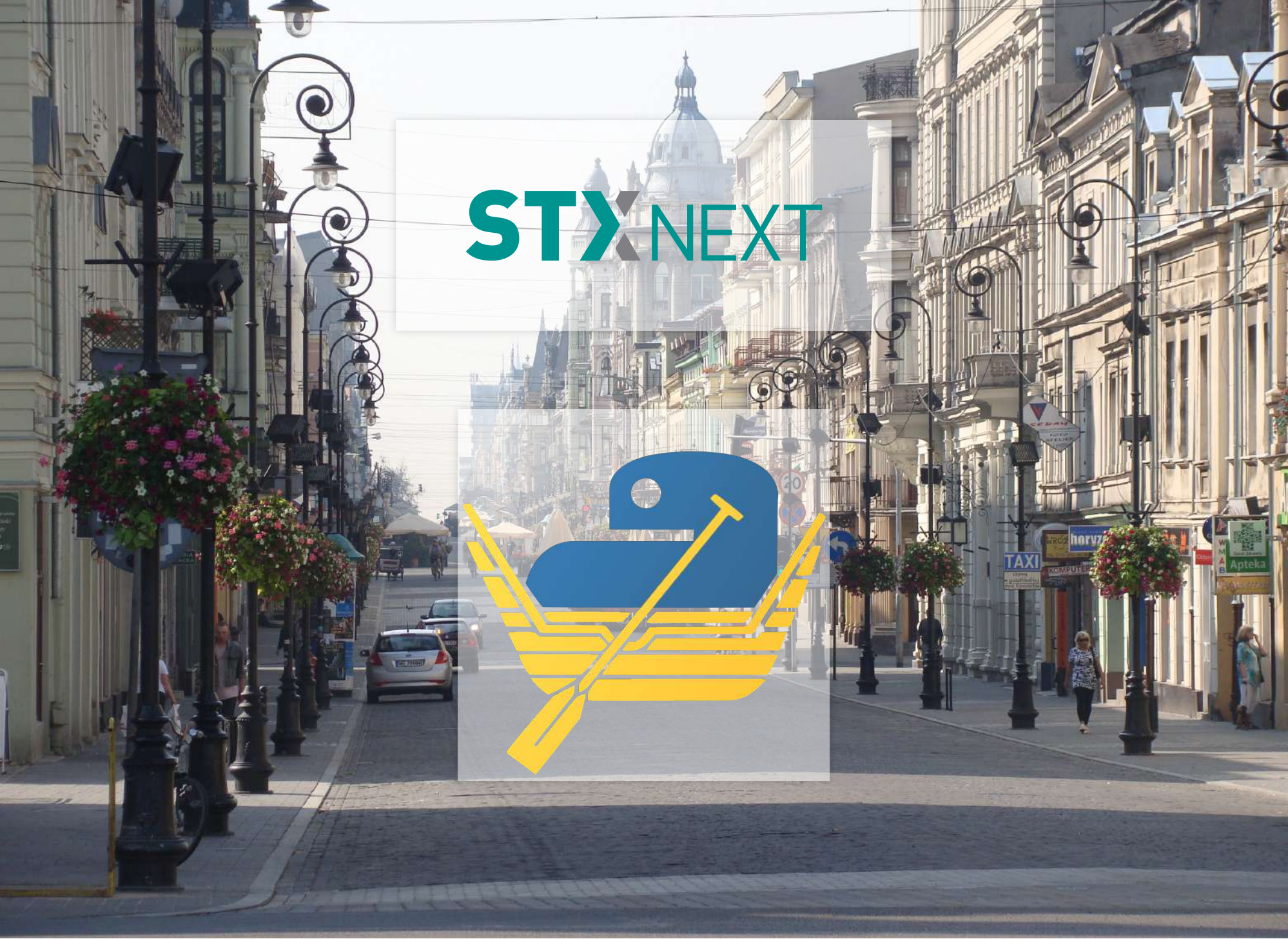
Czysta Architektura

Sebastian Buczyński @ Python Łódź 29.01.2018


STX NEXT
TAXI
Apteka

INŻYNIERIA OPROGRAMOWANIA ?

PIP INSTALL

Co daje Czysta Architektura?

