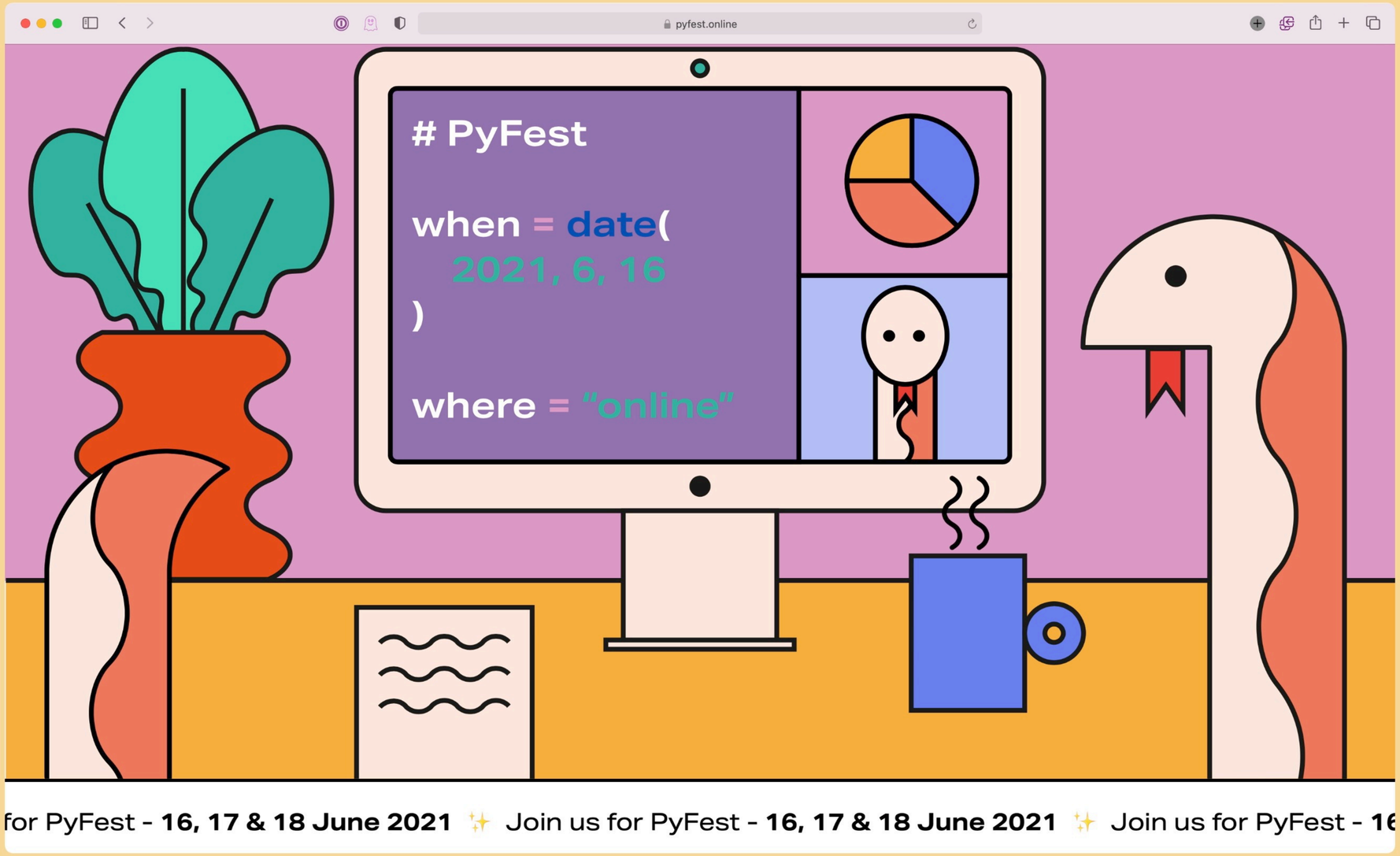
Domain Driven Design with Django & GraphQL

My name is **Patrick**!

I work at **Pollen** 🐝

I’m also the president of **Python Italia**

(https://pycon.it)


pyfest.online
# PyFest
when = date(
2021, 6, 16
)
where = "online"
for PyFest - 16, 17 & 18 June 2021 ✨ Join us for PyFest - 16, 17 & 18 June 2021 ✨ Join us for PyFest - 16

https://pyfest.online

What's GraphQL?

