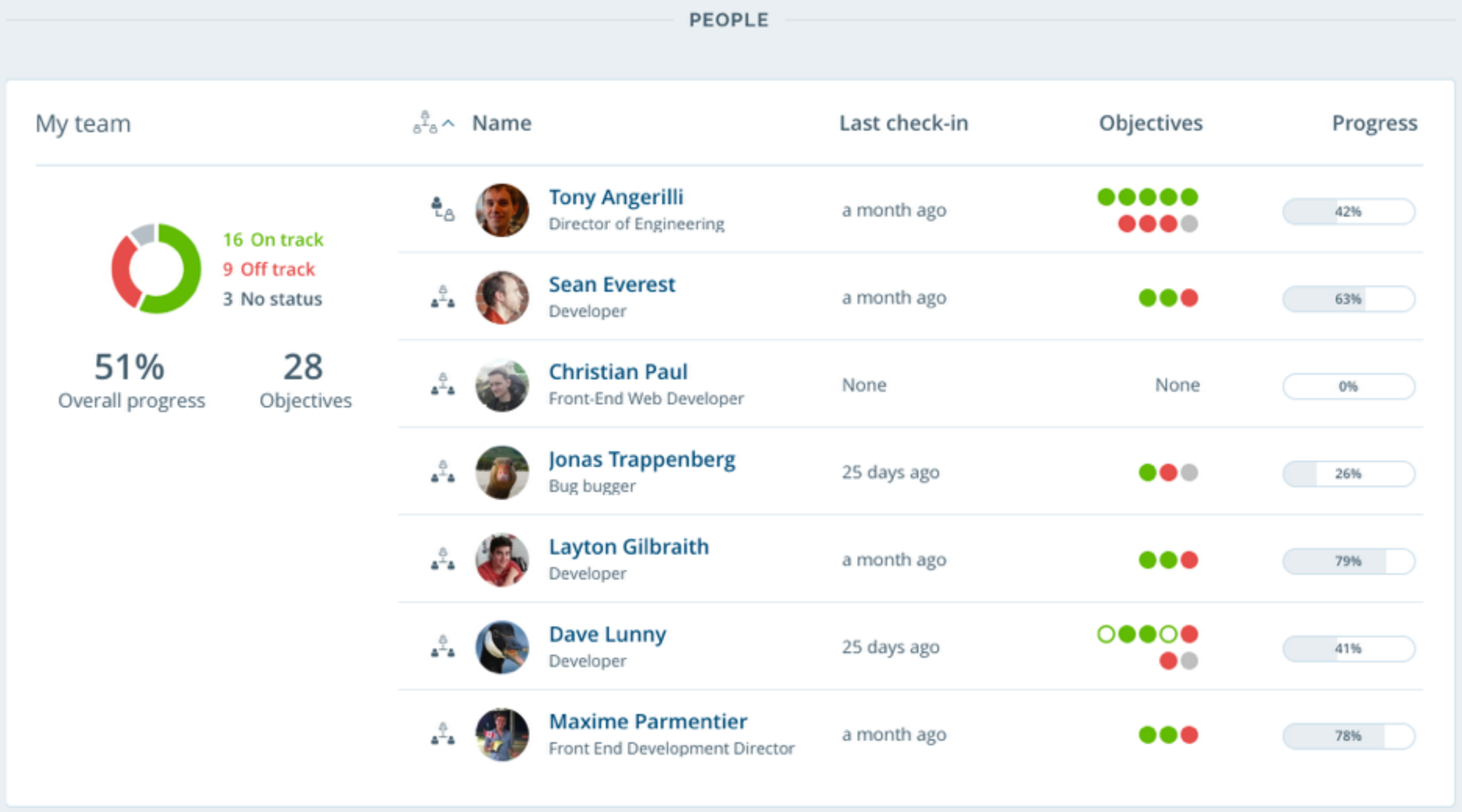
Become more involved in the Python community	CLOSED	a month ago	🟥	33%
Learn new things	CLOSED	a month ago	🟩	50%
Spread the knowledge	CLOSED	a month ago	🟩	75%

3/3	Instruct some workshops
7/7	Mentor a course
✓COMPLETE	Present a dev L&L
INCOMPLETE	Present at a meetup

Latest check-in by Arianne Dee

This objective was very successful. While I didn't present at, or even go to any meetups, I presented 2 lunch & learns and completed 10 hours of teaching and 14 hours of mentoring over the quarter.

I'll also be teaching a 6 hour class this weekend on



PEOPLE

My team

16 On track
9 Off track
3 No status

51% Overall progress

28 Objectives

Name		Last check-in	Objectives	Progress
Tony Angerilli	Director of Engineering	a month ago	🟢🟢🟢🟢🟢 🔴🔴🔴⚪	42%
Sean Everest	Developer	a month ago	🟢🟢🔴	63%
Christian Paul	Front-End Web Developer	None	None	0%
Jonas Trappenberg	Bug bugger	25 days ago	🟢🔴⚪	26%
Layton Gilbraith	Developer	a month ago	🟢🟢🔴	79%
Dave Lunny	Developer	25 days ago	○🟢🟢○🔴 🔴⚪	41%
Maxime Parmentier	Front End Development Director	a month ago	🟢🟢🔴	78%

QUICK SUMMARIES

ORGANIZATION

Launch 7Geese 2.0: Performance Management Reinvented	48%
Enable more organizations to succeed with 7Geese	36%

DEVELOPMENT

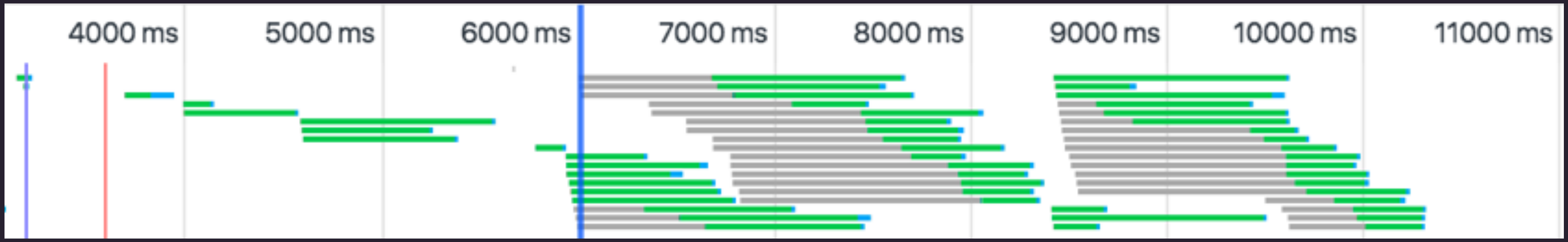
Increase the speed at which we ship new features	24%
Improve app performance	0%
Improve app reliability	56%
Improve our software development skills	74%

RELATED DATA

- /api/**user**/{pk}/
- /api/user/{pk}/**goals**/
- /api/user/{pk}/goals/{pk}/**tasks**/
- /api/user/{pk}/goals/{pk}/**progress_updates**/
- /api/**user**/{pk}/**teams**/
- /api/**user**/{pk}/teams/{pk}/**members**/

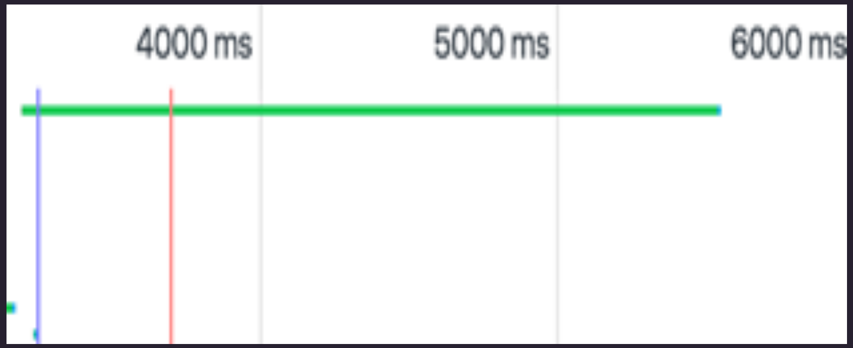
Challenge #1:
Dashboards

REST



10.3s

GraphQL



5.5s

What is GraphQL?

The ability to define precisely the data you want—and only the data you want—is a powerful advantage over the REST API.

