Layered Architecture

Why does architecture matters?

"To make system maintainable" - sure, but what does it mean?

System is maintainable when it's easy to make changes and add features.

What determines the ease of making changes and adding features?

- Time needed to understand what is happening and where
- Time needed to determine how to make a change
    - ...without breaking the rest of the system
- Time needed to implement a solution (usually the fastest step)
- Time needed for testing and debugging (usually the slowest step)
    - How many (logic) elements must be touched? It's exponential growth
    - Includes errors found after deployment and fixing them
- Time needed to make a code review
    - Directly proportional to effort put in previous steps

It all comes down to complexity, which is the nemesis of every developer.

Cognitive load

- Every piece of information we need to keep in mind adds to fatigue.
- Idioms, patterns, frameworks - after getting familiar - lower the cognitive load.
- Unexpected, custom solutions are the most expensive and tiring.

How does architecture solve the problem?

Logical structure

- It's easy to find things and recognize where the new element should belong.
- Areas of responsibility are grouped together and clearly separated.
- Frameworks shine here, it's probably the main reason to use them.

Levels of abstraction

- Good abstraction is **the** solution to tame complexity and reduce cognitive load.
- ...but wrong abstraction does just the opposite.
- Duplication is far cheaper than the wrong abstraction. Rule of three.

How to know if our abstraction is a good one?

- It's as close to the real world as possible.
- It not getting complicated over time.
- It's on a single level of abstraction.

Layered Architecture

- Dividing complicated systems into layers is a popular concept in software.
- It helps achieve decoupling, hide complexity and facilitates re-usability.
- Good examples are OSI model or TCP/IP protocol.
- When layers are well defined it's easy to do the right thing.
- In high-level programming there are multiple approaches, more or less similar.
- Each layer represents different level of abstraction.

Domain-driven design (DDD) style

Infrastructure / UI

1. Communication with external world e.g. database, API, CLI, email.

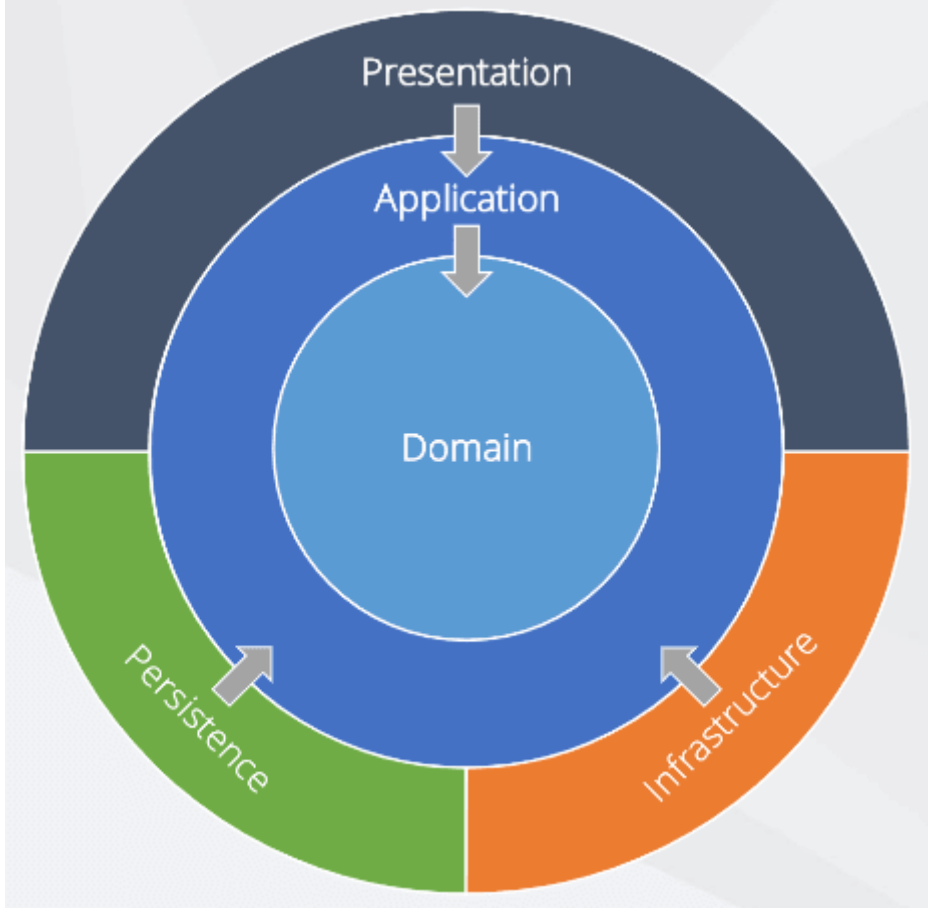
Application

1. Similar across apps, usually delivered by framework e.g. authentication, HTTP.
2. "Is it the reason we're building this app for?" - if not it's Application Layer.