GraphQL is a query language

/graphql

```
query {
    blogPost(id: "1") {
        title
        content
        author {
            fullName
        }
    }
}
```

GraphQL supports multiple operations

Queries

Mutations

Subscriptions

GraphQL has a Typed schema

```
type User {
    fullName: String!
}

type BlogPost {
    id: ID!
    title: String!
    content: String!
    author: User!
}

type Query {
    blogPost(id: ID!): BlogPost
}
```

Code first:

https://strawberry.rocks/

https://graphene-python.org/

Schema first:

https://ariadnegraphql.org/

https://tartiflette.io/


STRAW
BERRY

```python
import strawberry

@strawberry.type
class User:
    full_name: str

@strawberry.type
class BlostPost:
    id: strawberry.ID
    title: str
    content: str
    author: User
```

```python
import strawberry

from typing import Optional

@strawberry.type
class Query:
    @strawberry.field
    def blog_post(self, id: strawberry.ID) -> Optional[BlogPost]:
        # fetch and return a blog post

        ...
```

```python
schema = strawberry.Schema(Query)
```

GraphQL is very powerful and flexible

Do we need DDD for GraphQL?

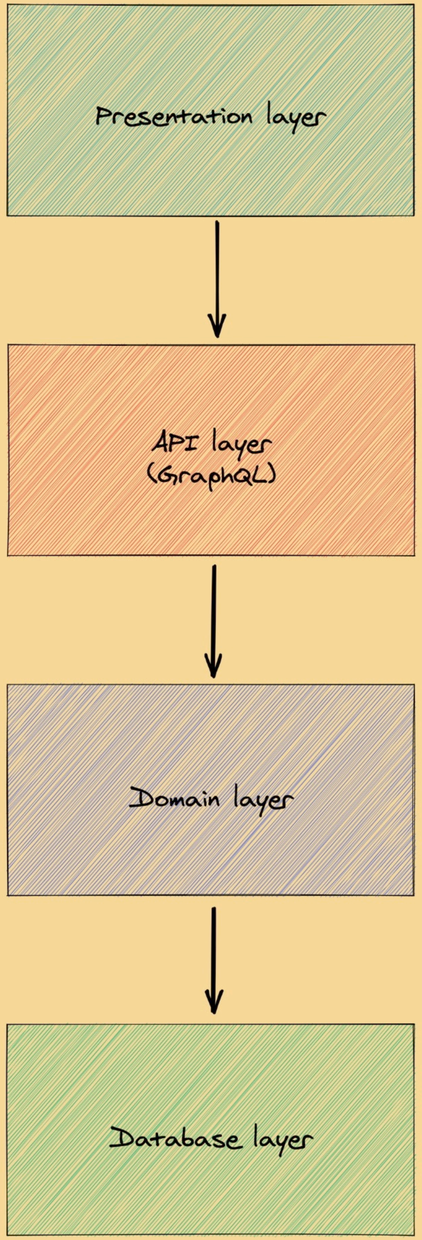
So, why DDD?