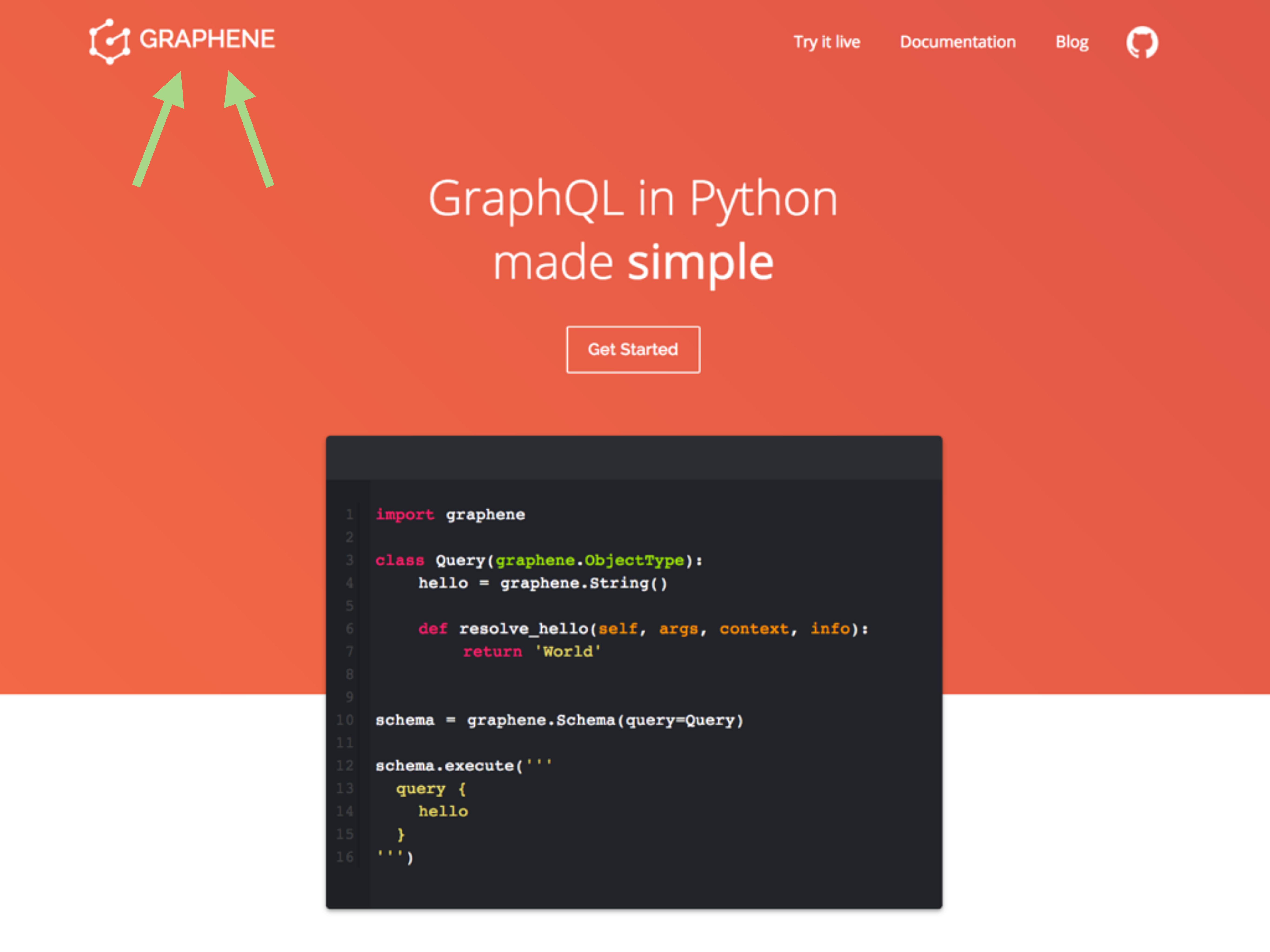
Github

Significant advantages of GraphQL include:

- Getting the data you need and nothing more
- Nested fields
- Strong typing

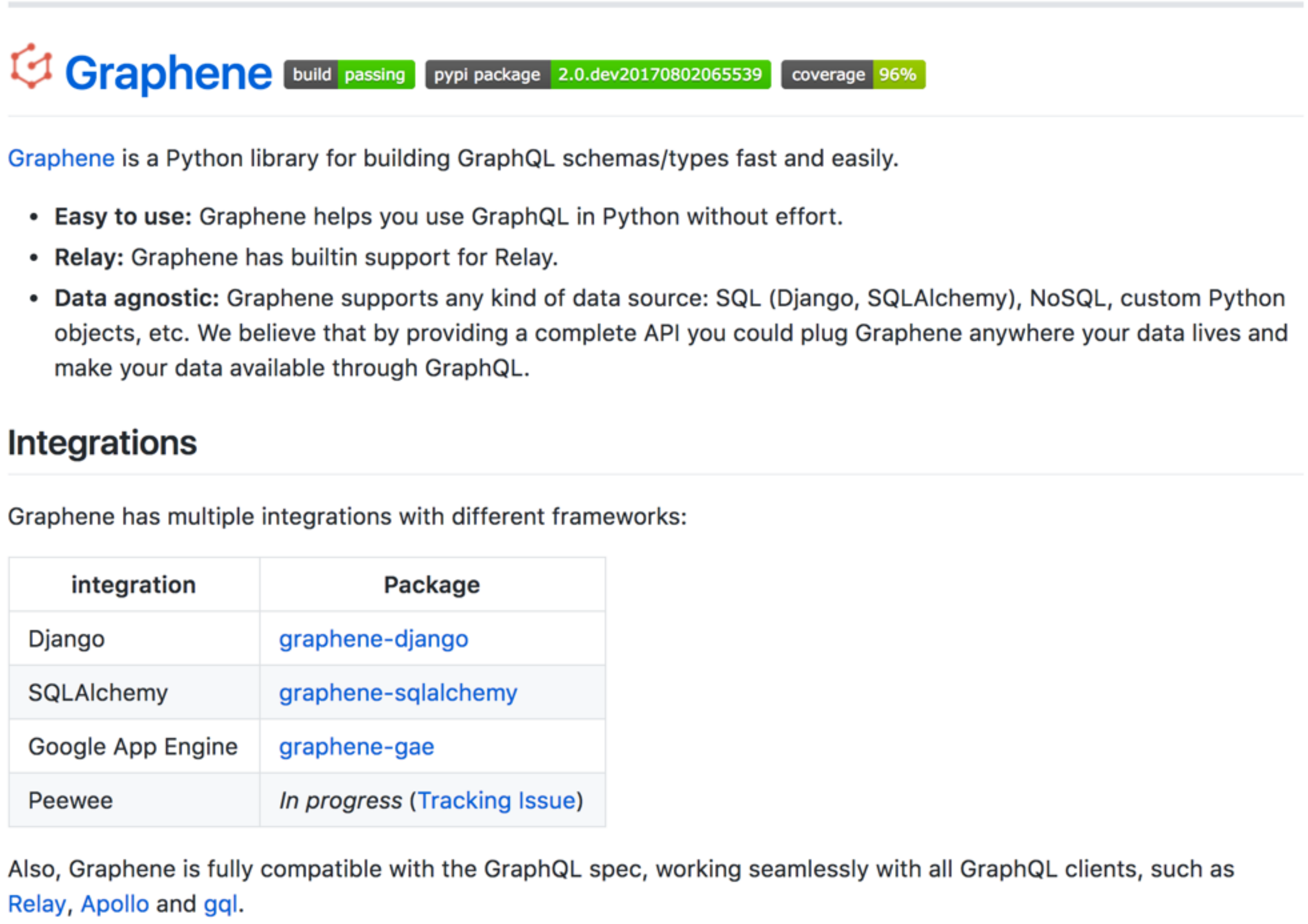
Github

Does it play well with Django?



GraphQL in Python
made **simple**

```python
import graphene

class Query(graphene.ObjectType):
    hello = graphene.String()

    def resolve_hello(self, args, context, info):
        return 'World'


schema = graphene.Schema(query=Query)

schema.execute('''
  query {
    hello
  }
''')
```



Graphene

build passing | pypi package 2.0.dev20170802065539 | coverage 96%

Graphene is a Python library for building GraphQL schemas/types fast and easily.