1. Niezależność od frameworków
2. Testowalność
3. Niezależność od UI
4. Niezależność od bazy danych

Projekt: Aukcje online

Historyjki użytkownika

- Jako licytujący chcę złożyć ofertę na aukcji by ją wygrać
- Jako licytujący chcę zostać powiadomiony mailowo gdy moja oferta jest najwyższa
- Jako administrator chcę wycofać ofertę z aukcji

Django

```
django-admin startproject mysite .
django-admin startapp auctions
```

Modele przodem

```
class Auction(models.Model):
    title = models.CharField(...)
    initial_price = models.DecimalField(...)
    current_price = models.DecimalField(...)

    def withdraw_bids(self, bids):
        ...

    @property
    def winners(self):
        ...


class Bid(models.Model):
    amount = models.DecimalField(...)
    bidder = models.ForeignKey(...)
    auction = models.ForeignKey(Auction, on_delete=PROTECT)
```

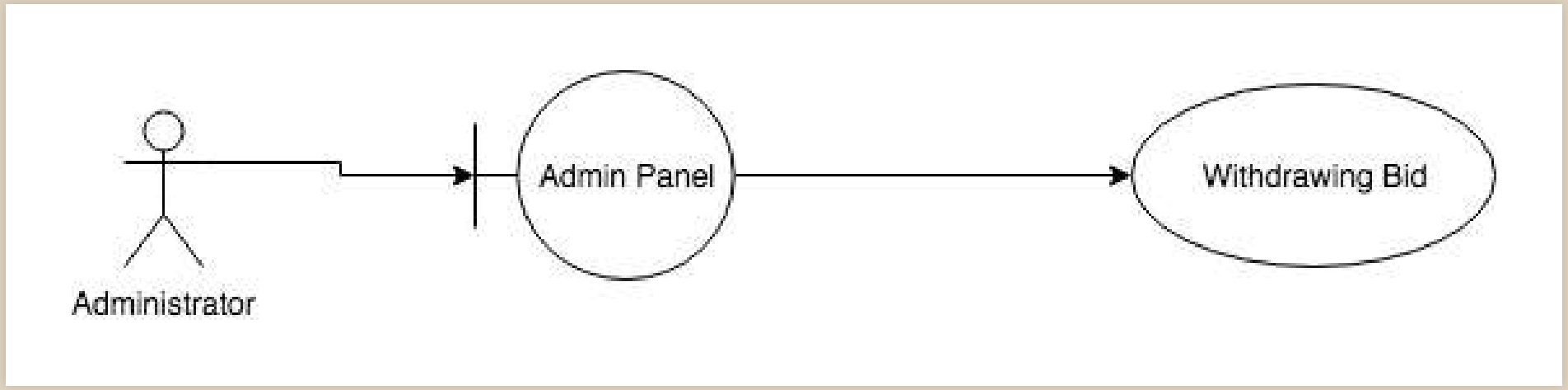
Historyjki użytkownika

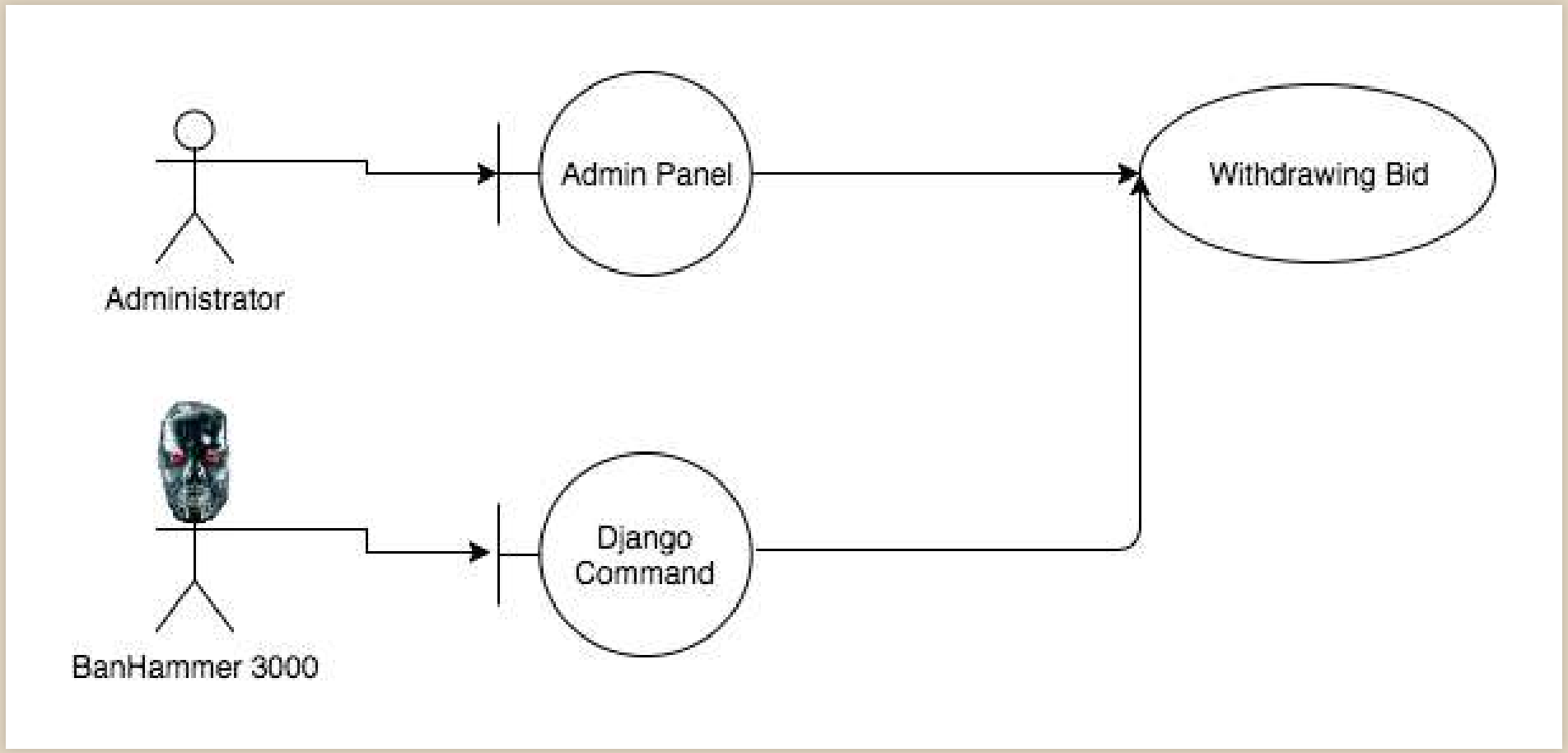
- ~~Jako licytujący chcę złożyć ofertę na aukcji by ją wygrać~~ ✔
- ~~Jako licytujący chcę zostać powiadomiony mailowo gdy moja oferta jest najwyższa~~ ✔
- Jako administrator chcę wycofać ofertę z aukcji

```
def save_related(self, request, form, formsets, *args, **kwargs):
    ids_of_deleted_bids = self._get_ids_of_deleted_bids(formsets)
    bids_to_withdraw = Bid.objects.filter(
        pk__in=ids_of_deleted_bids)

    auction = form.instance
    old_winners = list(auction.winners)
    auction.withdraw_bids(bids_to_withdraw)
    new_winners = list(auction.winners)

    self._notify_new_winners(new_winners)

    super().save_related(request, _form, formsets, *args, **kwarg
```

