Documento de construcción decide-part-chiquito

Integrantes

Juan Jesús Campos Garrido	decide-part-chiquito-1
Alejandro Campano Galán	decide-part-chiquito-1
Antonio Carretero Díaz	decide-part-chiquito-1
Pablo Mera Gómez	decide-part-chiquito-1
Carlos Baquero Villena	decide-part-chiquito-1
David Cortabarra Romero	decide-part-chiquito-1
Alejandro Pérez Vázquez	decide-part-chiquito-2
Juan Carlos Ramírez López	decide-part-chiquito-2
Carmen Ruiz Porcel	decide-part-chiquito-2
Alejandro Santiago Félix	decide-part-chiquito-2
Sergio Santiago Sánchez	decide-part-chiquito-2
María Vico Martín	decide-part-chiquito-2

1. Introducción:

Decide-part-chiquito es un proyecto que trata de extender otro ya existente, Decide, no solo dotándolo de nuevas formas de votación e inicio de sesión, sino también proporcionando una mejor interfaz gráfica, lo que permite al usuario tener una experiencia mucho mejor. Dejando a un lado todo lo relacionado con el usuario final, votaciones e interfaz, también hemos integrado un sistema de auditoría, facilitando enormemente a los desarrolladores la detección de causas de errores.

El documento de construcción permite transmitir, no solo cómo instalar y desplegar un proyecto sino también los posibles fallos, requisitos y posibilidades.

Un proyecto normalmente no es mantenido por el mismo equipo que lo desarrolla, puede incluso ser desarrollado en parte por un equipo y en parte por otro, debido a que en un empresa van variando los trabajadores. Por tanto, un documento como el de construcción permite que diferentes personas dispongan en un único lugar de toda la información necesaria para poder ejecutar un proyecto y continuar su desarrollo, probarlo o simplemente observar.

2. Requisitos del Sistema:

Para el proyecto en cuestión no se necesita un sistema operativo en concreto, puede ser desarrollado en Windows, Mac o Linux, si bien es cierto que en caso de emplear vagrant sí será necesaria alguna distribución Linux.

El proyecto emplea el framework basado en python, Django. Aprovechando el empleo de Python, haremos uso de los entornos virtuales, para evitar conflictos entre versiones tanto de Python como de dependencias/librerías. Para los despliegues podemos emplear tanto Docker como Vagrant. Las dependencias empleadas son las siguientes:

- Django==4.1
- pycryptodome==3.15.0
- djangorestframework==3.14.0
- django-cors-headers==3.13.0
- requests==2.28.1
- django-filter==22.1
- psycopg2==2.9.4
- coverage==6.5.0
- jsonnet==0.18.0
- django-nose==1.4.6
- django-rest-swagger==2.2.0
- selenium==4.7.2
- django-unfold==0.16.0
- locust==2.17.0
- webdriver-manager==4.0.1
- dj-database-url==2.1.0
- pynose==1.4.8
- whitenoise==6.5.0
- gunicorn==21.2.0
- django-livereload-server==0.5
- django-admin-autocomplete-filter==0.7.1
- social-auth-app-django==5.4.0
- django-user-agents==0.4.0
- django-extensions==3.2.3
- django-auditlog==2.3.0
- cryptography==41.0.7
- python-decouple==3.8

3. Configuración del Entorno de Desarrollo:

1) Instalamos Python 3.10 en nuestro equipo, podemos encontrarlo en el enlace.
2) Clonamos el proyecto desde el repositorio de GitHub.
3) En una ventana de terminal, navegamos hasta donde se encuentre la carpeta que acabamos de clonar de GitHub y ejecutamos el comando **./init.sh** para crear el entorno, la base de datos y realizar las migraciones pertinentes.
4) Finalmente para que el programa funcione correctamente se deberá de añadir el archivo .env a **./decide**, **/vagrant/files**, **./docker** el cual es el siguiente.

```
EMAIL_HOST_PASSWORD='hulp rfpq boxy otqa'
EMAIL_HOST_USER='decide202324@gmail.com'
SECRET_KEY='^##ydkswfu0+=ofw0l#$kv^8n)0$i(qd&d&ol#p9!b$8*5%j1+'
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY='591203465714-9ng1mqkp902c0gkjmkun63ctsapvofgp.apps.googleusercontent.com'
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET='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'
```