- **Easy to use:** Graphene helps you use GraphQL in Python without effort.
- **Relay:** Graphene has builtin support for Relay.
- **Data agnostic:** Graphene supports any kind of data source: SQL (Django, SQLAlchemy), NoSQL, custom Python objects, etc. We believe that by providing a complete API you could plug Graphene anywhere your data lives and make your data available through GraphQL.

Integrations

Graphene has multiple integrations with different frameworks:

integration	Package
Django	graphene-django
SQLAlchemy	graphene-sqlalchemy
Google App Engine	graphene-gae
Peewee	*In progress* (Tracking Issue)

Also, Graphene is fully compatible with the GraphQL spec, working seamlessly with all GraphQL clients, such as Relay, Apollo and gql.

DJANGO + GRAPHQL = GRAPHENE

- Setup

SETUP

- `pip install graphene_django`

- `INSTALLED_APPS += [‘graphene_django’,]`

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

DJANGO + GRAPHQL = GRAPHENE

- Setup
- **Define queries (GET)**

DEFINE NODES - BASIC

```python
class TaskNode(DjangoObjectType):
  class Meta:
    model = Task

class GoalNode(DjangoObjectType):
  class Meta:
    model = Goal
```

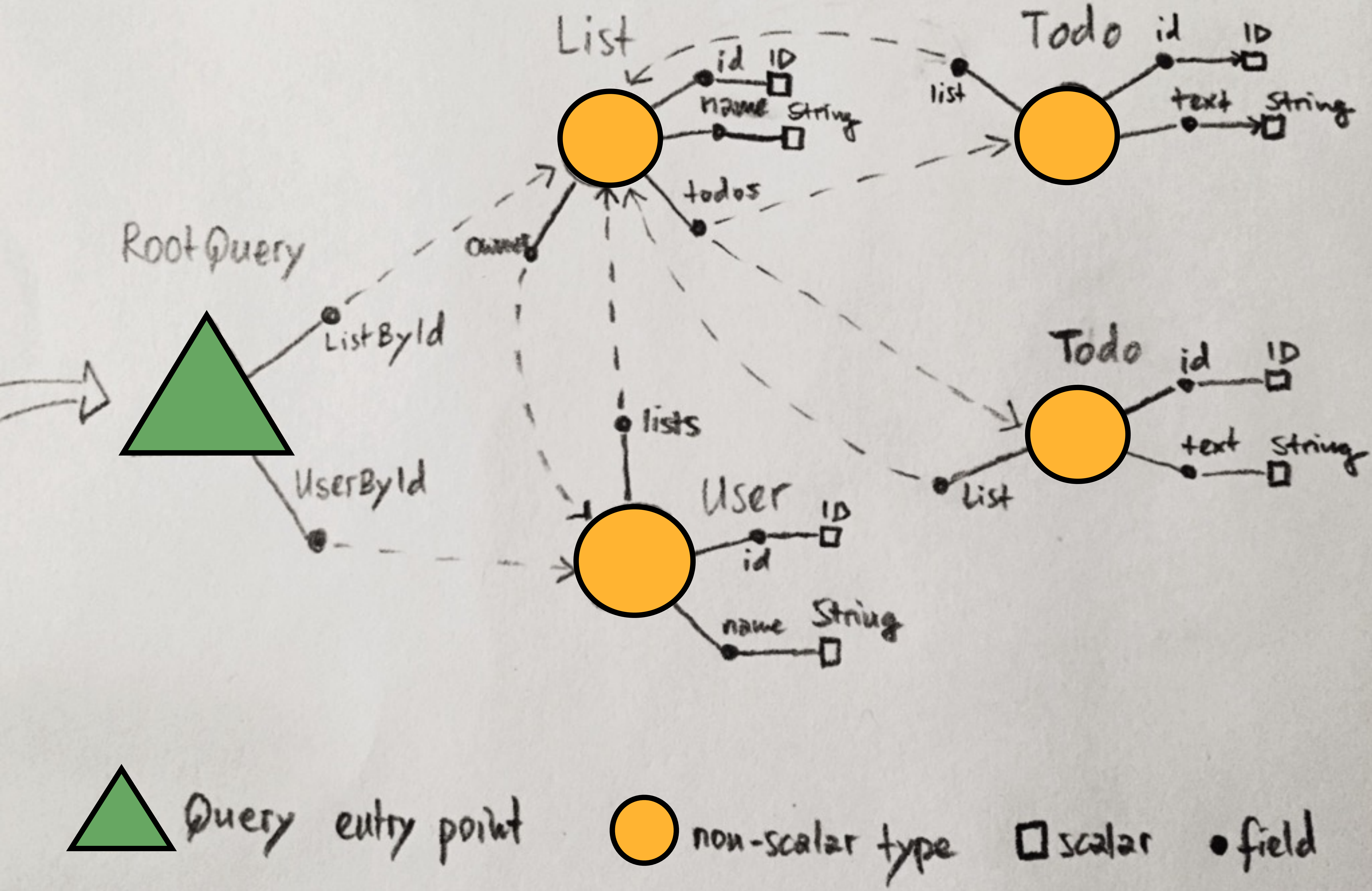
DEFINE QUERY + SCHEMA

```
class Query(ObjectType):
  goals = List(GoalNode)

  def resolve_goals(self):
    return Goal.objects.all()

schema = Schema(
  query=Query
)
```

Root query

Resolver

Schema


List
id
ID
name
String
todos
owner
Todo
id
ID
list
text
String
RootQuery
ListById
UserById
lists
User
id
ID
name
String
Todo
id
ID
text
string
List
Query entry point
non-scalar type
scalar
field

DEFINE NODES - CUSTOM

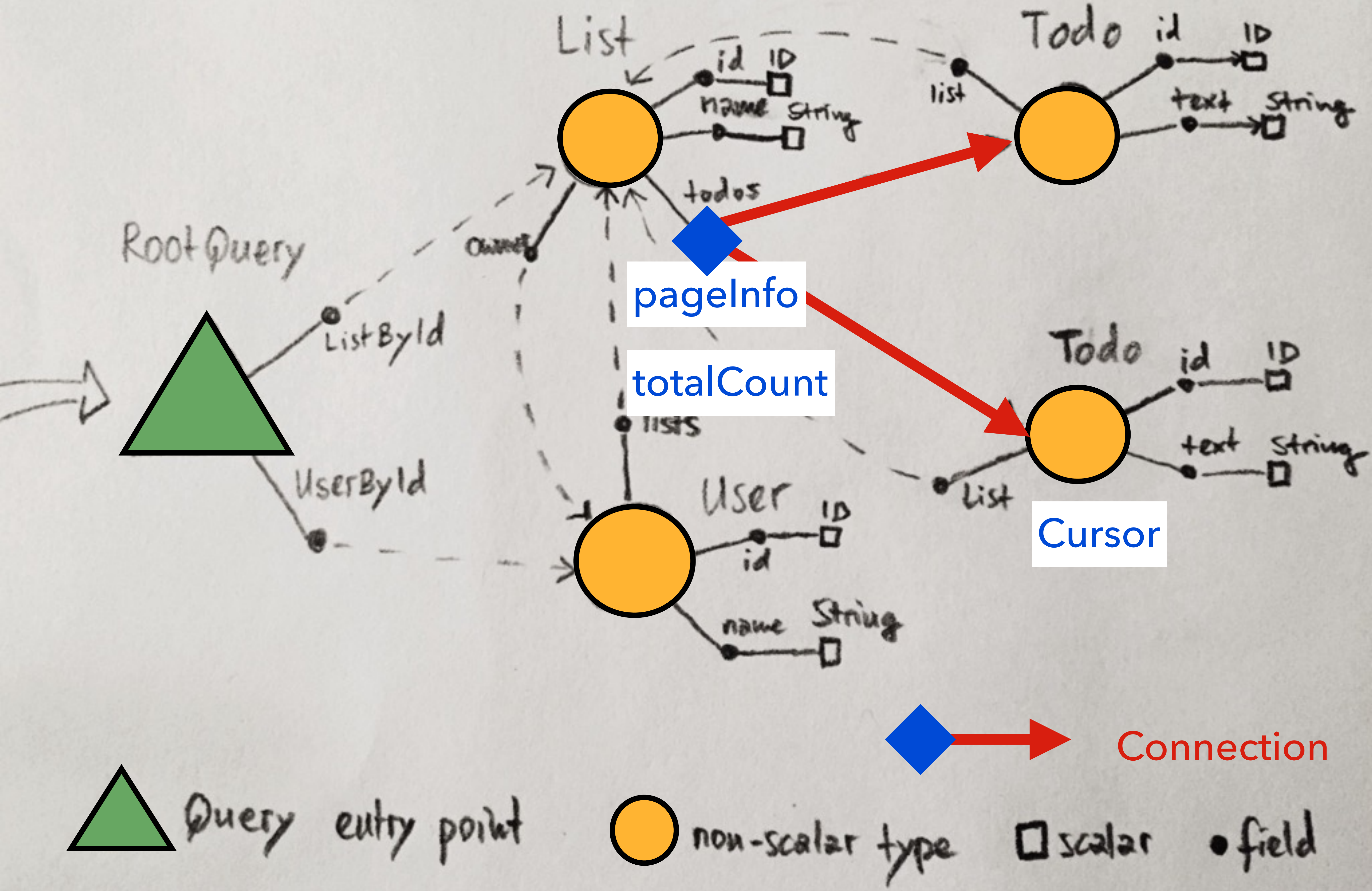
```
class UserNode(DjangoObjectType):
  full_name = String()

  def resolve_full_name(self, args, context, info):
    return u'{} {}’.format(
        self.first_name, self.last_name)

  class Meta:
    model = User
```