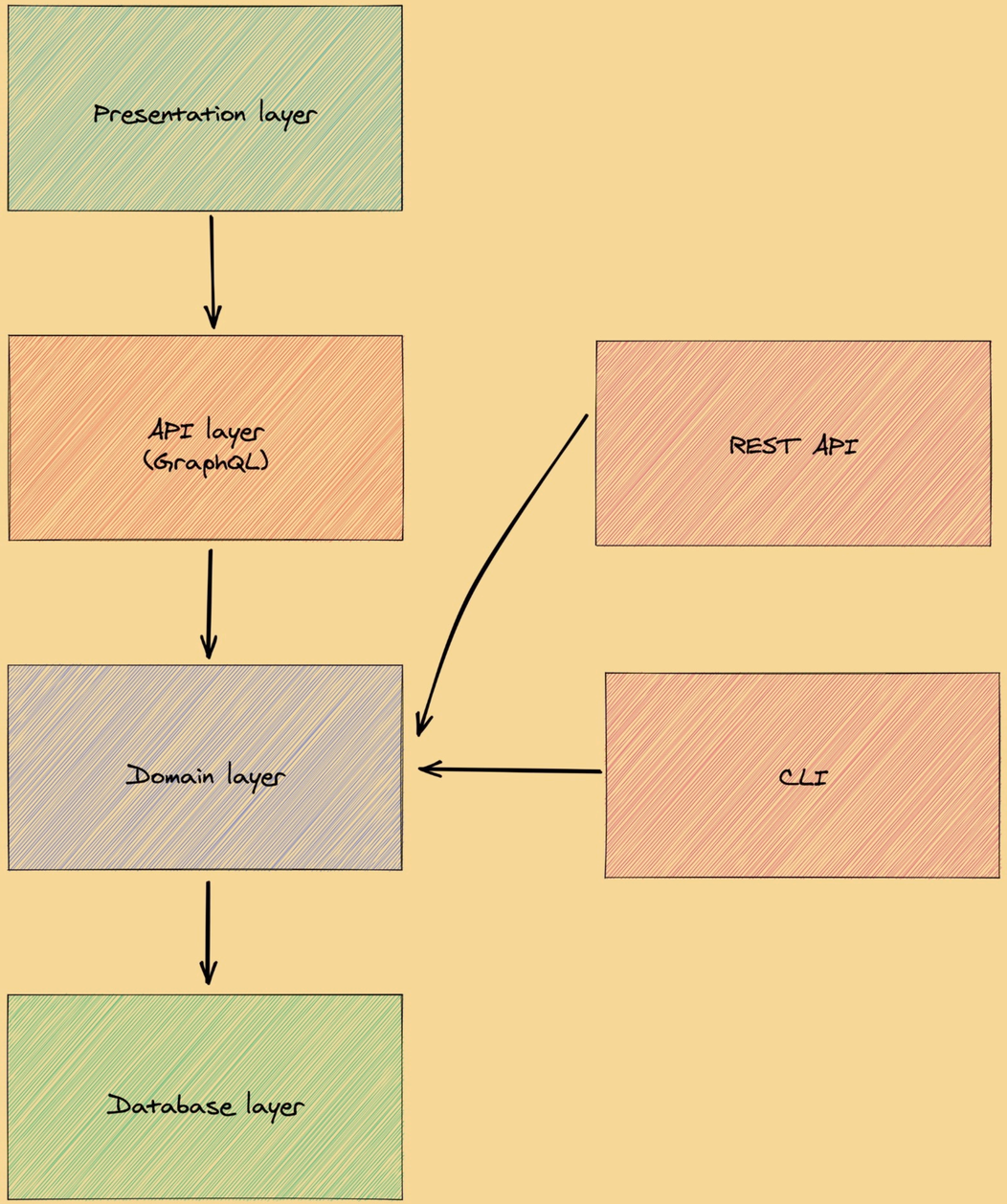
An approach that focuses on the domain

Ubiquitous language

Solutions first;
Technology later

Layered architecture


Presentation layer
API layer
(GraphQL)
Domain layer
Database layer


Presentation layer
API layer
(GraphQL)
REST API
Domain layer
CLI
Database layer

How can we implement domain driven design?

A basic ddd structure

- Entities
- Repositories
- Services

Entities

```python
from dataclasses import dataclass


@dataclass
class Event:
    name: str
    tags: list[str]
```

Entities represent business concepts, not database ones

Repositories

```python
class EventRepository:
    def get(self, id: str) -> Event:
        ...

    def save(self, event: Event) -> Event:
        ...
```

Services

```python
def assign_manager_to_event(event: Event, new_manager: Manager,
repository: EventRepository) → Event:

    if not manager.active:
        raise ValueError('Cannot assign inactive manager')

    event.manager = manager
    event = repository.save(event)

    # more side effects (ie. notifying old manager)

    return event
```

There's more but
this is a good starting point

Using ddd in a django view

```python
def event_detail(request, event_id):
    event_repository = EventRepository()

    event = event_repository.get(id=event_id)

    if event is None:
        raise Http404()

    return render(request, 'events/detail.html', {
        "event": event
    })
```