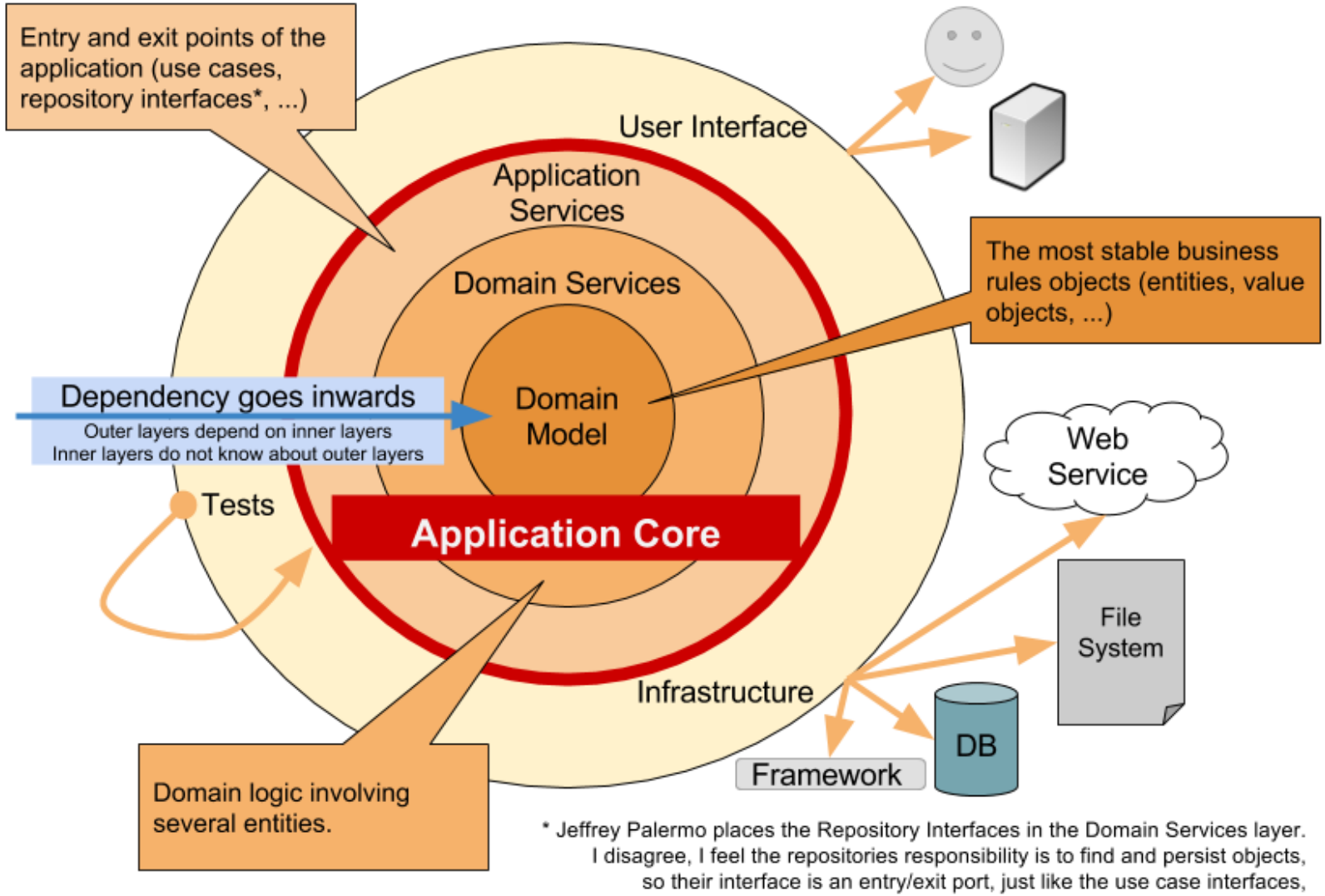
Domain

1. Business logic - reason why we even build this app - all should be here.
2. In ideal scenario domain logic should be readable even by non-developers.
3. It should be easy to test, even without mocking. (Django?)
4. Framework agnostic.