DJANGO + GRAPHQL = GRAPHENE

- Setup
- Define queries (GET)
- **Add filters & pagination**

RELAY

```
{
  goals {
    name
    progress
  }
}
```


List
id
ID
name
String
todos
list
Todo
id
ID
text
String
Root Query
ListById
owner
pageInfo
totalCount
lists
Todo
id
ID
text
String
list
Cursor
UserById
User
id
ID
name
String
Connection
Query entry point
non-scalar type
scalar
field

RELAY

```
{
  goals {
    name
    progress
  }
}
```

→

```
{
  goals (first: 5, after: “cursor”) {
    edges {
      node {
        name
        progress
      }
      cursor
      pageInfo {
        endCursor
        hasNextPage
      }
    }
  }
}
```

PAGINATION

```
class GoalNode(DjangoObjectType):
  class Meta:
    interfaces = (relay.Node,)
    model = Goal
```

DJANGO-FILTERS

```
class Query(ObjectType):
  goals = List(GoalNode)

  def resolve_goals(self):
    return Goal.objects.all()
```

Becomes

```
class Query(ObjectType):
  goals = DjangoFilterConnectionField(GoalNode,
    filterset_class=GoalFilter)
```

DJANGO + GRAPHQL = GRAPHENE

- Setup
- Define queries (GET)
- Add filters & pagination
- Documentation

DOCUMENTATION

```
class GoalNode(DjangoObjectType):
  progress = Float(
    description=“The average task progress”)

  def resolve_progress(self, args, context, info):
    return self.calculate_progress()

  class Meta:
   model = Goal
```

OTHER FANCY STUFF

- Unions
- Interfaces
- Fragments
- Aliases
- Variables
- Subscriptions

PROS

- Explorable / fun!
- Easy documentation
- More intuitive to implement than DRF

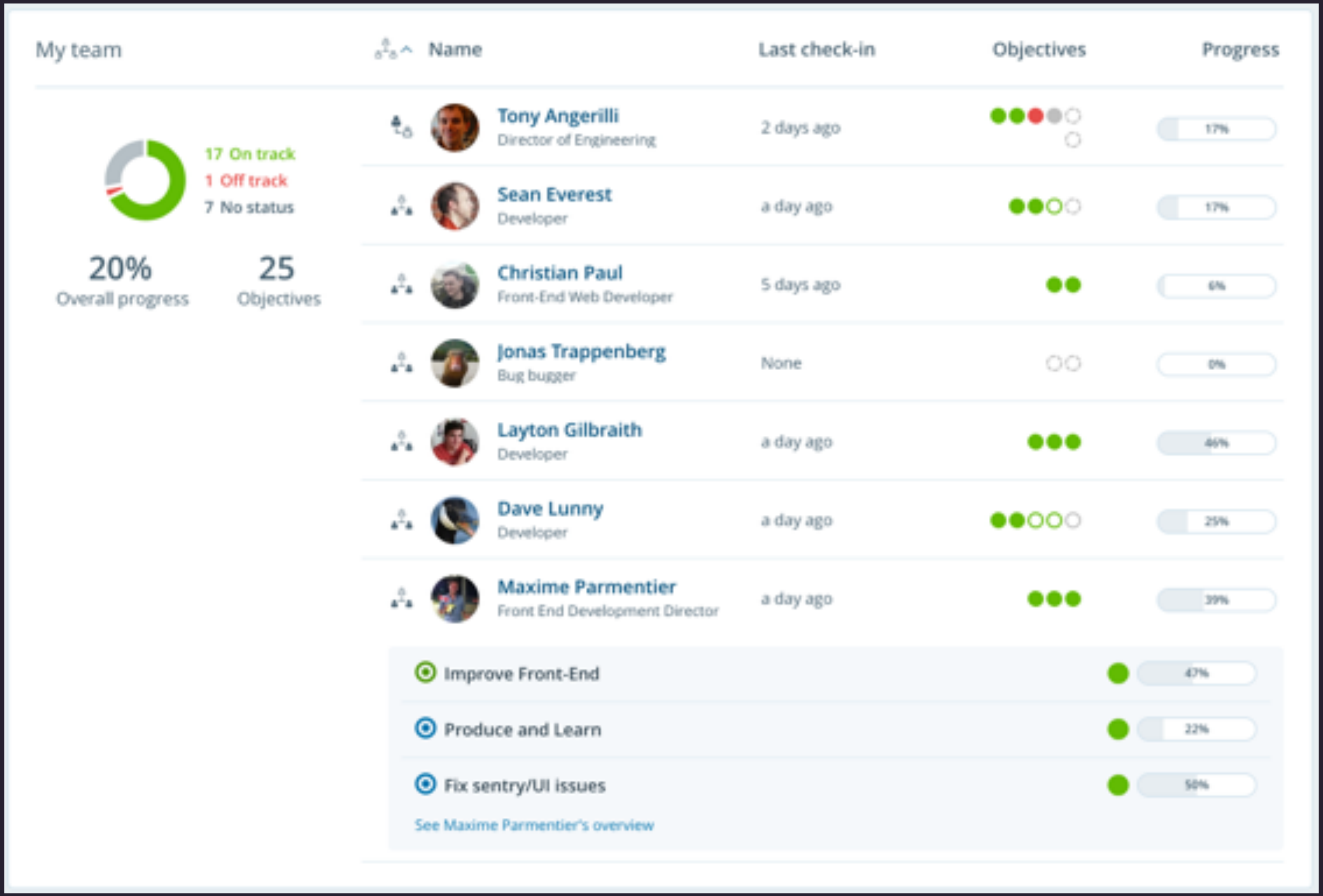
CONSIDERATIONS

- ?

What makes it better than REST?