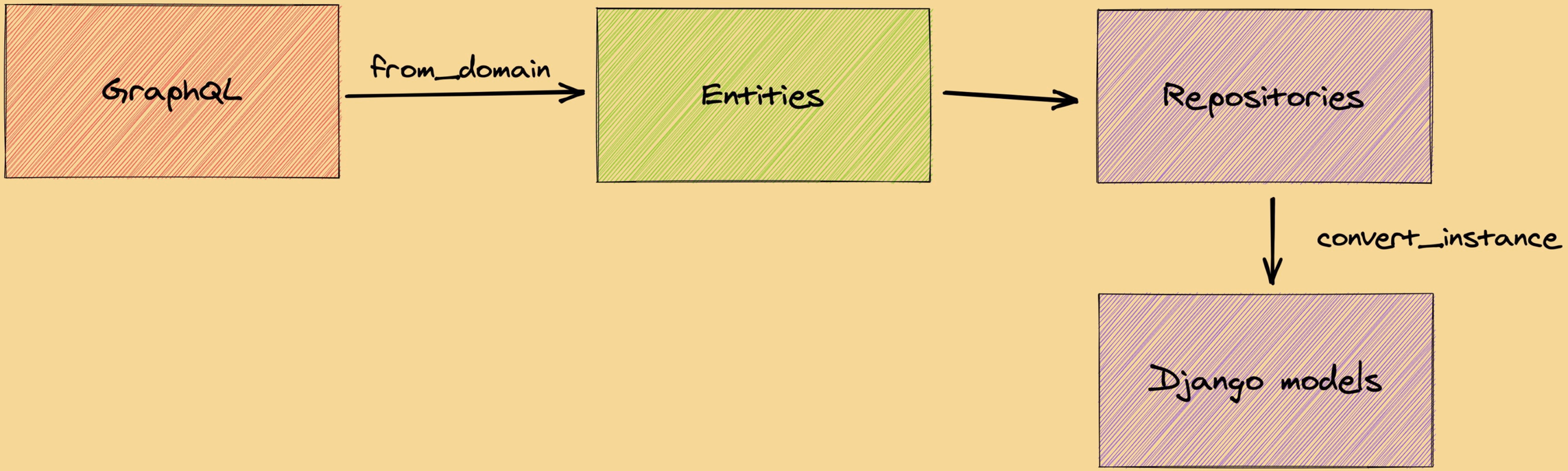
Django is opinionated

Django's guidelines might not work for big projects

Django & DDD

Maybe a leaner framework is better

How we do DDD with Django at Pollen


GraphQL
from_domain
Entities
Repositories
convert_instance
Django models

Using Django models as entities?

A small example

Our Django Project

- api
- app
- db

```python
def create_event(title: str, start_date: date, repository:
AbstractEventRepository):
    if start_date < start_date.today():
        raise InvalidEventDateError()

    event = Event(id=uuid4(), title=title, start_date=start_date)

    return repository.save(event)
```

Passing the repository as a dependency

```python
from typing import Optional, Protocol

from .entities import Event


class AbstractEventRepository(Protocol):
    def get(self, id: str) -> Optional[Event]:
        ...

    def save(self, event: Event) -> Event:
        ...
```

```python
class EventTestRepository:
    def get(self, id: str) -> Optional[Event]:
        return None

    def save(self, event: Event) -> Event:
        return event


def test_create_event():
    start_date = date.today() + timedelta(days=7)

    event = create_event(
        title="DjangoCon Europe",
        start_date=start_date,
        repository=EventTestRepository(),
    )

    assert event.id
    assert event.title == "DjangoCon Europe"
    assert event.start_date == start_date
```

DDD & GraphQL

```
mutation {
  createEvent(
    title: "A new event",
    startDate: "2021-06-16"
  ) {
    id
    title
  }
}
```

```python
@strawberry.type
class Event:
    id: strawberry.ID
    title: str
    start_date: date

    @classmethod
    def from_domain(cls, entity: entities.Event) -> Event:
        return cls(
            id=entity.id,
            title=entity.title,
            start_date=entity.start_date,
        )
```

```python
@strawberry.mutation
def create_event(
    info: Info[Context, Any], title: str, start_date: date
) -> Event:
    domain_event = services.create_event(
        title=title,
        start_date=start_date,
        repository=info.context.event_repository
    )

    return Event.from_domain(domain_event)
```

