Microservices in Django

by Biola Oyeniyi

@Beee_sama

The Microservice Architecture

Microservice is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities.

@Beee_sama

The Microservice Architecture

The microservice architecture enables the continuous deployment of large, complex applications. It also enables an organization to evolve its technology stack.

HEADS UP!

This talk isn't about the pros or cons of using microservices, for that check the link below.

https://microservices.io

@Beee_sama

The Django Web Framework

@Beee_sama

The Django Web Framework

Django makes it easier to build better Web apps more quickly and with less code

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design

Django takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel.

@Beee_sama

Useful features in Django for microservices.

@Beee_sama

Useful features in Django for microservices.

1 **Inbuilt scaffolding functionalities**

A useful django template for a simple service

https://github.com/gbozee/django-micro

2 **The Django Admin**

NB: Using the django admin is a short term solution depending on the size of the team.

@Beee_sama

Useful features in Django for microservices.

The Assumption is that all services share the same database and these databases are relational databases (PostgresQL)

> Django's multi db support provides assistance in this scenario.

> The management command `inspectdb` makes it possible to generate a model from a legacy database.

You would need to extend the generated models to support foreign key relationships.

@Beee_sama

Adapting Django's functionalities

@Beee_sama

Adapting the login_required decorator

```
def login_required(profile=False):
  def decorator(view_func):
    @wraps(view_func)
      def _wrapped_view(request, *args, **kwargs):
        auth_header = request.META.get("HTTP_AUTHORIZATION")

        if not auth_header:
          logger.error(
            "Failed authorization",
            exc_info=True,
            extra={
# Optionally pass a request and we'll grab any information we can
            "request": request
                  },)
              return JsonResponse(
                  {
                  “errors": "Ensure to set the Authorization Header
with your user token"
                  }, status=403,
              )
```

@Beee_sama

```
token = auth_header.replace("Bearer", "").replace("Token", "").strip()
            kwar = [token]
            kw = {}
            if profile:
                kw.update({"cv_details": True})
            if request.method == "POST":
                data = json.loads(request.body)
                request.cleaned_body = data
            result = authenticate(*kwar, **kw)
            if not result:
                return JsonResponse(
                    {"errors": "This token is either invalid or
expired"}, status=403
                )
            request.session["user_id"] = result["user_id"]
            request.user_id = result["user_id"]
            if profile:
                request.user = result.get("personal-info")
            return view_func(request, *args, **kwargs)

        return _wrapped_view

    return decorator
```

@Beee_sama

Deviating from Django

@Beee_sama

Deviating from Django

Non-strict apps structure.

Shared libraries instead of reusable apps

Shared libraries can be installed from a repository that can either be public or private.

e.g public repository (# requirements.txt)

```
-e git+https://github.com/gbozee/django-ravepay.git@master#egg=ravepay
```

e.g private repository (# requirements.txt)

```
-e git+https://
username:password@<repository>.git#egg=<repo_folder_consisting_of_code>
```

@Beee_sama

5 Things to think about when choosing a microservice architecture

#opinionated

@Beee_sama

5 things to consider for microservice architecture

1. Repository structure, a mono-repo or single repo per service

2. Leveraging the ORM instead of depending on third party apps for basic functionalities.

   * The point of a micro service architecture is to be flexible.

3. Taming inbuilt apps and middle wares

@Beee_sama

5 things to consider for microservice architecture

4 Continuous integration and delivery for each services.

5 Django is a good starting point but not the end goal.

@Beee_sama

Thank you!

@Beee_sama

gbozee@gmail.com