COMPLEX VIEWS

- Dashboards
- Summaries
- Stats



My team

17 On track
1 Off track
7 No status

20% Overall progress

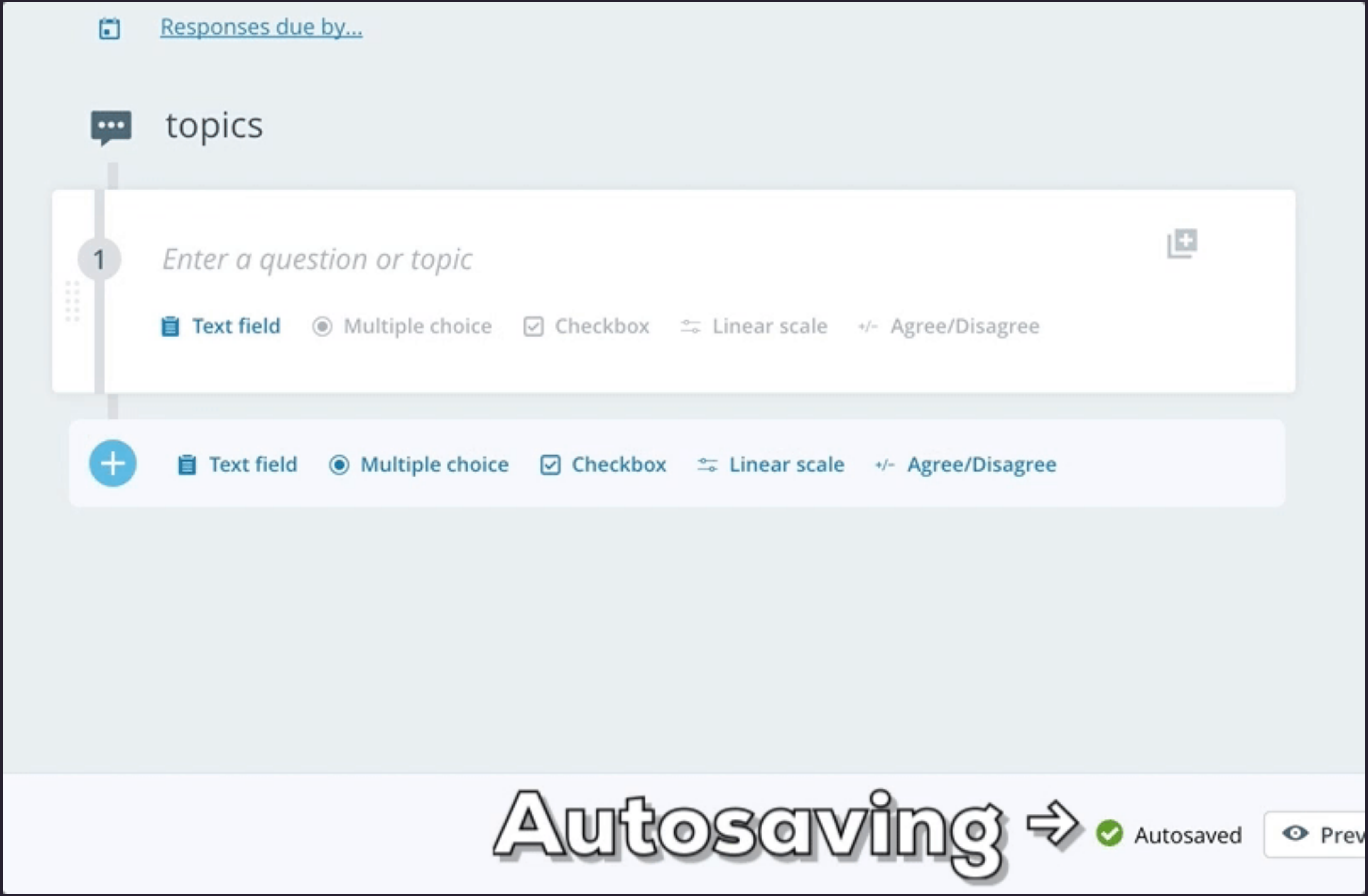
25 Objectives

Name	Last check-in	Objectives	Progress
Tony Angerilli Director of Engineering	2 days ago	●●●●○○	17%
Sean Everest Developer	a day ago	●●○○	17%
Christian Paul Front-End Web Developer	5 days ago	●●	6%
Jonas Trappenberg Bug bugger	None	○○	0%
Layton Gilbraith Developer	a day ago	●●●	46%
Dave Lunny Developer	a day ago	●●○○○	25%
Maxime Parmentier Front End Development Director	a day ago	●●●	39%

Improve Front-End	●	47%
Produce and Learn	●	22%
Fix sentry/UI issues	●	50%

See Maxime Parmentier's overview

Challenge #2:
Builders


Responses due by...

topics

1 Enter a question or topic

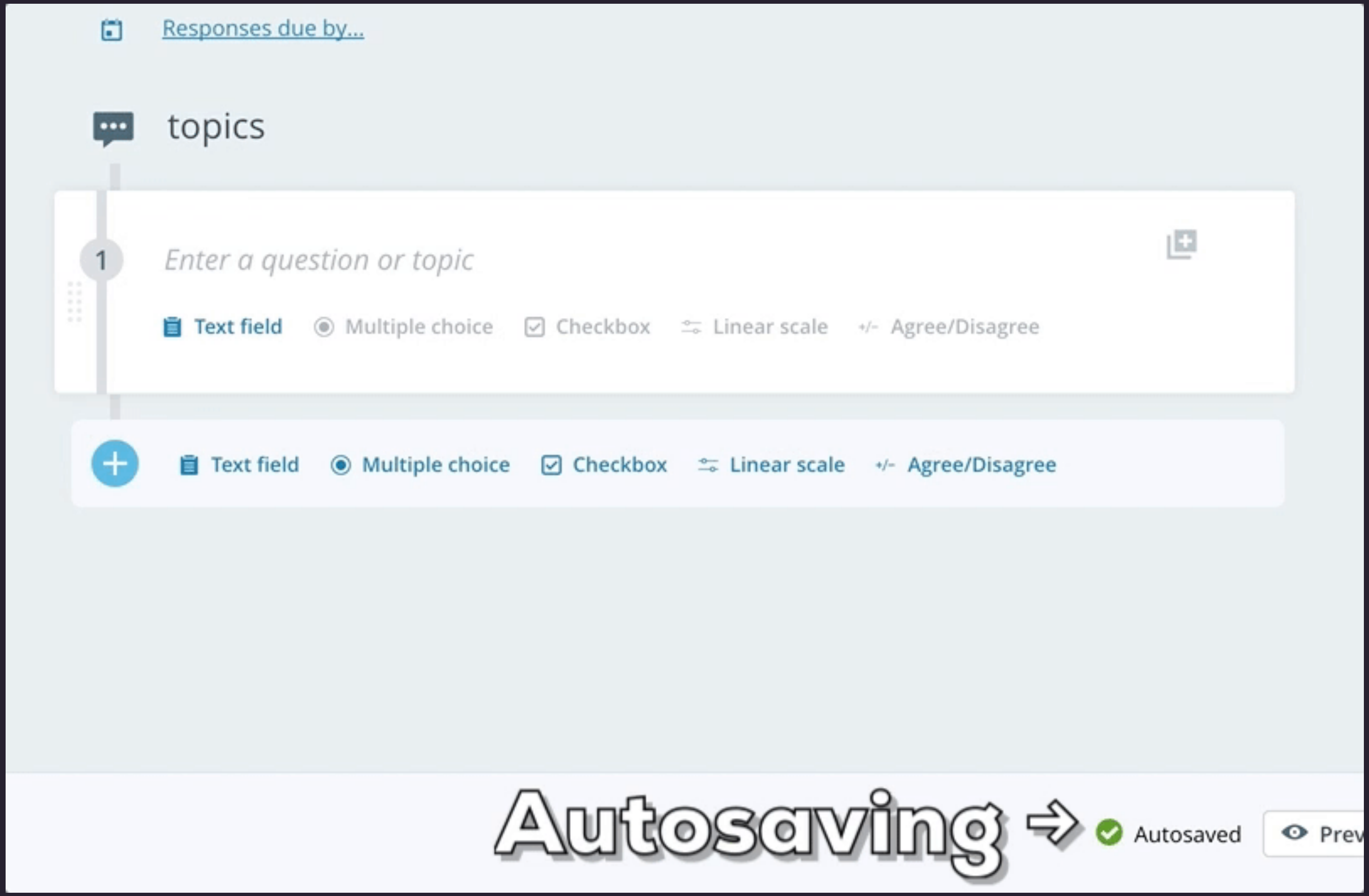
Text field · Multiple choice · Checkbox · Linear scale · Agree/Disagree