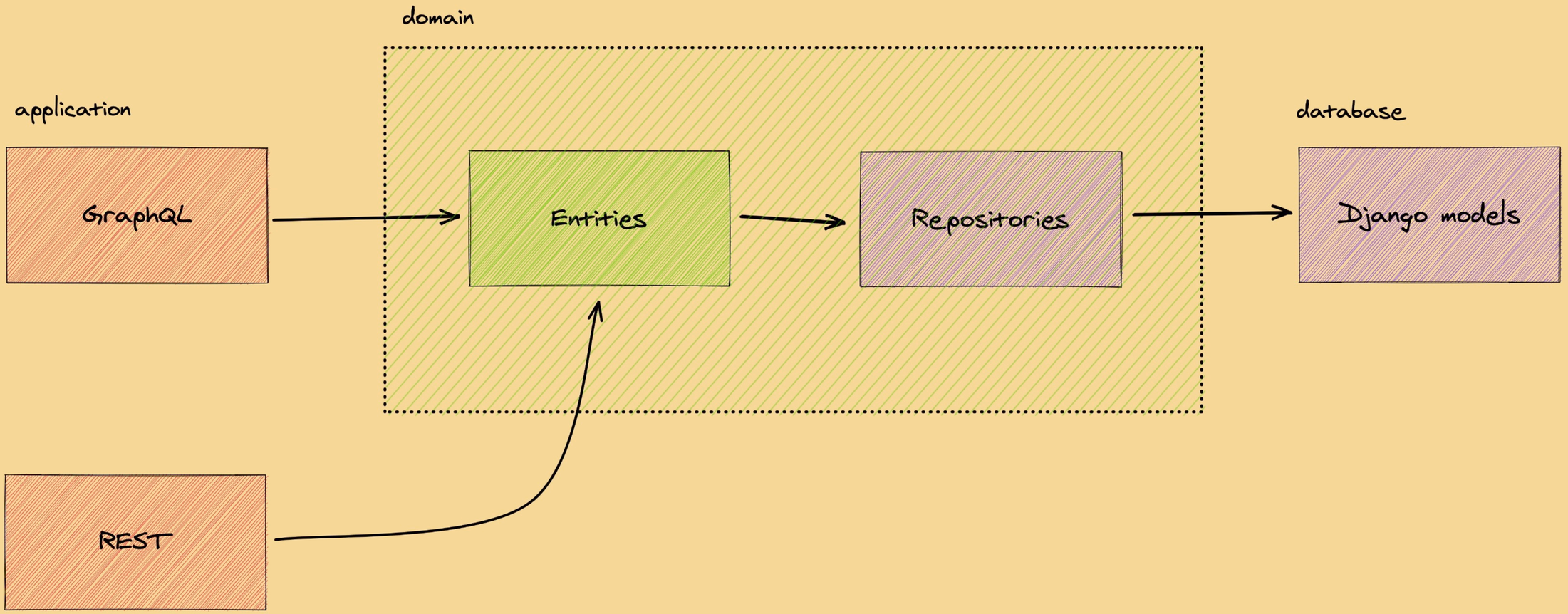
```python
class TestRepository:
    def get(self, id: str) -> Optional[Event]:
        return None

    def save(self, event: Event) -> Event:
        return event


def test_create_event():
    query = """mutation ($title: String!, $startDate: Date!) {
        createEvent(title: $title, startDate: $startDate) {
            id
            title
            startDate
        }
    }"""

    result = schema.execute_sync(
        query,
        variable_values={"title": "PyFest", "startDate": "2021-06-16"},
        context_value=Context(event_repository=TestRepository()),
    )

    assert result.data["createEvent"]["id"]
    assert result.data["createEvent"]["title"] == "PyFest"
    assert result.data["createEvent"]["startDate"] == "2021-06-16"
```

```python
from strawberry.django.views import GraphQLView as BaseGraphQLView

from app.events.repository import AbstractEventRepository, DjangoEventRepository

from .schema import schema


@dataclass
class Context:
    event_repository: AbstractEventRepository


class GraphQLView(BaseGraphQLView):
    def __init__(self):
        super().__init__(schema, True)

    def get_context(self, request: HttpRequest, response: HttpResponse) -> Context:
        return Context(event_repository=DjangoEventRepository())
```