```
def save_related(self, request, form, formsets, *args, **kwargs):
    ids_of_deleted_bids = self._get_ids_of_deleted_bids(formsets)
    bids_to_withdraw = Bid.objects.filter(
        pk__in=ids_of_deleted_bids)

    auction = form.instance
    old_winners = list(auction.winners)
    auction.withdraw_bids(bids_to_withdraw)
    new_winners = list(auction.winners)

    self._notify_new_winners(new_winners)

    super().save_related(request, _form, formsets, *args, **kwarg
```

```
def save_related(self, request, form, formsets, *args, **kwargs):
    ids_of_deleted_bids = self._get_ids_of_deleted_bids(formsets)
    bids_to_withdraw = Bid.objects.filter(
        pk__in=ids_of_deleted_bids)

    auction = form.instance
    old_winners = list(auction.winners)
    auction.withdraw_bids(bids_to_withdraw)
    new_winners = list(auction.winners)

    self._notify_new_winners(new_winners)

    super().save_related(request, _form, formsets, *args, **kwarg
```


Administrator
Admin Panel
Withdrawing Bid


Administrator
Admin Panel
Withdrawing Bid
BanHammer 3000
Django
Command

Czysta architektura - element #1

```
class WithdrawingBid:
    def withdraw_bids(self, auction_id, bids_ids):
        auction = Auction.objects.get(pk=auction_id)
        bids_to_withdraw = Bid.objects.filter(
            pk__in=ids_of_deleted_bids)

        old_winners = list(auction.winners)
        auction.withdraw_bids(bids_to_withdraw)
        new_winners = list(auction.winners)

        self._notify_new_winners(new_winners)
```

UseCase LUB Interactor

UseCase - odpowiedzialności

- Orkiestracja całego procesu
- Jedyny sposób zmiany stanu aplikacji

A testy?!