Text field · Multiple choice · Checkbox · Linear scale · Agree/Disagree

Autosaving → Autosaved

COMPLEX WRITES

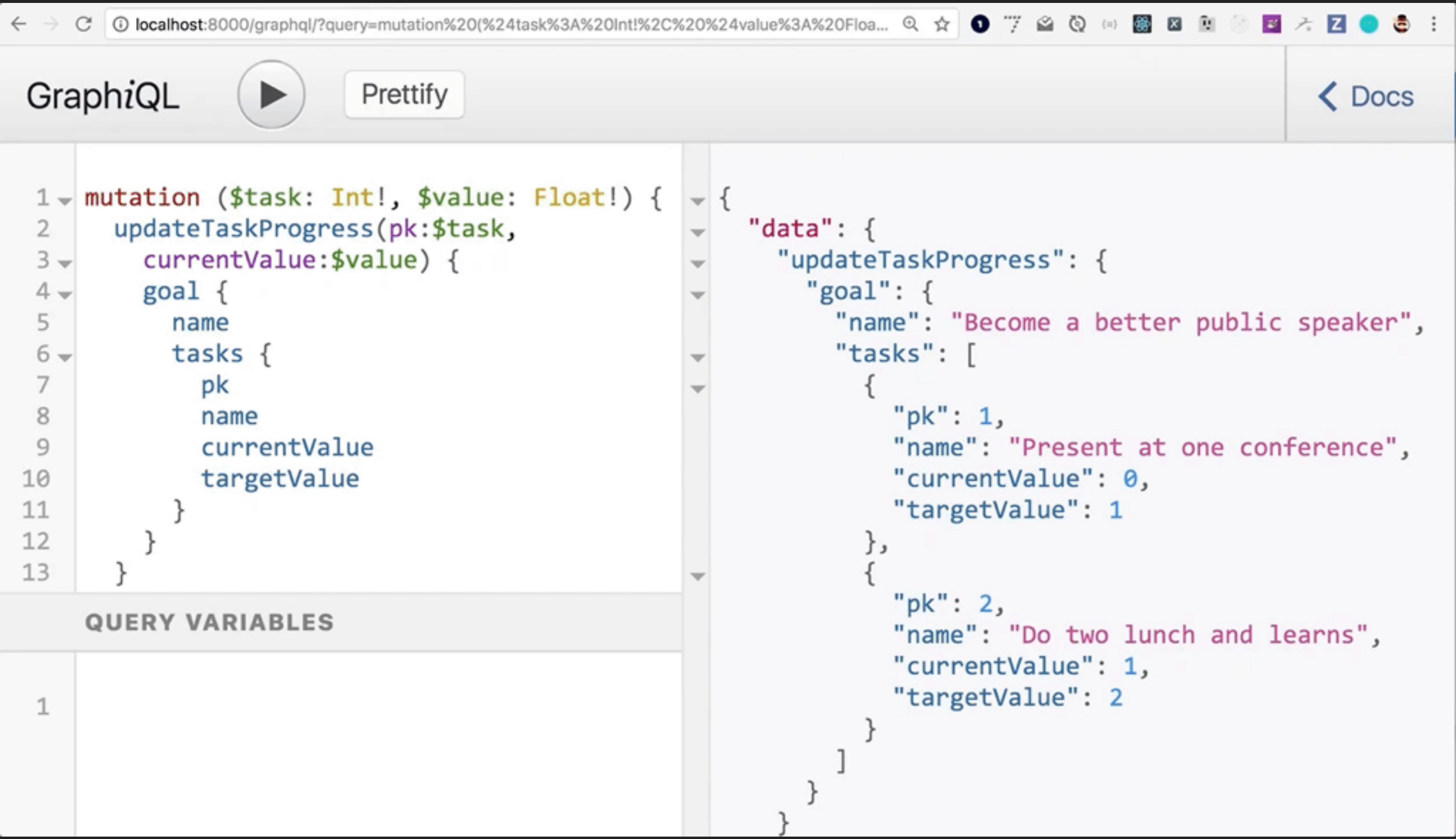
- Builders
- Autosaving
- Auto-updating



MUTATION?

DJANGO + GRAPHQL = GRAPHENE

- Setup
- Define queries (GET)
- Add filters & pagination
- **Define mutations (POST, PUT, DELETE)**



localhost:8000/graphql/?query=mutation%20(%24task%3A%20Int!%2C%20%24value%3A%20Floa...

```
mutation ($task: Int!, $value: Float!) {
  updateTaskProgress(pk:$task,
    currentValue:$value) {
    goal {
      name
      tasks {
        pk
        name
        currentValue
        targetValue
      }
    }
  }
```

```
{
  "data": {
    "updateTaskProgress": {
      "goal": {
        "name": "Become a better public speaker",
        "tasks": [
          {
            "pk": 1,
            "name": "Present at one conference",
            "currentValue": 0,
            "targetValue": 1
          },
          {
            "pk": 2,
            "name": "Do two lunch and learns",
            "currentValue": 1,
            "targetValue": 2
          }
        ]
      }
    }
```

DEFINE INPUTS

```python
class TaskInput(InputObjectType):
  name = String()
  progress = Float()

class GoalInput(InputObjectType):
  name = String()
  tasks = List(TaskInput)
```

DEFINE MUTATION

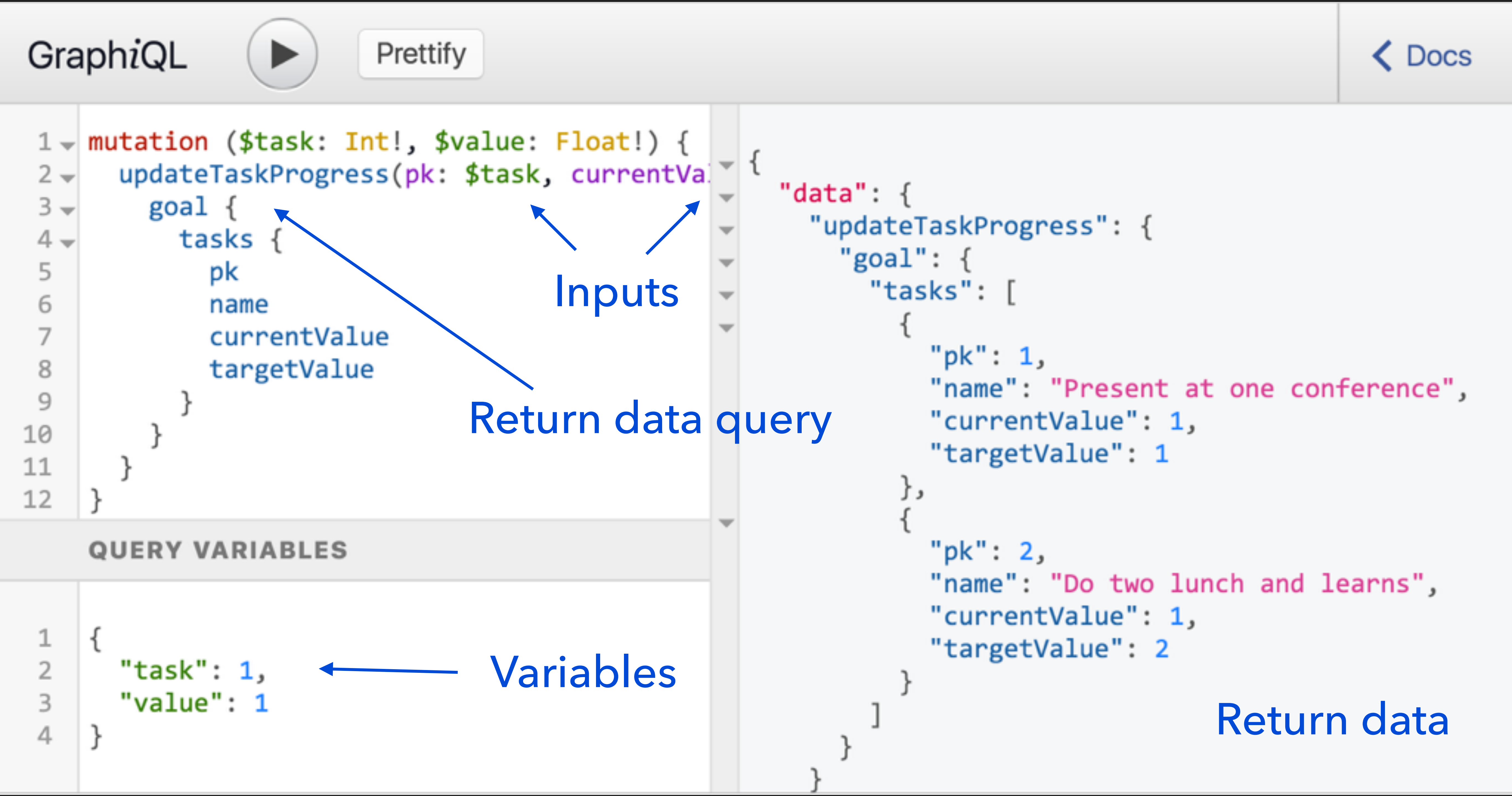
```
class UpdateTask(Mutation):
  class Input(object):
    pk = Int(required=True)
    current_value = Float(required=True)

  goal = Field(GoalNode)

  @atomic
  def mutate(self, args, context, info):
    ... do stuff here
      return UpdateTask(goal=task.objective)
```