4. Estructura del Proyecto:

El proyecto se encuentra englobado en la carpeta “decide-part-chiquito”, dentro de la misma tendremos otra carpeta con el nombre “**decide**”, donde encontramos el código fuente del sistema desarrollado. Sigue la estructura básica de un proyecto Django, donde encontramos el proyecto, decide, junto con todas las apps que lo forman. A su vez también contamos con los siguientes directorios:

1) **doc**: contiene los documentos del grupo de desarrollo.
2) **docker**: en esta carpeta encontramos todo lo necesario para el despliegue en contenedores.
3) **vagrant**: en esta carpeta encontramos todo lo necesario para el despliegue en máquinas virtuales.
4) **loadtest**: contiene los archivos de test de locust, para hacer pruebas de carga.
5) **resource**: cuenta con distintas capturas de pantalla del sistema en su estado original.
6) **requirements**: en este fichero encontramos las dependencias del sistema.
7) Cada app y/o proyecto está formado por una serie de archivos o directorios:
    a) **manage.py**: permite gestionar el proyecto, es único por proyecto.
    b) **__init__.py**: permite gestionar los paquetes, por ejemplo, cuando importamos un paquete entero podemos especificar de forma genérica cuál de todos ellos vamos a traernos.
    c) **setting.py**: nos permite gestionar la configuración, es único por proyecto. Contiene por ejemplo, las aplicaciones instaladas, la configuración de la base de datos y declaraciones de rutas estáticas.
    d) **urls.py**: nos permite gestionar el enrutamiento del proyecto.
    e) **asgi.py**: punto de entrada para servidores compatibles tipo ASGI.
    f) **wsgi.py**: igual que el anterior pero para servidores WSGI
    g) **views.py**: permite declarar las vistas de la app
    h) **forms.py**: permite definir formularios mediante código.
    i) **templates**: contiene todas las plantillas html
    j) **static**: es donde encontramos los archivos estáticos, como por ejemplo los CSS.

5. **Guía de Instalación y Configuración:**
    a. **Proporciona instrucciones detalladas sobre cómo instalar y configurar cualquier dependencia de terceros.**

1) Agregamos la dependencia al archivo requirements.
2) La instalamos mediante el comando: **pip install -r requirements.txt**
3) En caso de ser necesario, debemos añadir dicha dependencia a **INSTALLED_APPS** dentro del archivo **settings.py** del proyecto.
4) Ejecutamos la migración de los modelos mediante **python manage.py makemigrations** y **python manage.py migrate** o mediante **./init.sh,** para garantizar que la base de datos cuenta con toda la información necesaria.

6. **Descripción del Script de Construcción:**

El script del proyecto es el siguiente:

```
FROM python:3.10-slim

RUN apt-get update && apt-get install -y --no-install-recommends \
    git libpq-dev gcc libc-dev gcc g++ make libffi-dev python3-dev build-essential && \
    apt-get clean

# RUN pip install gunicorn
# RUN pip install psycopg2
# RUN pip install ipdb
# RUN pip install ipython

WORKDIR /app

RUN git clone https://github.com/Decide-chiquito/decide-part-chiquito.git . && \
    pip install --no-cache-dir -r requirements.txt

WORKDIR /app/decide

# local settings.py
COPY docker-settings.py ./local_settings.py

RUN ./manage.py collectstatic --noinput
RUN find . -mindepth 2 -type d -path '*/static' -exec mv -T {}/. /app/static \;


#CMD ["gunicorn", "-w 5", "decide.wsgi", "--timeout=500", "-b 0.0.0.0:5000"]
```