Django repository

```python
class DjangoEventRepository:
    def get(self, id: str) -> Optional[Event]:
        db_event = models.Event.objects.filter(id=id).first()

        if db_event:
            return self._convert_instance(db_event)

        return None

    def save(self, event: Event) -> Event:
        db_event, _ = models.Event.objects.update_or_create(
            id=event.id,
            defaults={"title": event.title, "start_date": event.start_date},
        )

        return self._convert_instance(db_event)

    @staticmethod
    def _convert_instance(instance: models.Event) -> Event:
        return Event(
            id=instance.id, title=instance.title, start_date=instance.start_date
        )
```

Event, Event, Event


domain
application
database
GraphQL
Entities
Repositories
Django models
REST

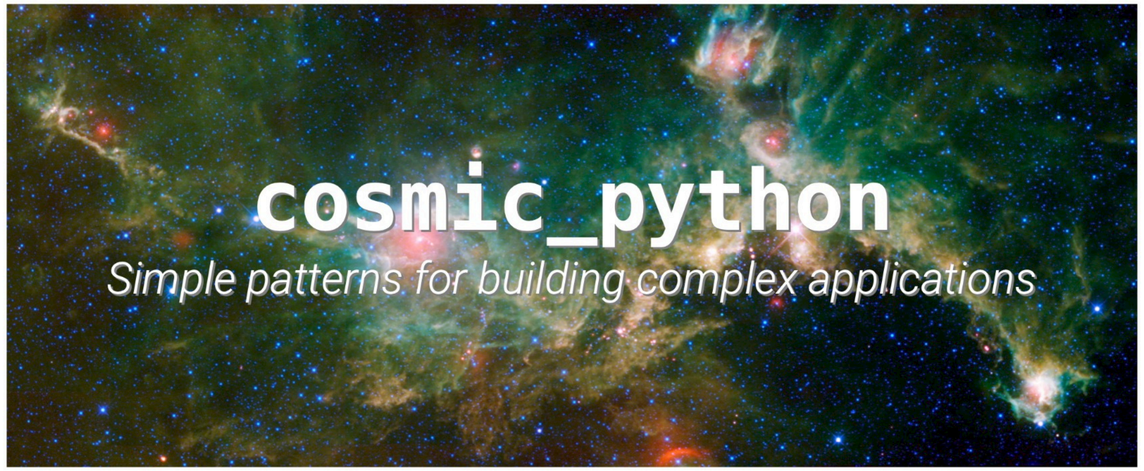
Auto generating GraphQL types?

That was
just code

The focus should be on the domain

Some resources

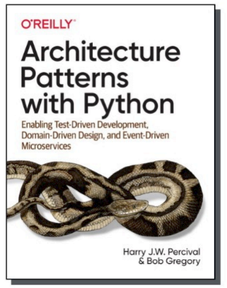
Cosmic Python



(Because "Cosmos" is the opposite of Chaos, you see)

The Book

There are lots of ways you can read the book. Some of them even involve us, the authors, receiving a small amount of money!