Kod związany z frameworkiem, to i testy...

Testy przez widoki

```
from django.test import TestCase

class LoginTestCase(TestCase):

    def test_login(self):

        # First check for the default behavior
        response = self.client.get('/sekrit/')
        self.assertRedirects(response, '/accounts/login/?next=/se
```

Co zrobiliśmy nie tak? Poszukajmy w necie...

```
class MyTest(unittest.TestCase):
    def test_add(self):
        expected = 7

        self.assertEqual(add(3, 4), 7)
```

Obserwacja: Brak efektów ubocznych i zależności ułatwia testowanie

```
class WithdrawingBidUseCase:
    def withdraw_bids(self, auction_id, bids_ids):
        auction = Auction.objects.get(pk=auction_id)
        bids_to_withdraw = Bid.objects.filter(
            pk__in=ids_of_deleted_bids)

        old_winners = list(auction.winners)
        auction.withdraw_bids(bids_to_withdraw)
        new_winners = list(auction.winners)

        self._notify_new_winners(new_winners)
```

```
class WithdrawingBidUseCase:
    def withdraw_bids(self, auction_id, bids_ids):
        auction = self.auctions_repository.get(auction_id)
        bids = self.bids_repository.get_by_ids(bids_ids)

        old_winners = list(auction.winners)
        auction.withdraw_bids(bids)
        new_winners = list(auction.winners)

        self.auctions_repository.save(auction)
        for bid in bids:
            self.bids_repository.save(bid)

        self._notify_new_winners(new_winners)
```

Czysta architektura - element #2

```
class AuctionsRepo(metaclass=ABCMeta):

    @abstractmethod
    def get(self, auction_id):
        pass

    @abstractmethod
    def save(self, auction):
        pass
```

Interface / Port

Czysta architektura - element #3

```
class DjangoAuctionsRepo(AuctionsRepo):

    def get(self, auction_id):
        return Auction.objects.get(pk=auction_id)
```

Interface / Port Adapter

Łączymy razem

```
class WithdrawingBidUseCase:
    def __init__(self, auctions_repository: AuctionsRepo):
        self.auctions_repository = auctions_repository
```

```
django_adapter = DjangoAuctionsRepo()
withdrawing_bid_uc = WithdrawingBidUseCase(django_adapter)
```

Dependency Injection

```
class WithdrawingBidUseCase:

    auctions_repo: AuctionsRepo = ...
```

```
class WithdrawingBidUseCase:

    auctions_repo: AuctionsRepo = inject.attr(AuctionsRepo)
```

```
import inject

def configure_inject(binder: inject.Binder):
    binder.bind(AuctionsRepo, DjangoAuctionsRepo())

inject.configure_once(configure_inject)
```

Dependency Injection pozwala odwrócić zależności i utrzymać kod jeszcze czystszym

Cześć konfiguracji; w Django dobrym miejscem jest *AppConfig.ready()*

Korzyści z dodatkowej warstwy

- Ułatwia zrozumienie
- Prawdziwe testy jednostkowe logiki aplikacji
- Zrównoleglenie pracy nad tym samym zadaniem
- **Możliwość odroczenia decyzji**

Nadal logika jest splątana z bazą danych!

```
class WithdrawingBidUseCase:
    def withdraw_bids(self, auction_id, bids_ids):
        auction = self.auctions_repository.get(auction_id)
        bids = self.bids_repository.get_by_ids(bids_ids)

        old_winners = list(auction.winners)
        auction.withdraw_bids(bids)
        new_winners = list(auction.winners)

        self.auctions_repository.save(auction)
        for bid in bids:
            self.bids_repository.save(bid)

        self._notify_new_winners(new_winners)
```

Czysta architektura - element #0

```
class Auction:
    def __init__(self, id: int, title: str, bids: List[Bid]):
        self.id = id
        self.title = title
        self.bids = bids

    def withdraw_bids(self, bids: List[Bid]):
        ...

    def make_a_bid(self, bid: Bid):
        ...

    @property
    def winners(self):
        ...
```