Domain-driven design (DDD) style

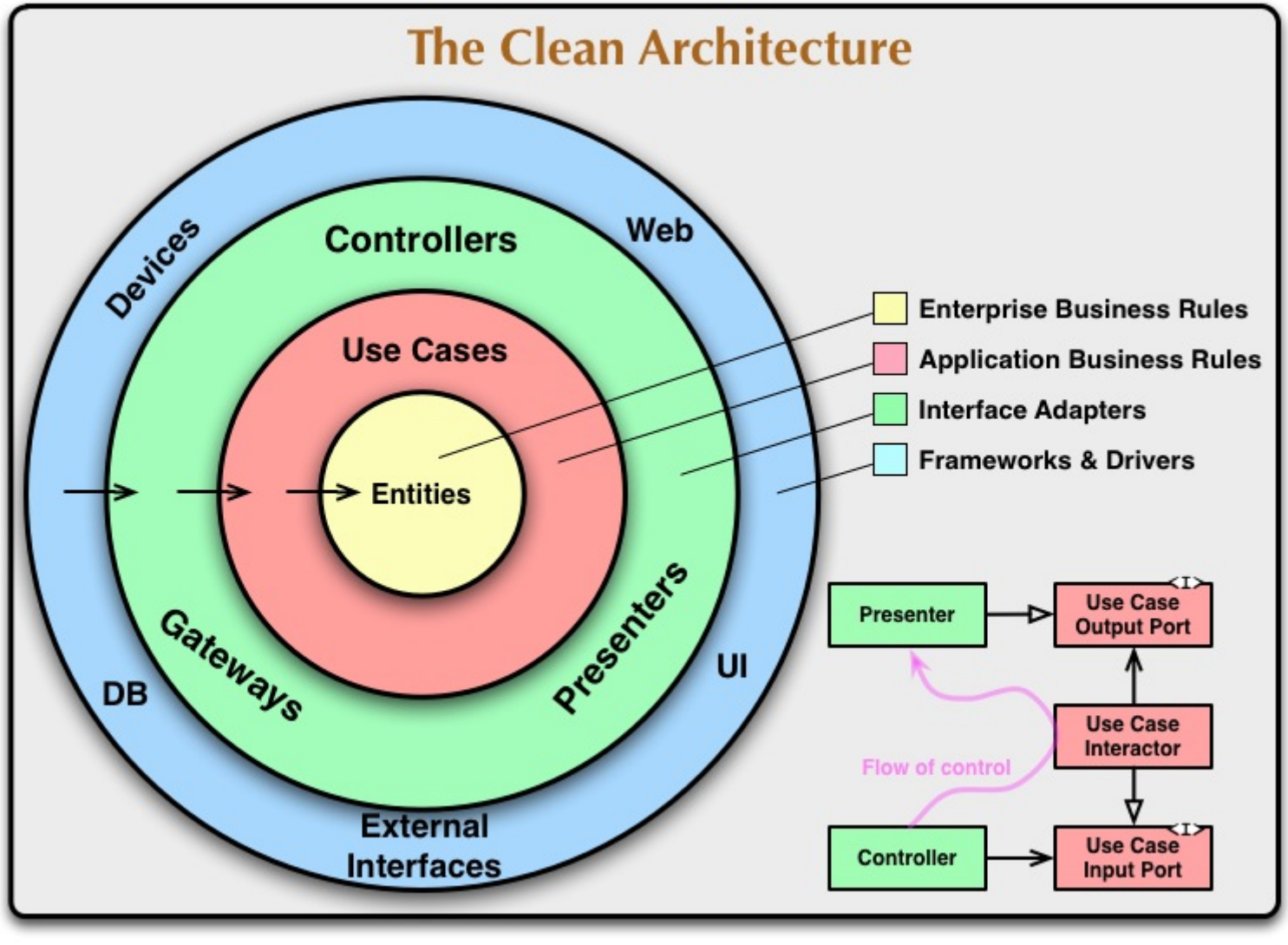


Onion Architecture

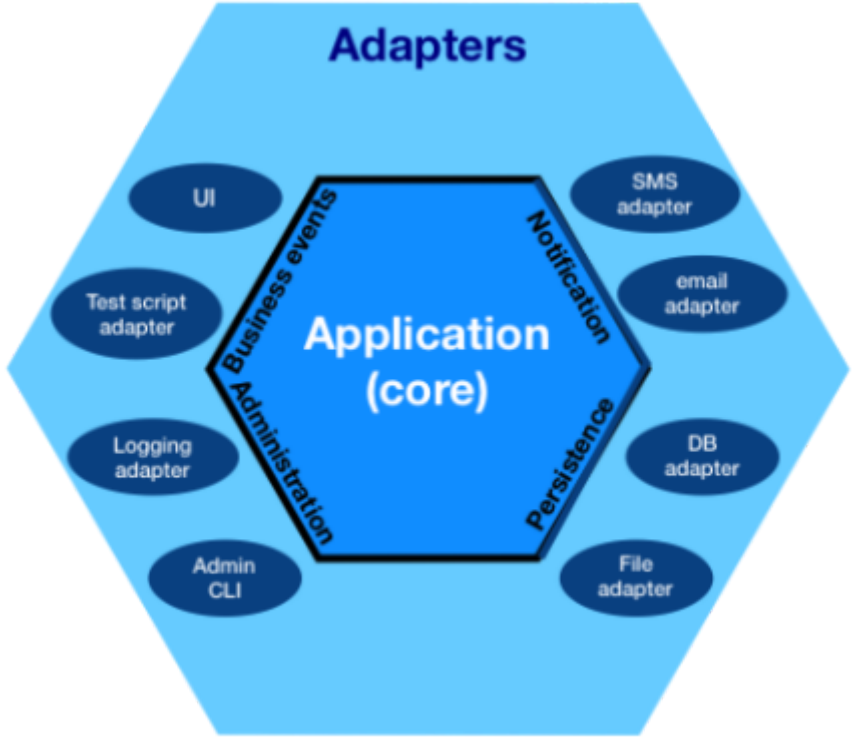


* Jeffrey Palermo places the Repository Interfaces in the Domain Services layer. I disagree, I feel the repositories responsibility is to find and persist objects, so their interface is an entry/exit port, just like the use case interfaces, and therefore should not be known by our domain layers.