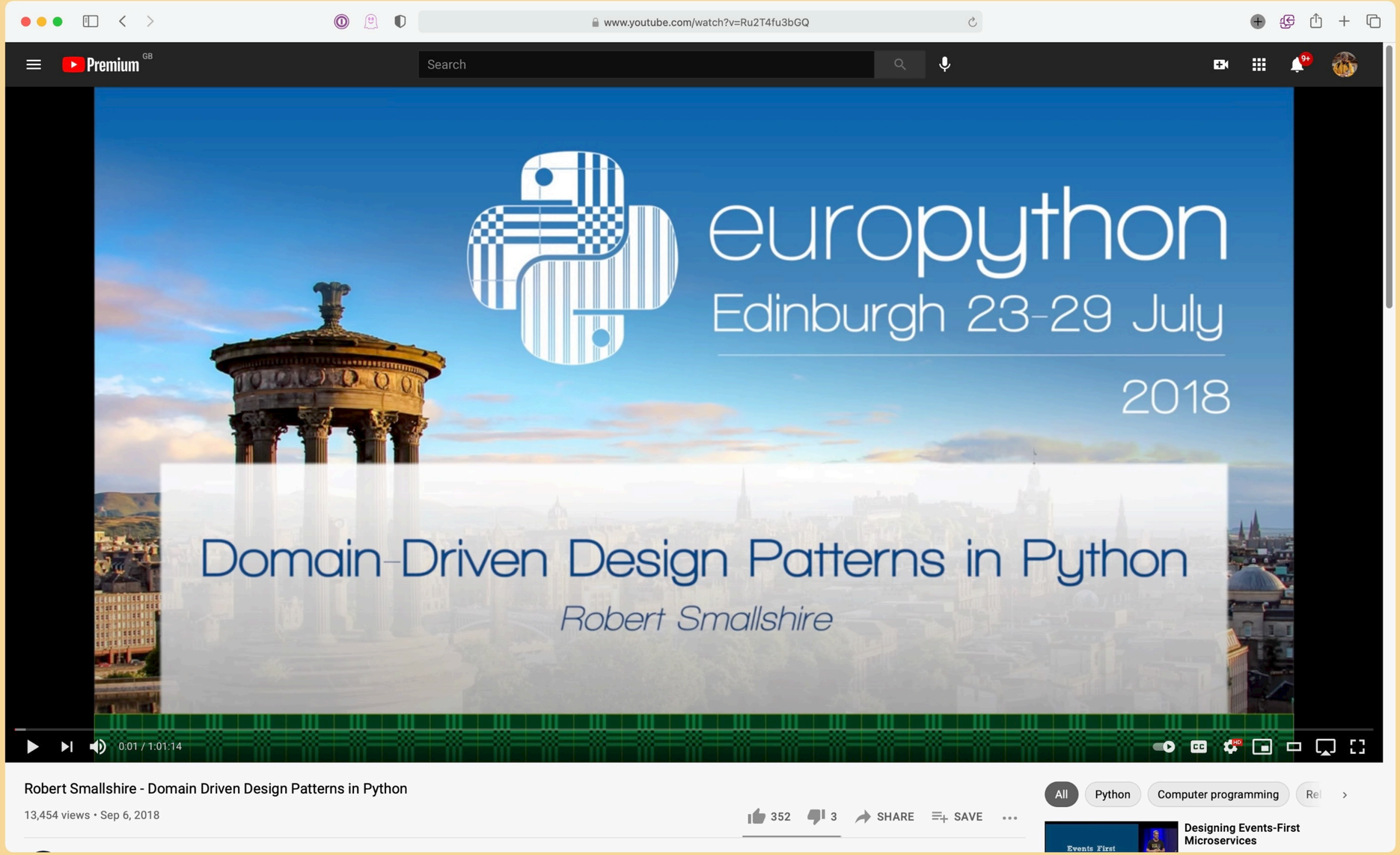


- Read it online on O'Reilly Learning (aka Safari) learning.oreilly.com
- Buy print books on Amazon.com or Amazon.co.uk.
- Buy a DRM-free ebook at ebooks.com
- Or **read it for free on this site** (license: CC-By-NC-ND).