Entity

Czysta architektura - element #3

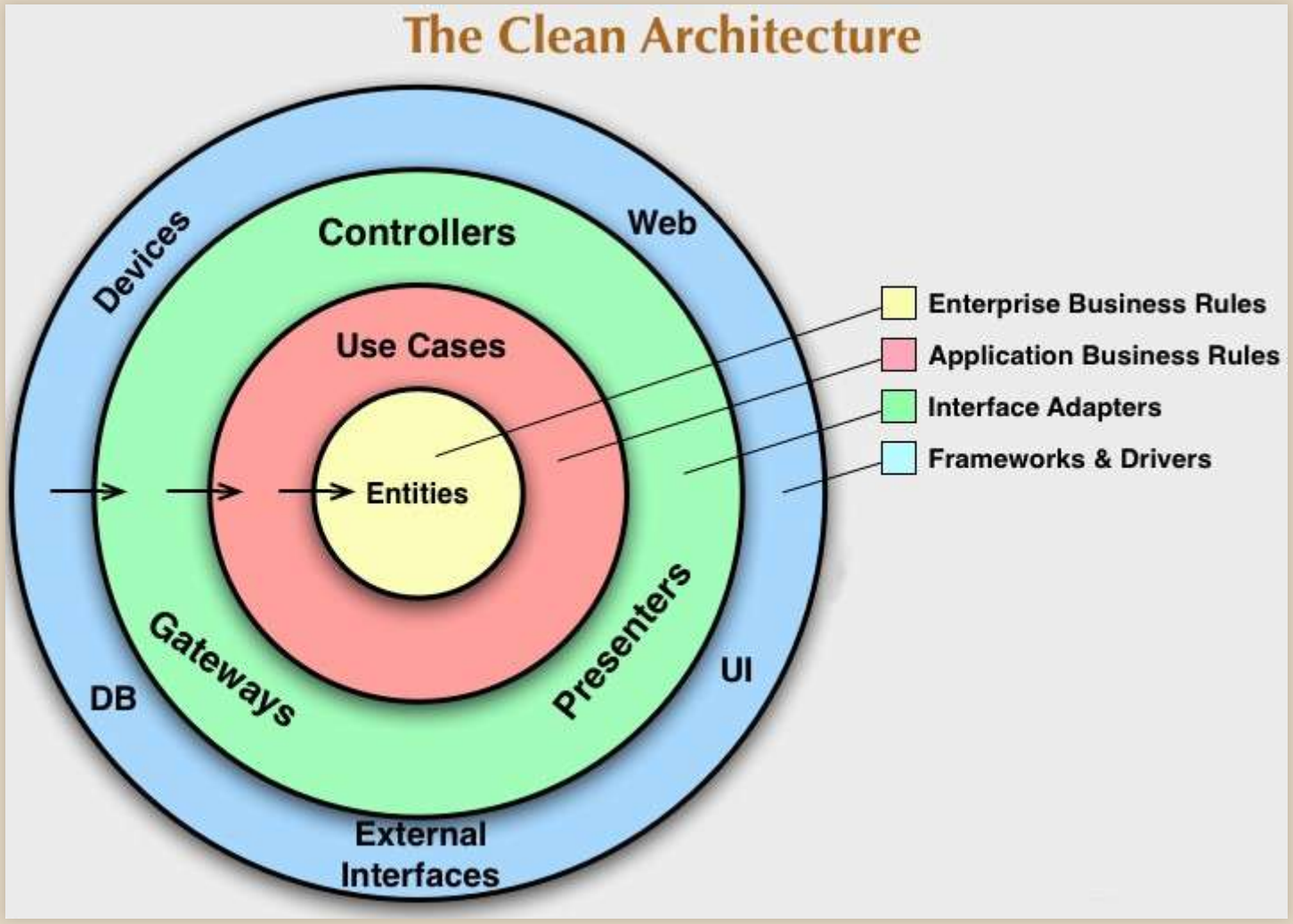
```
class DjangoAuctionsRepo(AuctionsRepo):
    def get(self, auction_id: int) -> Auction:
        auction_model = Auction.objects.prefetch_related(
            'bids'
        ).get(pk=auction_id)

        bids = [
            self._bid_from_model(bid_model)
            for bid_model in auction_model.bids.all()
        ]

        return Auction(
            auction_model.id,
            auction_model.title,
            bids
        )
```

