DVHB Hybrid Documentation

Alexander Malev

Nov 24, 2018

Contents

Package to create hybrid django and asyncio applications.

CHAPTER 1

Features

- Defines your database schema using Django models.
- Uses Django migration to propagating changes you make to your models in your database schema.
- Uses powerful Django administration to manage your application data.
- Allows easily creating a REST API based on aiohttp and aiohttp-apiset.
- Mailer application.

CHAPTER 2

Documentation contents

2.1 Tutorial

You can use this tutorial to start a new project or integrate aiohttp application into your Django project.

2.1.1 Start project

Start you project using *django-admin startproject* command:

```
$ mkdir tutorial
$ cd tutorial
$ pyvenv venv
$ . venv/bin/activate
$ pip install django aioworkers
$ django-admin startproject tutorial .
```

Project structure will look like this:

```
tutorial/
├── manage.py
├── tutorial
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── venv
```

Install dvhb-hybrid:

```
$ pip install git+https://github.com/dvhb/dvhb-hybrid
```

2.1.2 Create an application

Generally, Django project consists of a few applications.

For instance, we need to create a *GET /users* method which returns a list of the application's users. So create a *users* django application and place it as a subpackage of the *tutorial* package:

```
$ mkdir tutorial/users
$ django-admin startapp users tutorial/users
```

Now we can declare the *User* model in *users/models.py* by extending *django.contrib.auth.models.AbstractUser*:

```
from django.contrib.auth.models import AbstractUser


class User(AbstractUser):
    pass
```

It's necessary to create a *tutorial/users/amodels.py* with *User* model based on *dvhb_hybrid.amodels.Model*. It needs to work with the model from asyncio. This model will be loaded by *dvhb_hybrid.amodels.AppModels* and used to access the data from async functions.

```
from dvhb_hybrid.amodels import Model

from .models import User as DjangoUser

class User(Model):
    # Create SQLAlchemy table based on Django table
    table = Model.get_table_from_django(DjangoUser)
```

Let's add an async function which will be using our model in module *tutorial/users/views.py*:

```
async def get_users(request):
        return await request.app.m.user.get_list(fields=['username', 'email'])
```

Our aiohttp application uses *SwaggerRouter* from *aiohttp_apiset* to build application routes and we need to specify our endpoint as swagger spec here *tutorial/users/users_api.yaml*:

```
paths:
  '':
    get:
      $handler: tutorial.users.views.get_users
      tags:
        - user
      summary: Users list
      description: Returns list of users

      produces:
        - application/json

      responses:
        200:
        description: OK
```

2.1.3 Configuring of project

You can configure the project in any way you like. But we suggest to use common config for your Django Admin and aiohttp application. It allows you to avoid duplication of parameters.

For instance, an application can be configured using *load_conf* function from *aioworkers.config*. Create a *config.yaml* in the base folder and specify the database configuration and some other parameters required by *aioworkers*:

Load configuration to *settings.py* and use it to build Django *DATABASES*:

```
from dvhb_hybrid.config import load_conf, db_to_settings

...

config = load_conf(os.path.abspath(os.path.join(BASE_DIR, 'config.yaml')))

...

DATABASES = db_to_settings(config.databases, BASE_DIR)
```

Add our *users* application to *settings.py*:

```
...

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'tutorial.users',
]

AUTH_USER_MODEL = 'users.User'

...
```

Create DB, make the migrations and migrate it:

```
$ createdb tutorial
$ python manage.py makemigrations
$ python manage.py migrate
```

Now you can create a super user for your application:

```
$ python manage.py createsuperuser --username admin --email admin@example.com
```

Run Django Administration and login here using the username and password specified in previous step:

```
$ python manage.py runserver
```

2.1.4 Create an asyncio application

Add *tutorial/api.yaml* with specification from *users* application:

```yaml
swagger: '2.0'

basePath: /api

info:
  title: TUTORIAL API
  version: '1.0'
  description: API  1.0

paths:
  /users:
    - $include: users/users_api.yaml
```

Create *tutorial/app.py*:

```python
import os

import aiopg.sa
import django
from aiohttp_apiset import SwaggerRouter
from aiohttp_apiset.middlewares import jsonify
import aioworkers.http

from dvhb_hybrid.amodels import AppModels

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "tutorial.settings")
django.setup()

import tutorial
AppModels.import_all_models_from_packages(tutorial)


class Application(aioworkers.http.Application):
    def __init__(self, *args, **kwargs):
        router = SwaggerRouter(search_dirs=['tutorial'])
        kwargs['router'] = router

        kwargs.setdefault('middlewares', []).append(jsonify)

        super().__init__(**kwargs)

        router.include('api.yaml')

        cls = type(self)
        self.on_startup.append(cls.startup_database)
        self.on_cleanup.append(cls.cleanup_database)

    async def startup_database(self):
        dbparams = self.config.databases.default
        self['db'] = await aiopg.sa.create_engine(**dbparams)
        self.models = self.m = AppModels(self)

    async def cleanup_database(self):
        self['db'].close()
        await self['db'].wait_closed()
```

(continues on next page)

(continued from previous page)

```
So now we can run an application:
```

```
$ python -m aioworkers -c config.yaml
```

This will run the application on *localhost:8080* with Swagger UI here *http://localhost:8080/apidoc/*.

Test API via curl:

```
$ curl -X GET http://localhost:8080/api/users
[{"username": "admin", "email": "admin@example.com"}]
```

Final project structure will look like this:

```
tutorial/
├── config.yaml
├── manage.py
├── tutorial
│   ├── __init__.py
│   ├── api.yaml
│   ├── app.py
│   ├── settings.py
│   ├── urls.py
│   ├── users
│   │   ├── __init__.py
│   │   ├── amodels.py
│   │   ├── apps.py
│   │   ├── migrations
│   │   │   ├── 0001_initial.py
│   │   │   ├── __init__.py
│   │   ├── models.py
│   │   ├── tests.py
│   │   ├── users_api.yaml
│   │   └── views.py
│   └── wsgi.py
```

2.2 History

2.2.1 0.1.0 (2017-04-07)

- Mailer application.

2.2.2 0.0.5 (2016-02-15)

- Tutorial.
- Application example.

2.2.3 0.0.4 (2016-02-07)

2.3 Indices and tables

- genindex
- modindex
- search