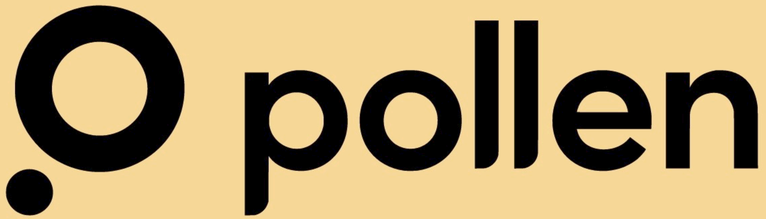
Blog

Recent posts

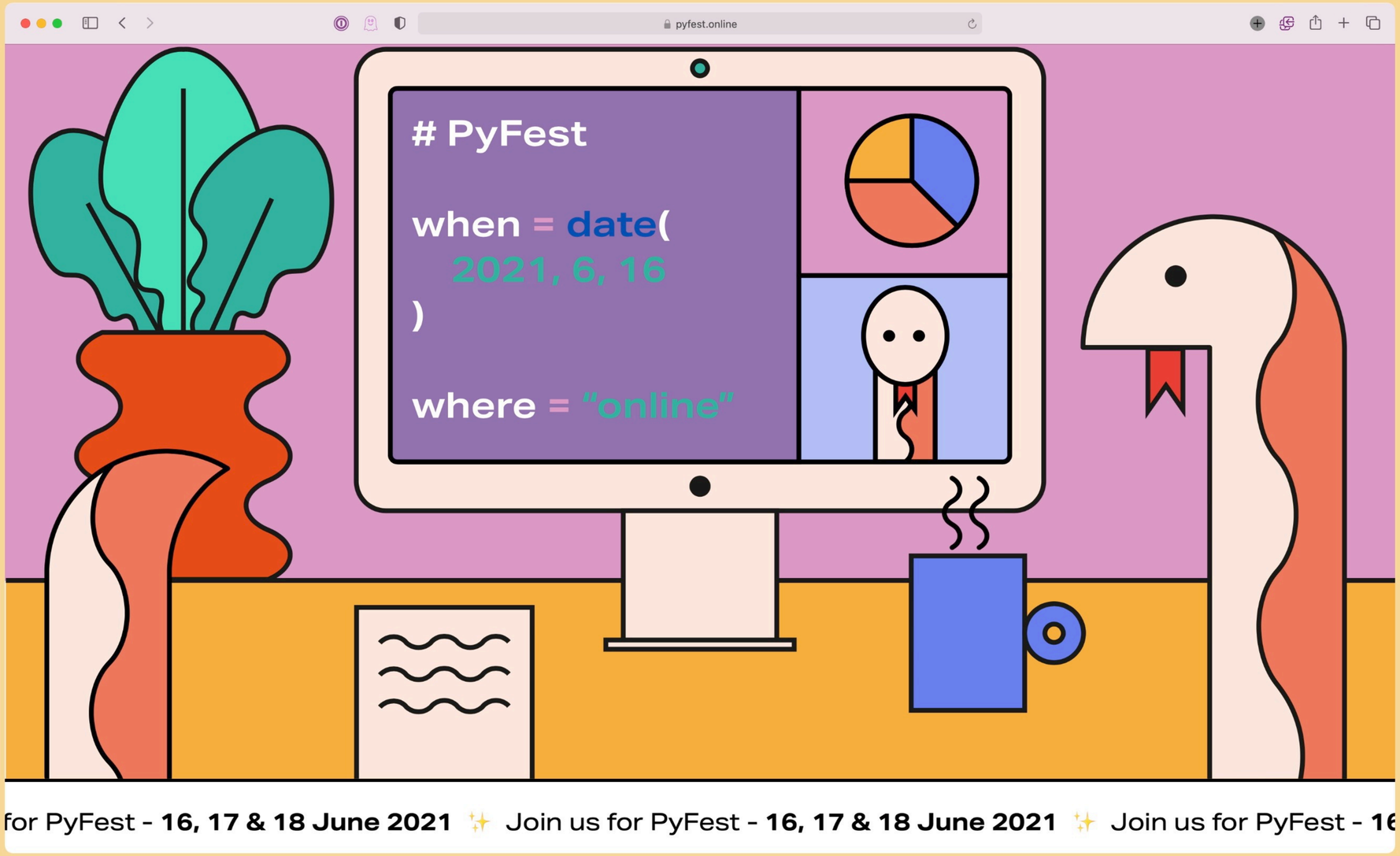
- 2020-10-27 Making Enums (as always, arguably) more Pythonic


www.youtube.com/watch?v=Ru2T4fu3bGQ
Premium GB
Search
europython
Edinburgh 23-29 July
2018
Domain-Driven Design Patterns in Python
Robert Smallshire
0:01 / 1:01:14
Robert Smallshire - Domain Driven Design Patterns in Python
13,454 views • Sep 6, 2018
352
3
SHARE
SAVE
All
Python
Computer programming
Designing Events-First Microservices

We are hiring!



https://pollen.co/team


pyfest.online
# PyFest
when = date(
2021, 6, 16
)
where = "online"
for PyFest - 16, 17 & 18 June 2021 ✨ Join us for PyFest - 16, 17 & 18 June 2021 ✨ Join us for PyFest - 16

Thank you!

You can find me as **@patrick91** on twitter

or on my website: **https://patrick.wtf**