Define inputs

Define return data

ADD TO SCHEMA

```
class Mutation(AbstractType):
  updateTaskProgress = UpdateTask.Field()


schema = Schema(
  query=Query,
  mutation=Mutation
)
```



```
mutation ($task: Int!, $value: Float!) {
  updateTaskProgress(pk: $task, currentVa
    goal {
      tasks {
        pk
        name
        currentValue
        targetValue
      }
    }
  }
}
```

```
{
  "task": 1,
  "value": 1
}
```

```
{
  "data": {
    "updateTaskProgress": {
      "goal": {
        "tasks": [
          {
            "pk": 1,
            "name": "Present at one conference",
            "currentValue": 1,
            "targetValue": 1
          },
          {
            "pk": 2,
            "name": "Do two lunch and learns",
            "currentValue": 1,
            "targetValue": 2
          }
        ]
      }
    }
```

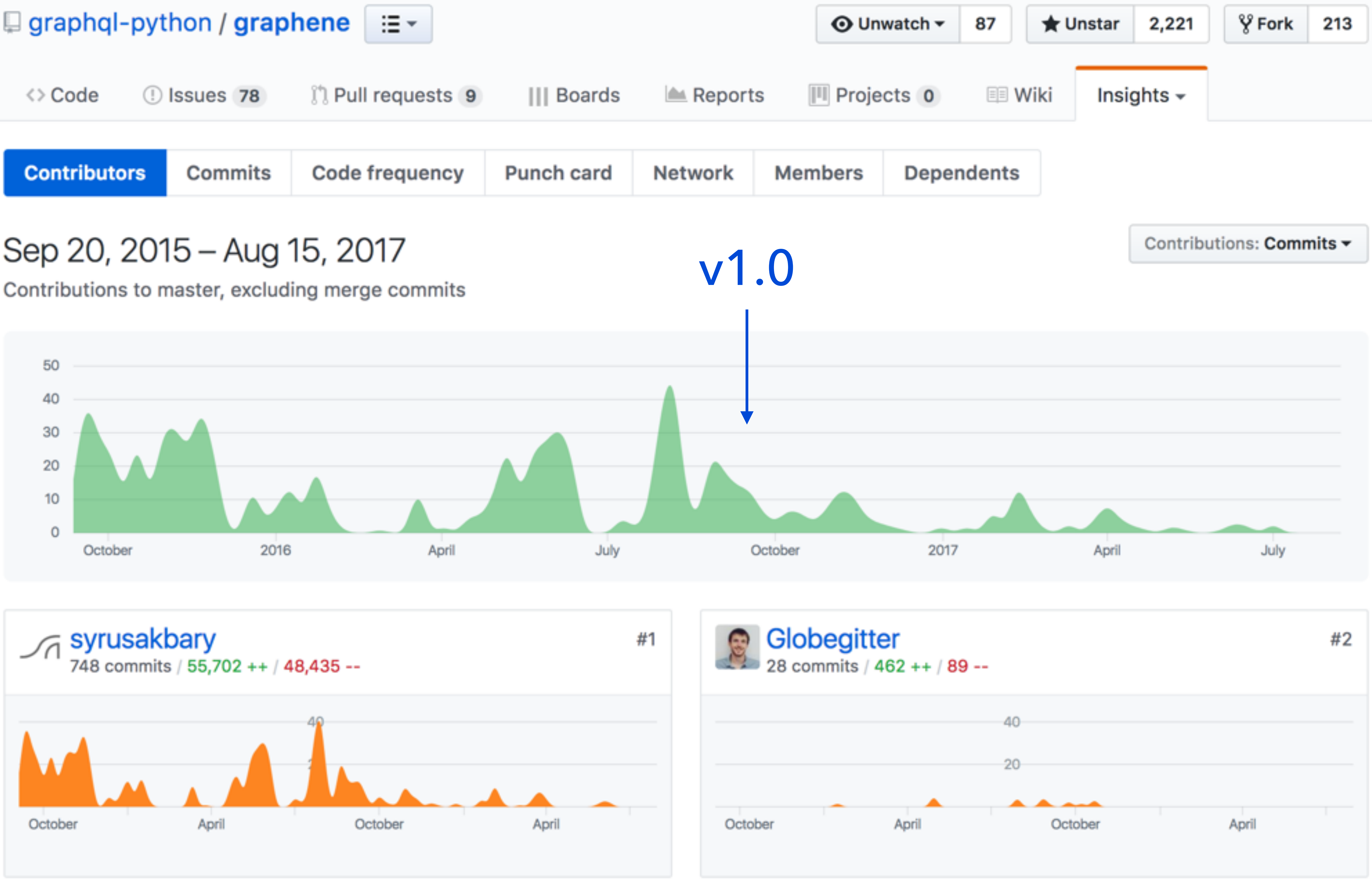
PROS

- Explorable / fun!
- Easy documentation
- More intuitive to implement than DRF
- **Simplifies client-side logic**

CONSIDERATIONS

- ?

What’s the catch?



graphql-python / graphene

Unwatch 87 | Unstar 2,221 | Fork 213

Code | Issues 78 | Pull requests 9 | Boards | Reports | Projects 0 | Wiki | Insights

Contributors | Commits | Code frequency | Punch card | Network | Members | Dependents

Sep 20, 2015 – Aug 15, 2017

Contributions to master, excluding merge commits

Contributions: Commits

syrusakbary #1
748 commits / 55,702 ++ / 48,435 --

Globegitter #2
28 commits / 462 ++ / 89 --

GRAPHENE

- v1.0 released Sept 2016
- Docs need work
- Lags behind GraphQL specs
- Some known bugs
- Source code is complicated

THE REAL WORLD IS MESSY

spaghetti_88 Angerilli

What about permissions?