Interface / Port Adapter

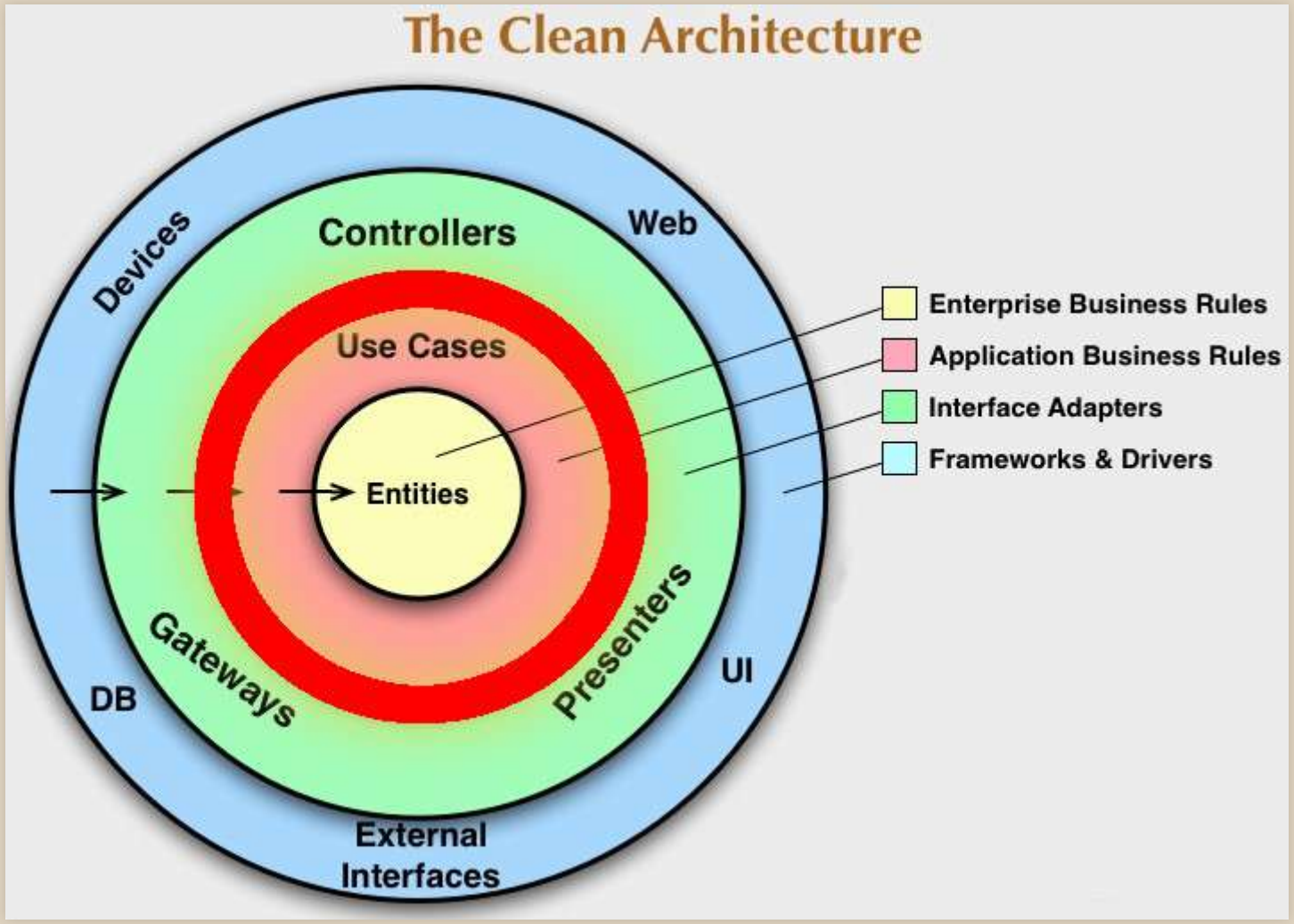
Zostało już tylko zawołać UseCase z Django

Zostało już tylko zawołać UseCase z ~~Django~~ dowolnego frameworka

Wszystko razem na jednym obrazku



Wszystko razem na jednym obrazku 2



Na co uważać?

- więcej kodu (type hinty pomagają)
- "przerzucanie" danych z jednych obiektów do drugich
- walidacja?
- uwaga na overengineering

Kiedy się opłaca?

- Nowy projekt - odraczanie decyzji
- testowalność
- skomplikowana domena

Co dalej mogę z tym zrobić?

- CQRS
- Event Sourcing
- Domain Driven Design

Pytania?

Więcej informacji

https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

Clean Architecture: A Craftsman's Guide to Software Structure and Design

Clean Architecture Python (web) apps - Przemek Lewandowski

Software architecture chronicles - seria blog postów

Boundaries - Gary Bernhardt

Przykładowy projekt w PHP (blog post)

Przykładowy projekt w PHP (repo)

Przykładowy projekt w .NET (repo)

Przykładowy projekt w Pythonie (repo)