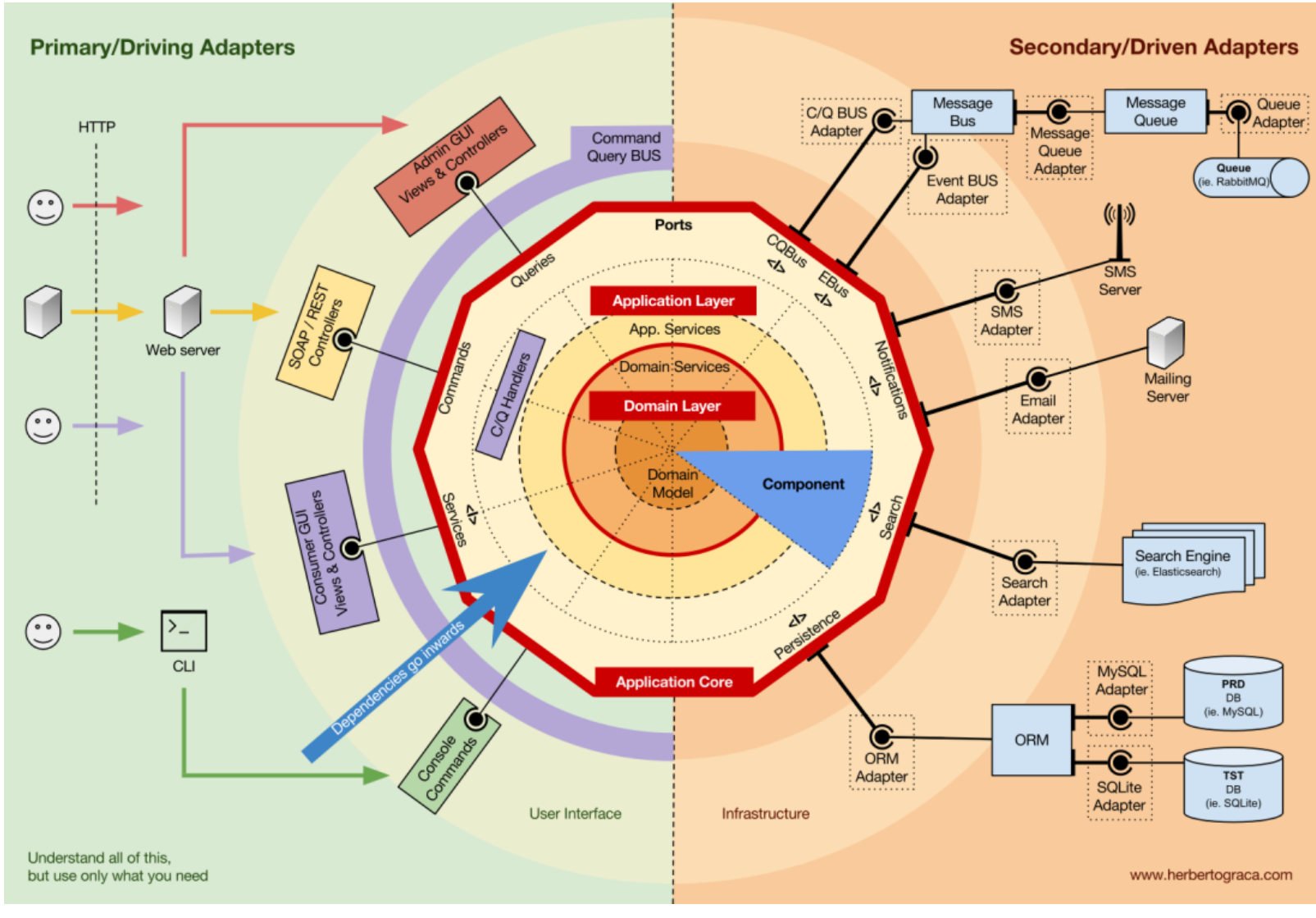
Clean Architecture



Hexagonal Architecture



Explicit Architecture



Examples

Model-View-Controller (MVC)

1. Model = Domain (+ Infrastructure)
2. Controller = Application
3. View = UI
4. (Framework = Application + Infrastructure)
5. (In Django View = Controller)

Repository pattern

1. Separates infrastructure from the rest of the system.
2. It's easy to change storage, add cache etc.

Inversion of Control (IoC)

1. Implemented mostly as Dependency Injection which is useless in Python.
2. ...but as a principle should be used even implicitly.

In [ ]:
```python
class BrandSafetyPoliceman:
    def __init__(self, advertisers_repository):
        self.advertisers_repository = advertisers_repository

    def check_brand_safety(self, advertiser_id):
        advertiser = self.advertisers_repository.get(advertiser_id)
        ...
```

Real-life example

In [ ]:

```
def get_brand_safety_status(request: HttpRequest) -> HttpResponse:
    if request.headers['Content-Type'] != 'application/json':
        raise ContentTypeError()

    data = request.json()
    with database.cursor() as cur:
        cur.execute(
            'SELECT * FROM advertisers WHERE id = %s',
            (data['advertiser_id'],)
        )
        advertiser = cur.fetchone()

    bid_request = data['bid_request']
    if bid_request['host'] in advertiser.host_blacklist:
        return HttpResponse({'status': False})

