Report

Masinde Mtesigwa Masinde 41969

11/9/2019

YAAS

The use cases I mplimented

UC1 Create a user account

UC2 Edit account information

UC3 Create new auction

UC4 Edit the description of an auction

UC5 Browse and search auctions

UC6 Bid

UC7 Ban an auction

UC8 Resolve auction

UC9 Support for multiple languages English/Swahili

UC11 Support Multiple concurent sessions

UC12 RESTFUL webservice

Microservices

In this project I decided to use microservice i.e Docker container to have different services running in defferent containers. In this project the web application and the database they run in separate containers

Docker

Docker is a way to isolate an entire operating system via Linux containers which are a type of virtualization. The important distinction between virtual environments and Docker is that virtual environments can only isolate Python packages. They cannot isolate non-Python software like a PostgreSQL or MySQL database. And they still rely on a global, system-level installation of Python (in other words, on your computer). The virtual environment points to an existing Python installation; it does not contain Python itself.

Install docker with the following commands

- sudo apt update
- sudo apt install apt-transport-https ca-certificates curl software-properties-common
- curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
- sudo add-apt-repository “deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable”
- sudo apt update
- apt-cache policy docker-ce
- sudo apt install docker-ce
- sudo systemctl status docker
- sudo usermod -aG docker ${USER}
- su - ${USER}

Docker-compose Install

Docker is often used with an additional tool, Docker Compose, to help automate commands. Docker Compose makes it easier for users to orchestrate the processes of Docker containers, including starting up, shutting down, and setting up intra-container linking and volumes. Use the following command to install docker-compose, the current version of docker-compose is 3.7 but it is specified only in the docker-compose file. To downlonad and install docker-compose use the following commmands.

- curl -L https://github.com/docker/compose/releases/download/1.25.1-rc1/docker-compose-%60uname -s‘uname -m‘ -o /usr/local/bin/docker-compose ´
- chmod +x /usr/local/bin/docker-compose

Run the project with following commands

This command builds the project from the scratch, meaning that the dockerfile which I have created and the docker-compose file which is YML file will be excuted

- docker-compose up -d –build
- docker-compose exec web python manage.py makemigrations
- docker-compose exec web python manage.py migrate
- docker-compose exec web python manage.py createsuperuser

Admin

http://127.0.0.1:8000/admin

username

admin@yaas.com

password

testpass123

Use the comand docker-compose logs if encoutered with problems

- docker-compose logs

Shutdown the containers

- docker-compose down

fixtures

dumpdata

- docker-compose exec web python manage.py dumpdata auctions –indent 2 -o fixtures/db_auctions.json

load data

- docker-compose exec web python manage.py loaddata fixtures/db_auctions.json

Postgresql

One of the most immediate differences between working on a “toy app” in Django and a production-ready one is the database. Django ships with SQLite as the default choice for local development because it is small, fast, and file-based which makes it easy to use. No additional installation or configuration is required.

Django ships with built-in support for four databases: SQLite, PostgreSQL, MySQL, and Oracle. We'll be using PostgreSQL in this project as it is the most popular choice for Django developers, however, the beauty of Django's ORM is that even if we wanted to use MySQL or Oracle, the actual Django code we write will be almost identical. The Django ORM handles the translation from Python code to the databases for us which is quite amazing.

Custom User Model

I have used AbstractUser https://docs.djangoproject.com/en/2.2/topics/auth/customizing/#django.contrib.auth.models.AbstractUser

So custom user has its own app, I did that with the following command

- docker-compose exec web python manage.py startapp users

User Registration

Django's default username/email/password pattern is somewhat dated these days. It's far more common to simply require email/password for signup and log in.

I am using django-allauth which is the third part application, so logging in I have customized to use only email to make it easier for the user. I am also using third part app for forms which is django-crispy forms

django-allauth config

- AUTH_USER_MODEL = 'users.CustomUser'
- LOGIN_REDIRECT_URL = 'home'
- ACCOUNT_LOGOUT_REDIRECT = 'home'
- ACCOUNT_SESSION_REMEMBER = True
- ACCOUNT_SIGNUP_PASSWORD_ENTER_TWICE = False
- ACCOUNT_USERNAME_REQUIRED = False
- ACCOUNT_AUTHENTICATION_METHOD = 'email'
- ACCOUNT_EMAIL_REQUIRED = True
- ACCOUNT_UNIQUE_EMAIL = True
- SITE_ID = 1
- AUTHENTICATION_BACKENDS = ('django.contrib.auth.backends.ModelBackend', 'allauth.account.auth_backends.AuthenticationBackend',)
- https://django-allauth.readthedocs.io/en/latest/configuration.html
- https://django-crispy-forms.readthedocs.io/en/latest/

In that case I am using login via e-mail and I took out the remember me box option.

- ACCOUNT_SESSION_REMEMBER = True

Like any good third-party package django-allauth comes with its own tests

Email

Custom Confirmation Emails can be seen from terminal using the docker-compose logs command after sign up

Custom Permissions

Logged in users only

Edit

user can edit own auctions

UUID

Using the pk field in the URL of our DetailView is quick and easy, but not ideal for a real-world project. The pk is currently the same as our auto-incrementing id . Among other concerns, it tells a potential hacker exactly how many records you have in your database; it tells them exactly what the id is which can be used in a potential attack.

In this project I have UUID, also I have used uuid4 for the encryption of the auctions

- https://docs.python.org/3/library/uuid.html?highlight=uuid#module-uuid

Cron Job

resolve when logged in as admin

Translation

Translation is done from English to Swahili

Bid

User can bid only to the auctions which are not his/hers

Search

user can search for auctions

Tests

Use the following command to run tests

docker-compose exec web python manage.py test

After adding some special status the two tests fails and that is on AuctionDetailView

API

I implemented the API and the API is searchable. The permissions are allowany but it may conflict with the auctions app, because auctions app has special status permissions.

Search Functionality

I use Q objects for searching complex queries

- https://docs.djangoproject.com/en/2.2/topics/db/queries/#complex-lookups-with-q-objects