AUTHORIZATION

1. Perform authorization on each resolver
2. Extend Graphene to perform authorization on every connection

AUTHORIZATION

1. Specify DRF Authorization class on each node
2. Extend DjangoFilterConnectionField
   - `connection_resolver`:

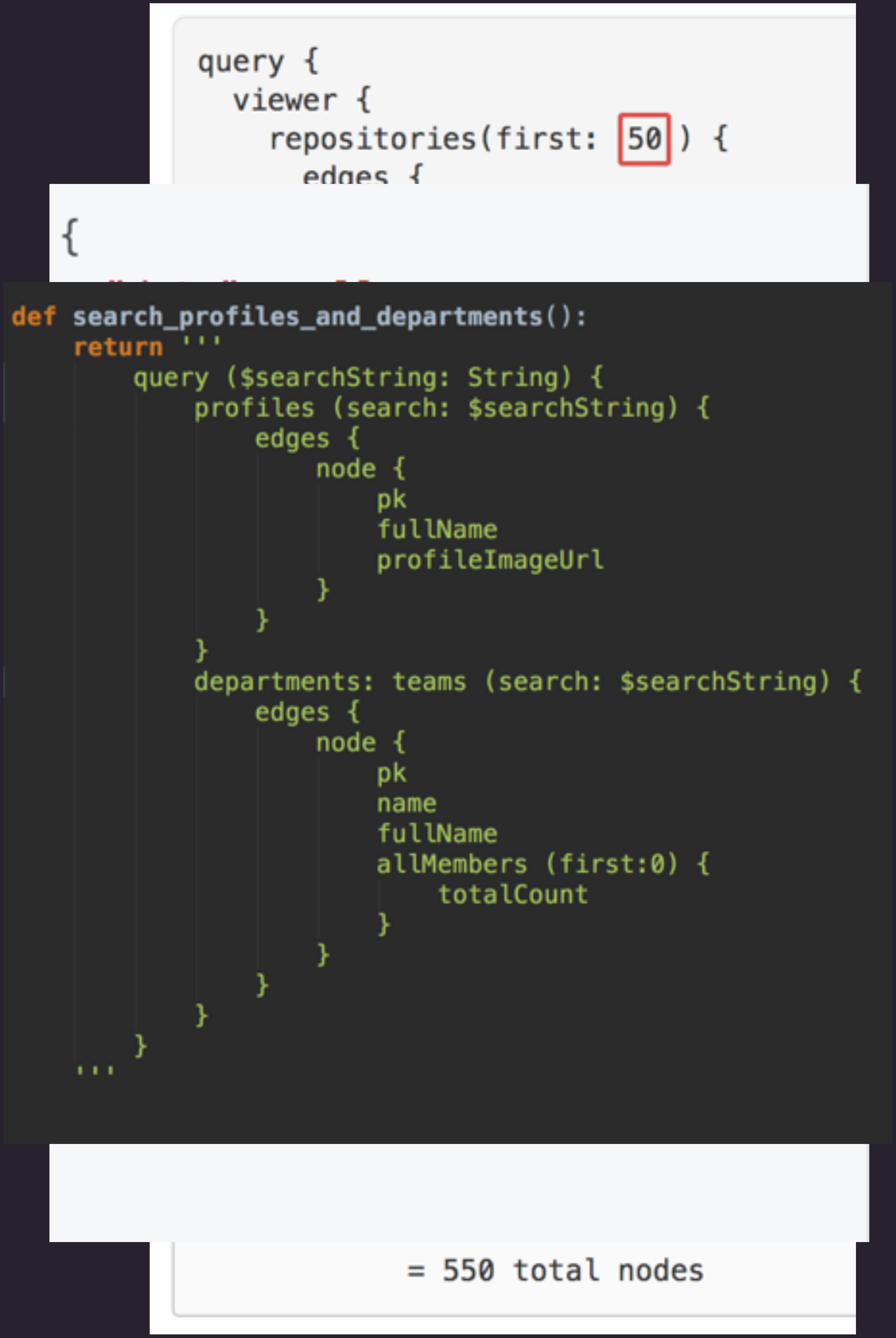
     add user authentication
   - `resolve_connection`:

     get auth class from node & apply auth

What if someone requests too much data?

DENIAL OF SERVICE (DoS)

1. Whitelist for allowed queries
2. Maximum limit
3. Maximum query cost
4. Rate limiting based on query cost



```
query {
  viewer {
    repositories(first: 50) {
      edges {
```

```
{
```

```
def search_profiles_and_departments():
    return '''
        query ($searchString: String) {
            profiles (search: $searchString) {
                edges {
                    node {
                        pk
                        fullName
                        profileImageUrl
                    }
                }
            }
            departments: teams (search: $searchString) {
                edges {
                    node {
                        pk
                        name
                        fullName
                        allMembers (first:0) {
                            totalCount
                        }
                    }
                }
            }
        }
    '''
```

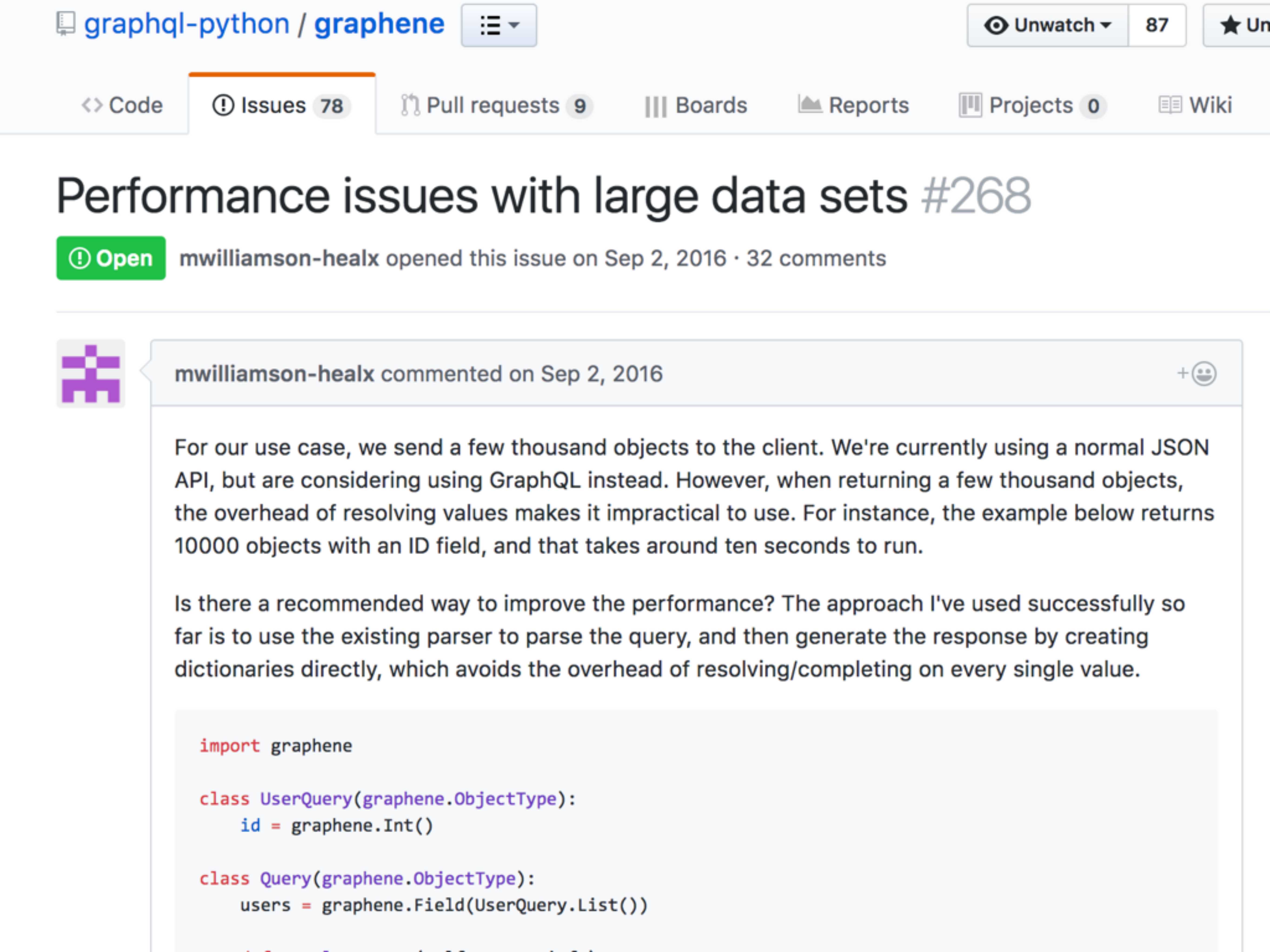
```
= 550 total nodes
```

What about performance?

TOO MUCH TIME

- Performance enhancements
- Data Loader
- Max query cost
- Front-end education

```
goals {
  edges {
1 -  node {
      pk
1 -   }
    }
1 - }
}
5    -
1 -
=>
5 4!    -
= 13
goals (first: 0) {
  totalCount
}
```



Performance issues with large data sets #268

Open mwilliamson-healx opened this issue on Sep 2, 2016 · 32 comments

mwilliamson-healx commented on Sep 2, 2016

For our use case, we send a few thousand objects to the client. We're currently using a normal JSON API, but are considering using GraphQL instead. However, when returning a few thousand objects, the overhead of resolving values makes it impractical to use. For instance, the example below returns 10000 objects with an ID field, and that takes around ten seconds to run.

Is there a recommended way to improve the performance? The approach I've used successfully so far is to use the existing parser to parse the query, and then generate the response by creating dictionaries directly, which avoids the overhead of resolving/completing on every single value.

```
import graphene

class UserQuery(graphene.ObjectType):
    id = graphene.Int()

class Query(graphene.ObjectType):
    users = graphene.Field(UserQuery.List())
```

PROS

- Explorable / fun!
- Easy documentation
- More intuitive to implement than DRF
- Simplifies client-side logic

CONSIDERATIONS

- **Graphene is still young**
- **Authorization**
- **Denial of Service**
- **Performance!!**

Should you use it?

🚀 🚀 🚀?

GO FOR IT

- Side project / for fun
- REST is causing performance issues
- REST format is complicating reading or writing
- You have the resources / know-how to extend it

HOLD UP

- Sensitive information
- Public API

AND

- Not enough development resources
- Not enough experience to extend it

RESOURCES

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