    return HttpResponse({'status': True})
```

In [ ]:
```
# infrastructure
def get(advertiser_id):
    with database.cursor() as cur:
        cur.execute(
            'SELECT * FROM advertisers WHERE id = %s',
            (data['advertiser_id'],)
        )
        return cur.fetchone()
```

In [ ]:
```
# application
def get_brand_safety_status(request: HttpRequest) -> HttpResponse:
    if request.headers['Content-Type'] != 'application/json':
        raise ContentTypeError()

    data = request.json()
    advertiser = advertisers_repository.get(data['advertiser_id'])
    bid_request = data['bid_request']

    status = check_brand_safety(advertiser, bid_request)
    return HttpResponse({'status': status})
```

In [ ]:
```
# domain - pure function
def check_brand_safety(advertiser, bid_request) -> bool:
    return bid_request['host'] in advertiser.host_blacklist
```

Links

1. "The Wrong Abstraction" - https://www.sandimetz.com/blog/2016/1/20/the-wrong-abstraction (https://www.sandimetz.com/blog/2016/1/20/the-wrong-abstraction)
2. "The Clean Architecture" - https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html)
3. "DDD, Hexagonal, Onion, Clean, CQRS, … How I put it all together" - https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/ (https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/)


LAYERS!
ONIONS HAVE LAYERS. SOFTWARE HAS LAYERS!
memegenerator.net