Por favor, explique con mayor detalle el procedimiento. En contraste con el script predeterminado proporcionado por el proyecto Decide, hemos optado por emplear una imagen de Python en su versión 3.10 en lugar de la 3.9. Asimismo, en vez de utilizar la versión Alpine, hemos optado por la versión Slim, la cual nos permite utilizar comandos de sistemas de distribución Linux como Debian o Ubuntu, lo que facilita la comprensión del script para todo el equipo. Además, nos hemos asegurado de instalar únicamente las dependencias estrictamente necesarias para el correcto funcionamiento de Decide, con el fin de reducir el tiempo de creación de la imagen.

1) Descargamos desde el repositorio de Docker, DockerHub, una imagen de python 3.10
2) Empleamos RUN para ejecutar un comando en el contenedor,descarga la información más reciente sobre los paquetes disponibles en los repositorios configurados en el sistema y luego se realiza la instalación de unos paquetes, para ello hacemos uso de --no-install-recommends para evitar que se guarden en caché, en concreto instalamos: **git**, **libpq-dev**, **gcc**, **libc-dev**, **gcc**, **g++**, **make**, **libffi-dev**,

python3-dev y **build-essential**. Finalmente, se eliminan todos los paquetes descargados previamente que se encuentran en la caché de apt.

3) Establecemos el nuevo directorio de trabajo, en este caso /app, mediante **WORKDIR**.
4) Dentro de este directorio, clonamos el proyecto desde GitHub de tal modo que se encuentre todo en dicho directorio, es decir, no se crea una carpeta extra, esto lo hacemos mediante el punto final, e instalamos las dependencias.
5) Copiamos el archivo settings localizado dentro de la carpeta de Docker, dentro de la carpeta del proyecto, ya que tiene la información necesaria para ejecutar el proyecto, esto lo hacemos mediante **ADD**.
6) Ejecutamos el collectstatic, con el fin de recopilar todos los archivos estáticos y nos aseguramos que todos los estáticos estén correctamente ubicados.

Es importante mencionar que contamos con un archivo docker-compose.yaml, que nos permite mapear puertos, gestionar los volúmenes y las subredes entre otras cosas.

7. **Instrucciones de Uso:**

En caso de querer mapear a un puerto distinto al 8000 en la máquina Host, deberemos gestionarlo en el docker-compose.yaml, al igual que con los volúmenes.

Debemos de instalar Docker, para ello ejecutamos:

1) La actualización de paquetes mediante: **sudo apt update**
2) Instalamos algunos prerrequisitos que permitan a apt usar paquetes a través de HTTPS: **sudo apt install apt-transport-https ca-certificates curl software-properties-common**
3) Añadimos la clave GPG del repositorio oficial de Docker al sistema: **curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg**
4) Añadimos el repositorio de Docker a las fuentes de apt: **echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**
5) Actualizamos la lista de paquetes existente para que se reconozca el repositorio añadido: **sudo apt update**
6) Instalamos Docker: **sudo apt install docker-ce**
7) Verificamos que Docker está instalado, iniciado y el proceso habilitado para iniciarse al arrancar: **sudo systemctl status docker.**

Por defecto, Docker debe usarse con el usuario root o con un usuario entre del grupo docker, que es creado durante la instalación de Docker. Si intentamos utilizar el comando "docker" sin sudo obtendremos un mensaje informativo indicando que no tenemos permisos suficientes.

Ya simplemente desde el directorio de Docker, dentro de decide-part-chiquito, ejecutamos el comando **sudo docker-compose up**, se recomienda utilizar antes el comando **sudo docker-compose build --no-cache** pero no es estrictamente necesario.

8. Manejo de Errores y Solución de Problemas:

Durante la instalación podría darse el caso en el que Docker no sea capaz de descargar la imagen solicitada desde el repositorio, si este fuera el caso, debemos de ejecutar **docker pull <nombreImagen>** con el fin de descargar la misma directamente, acto seguido volvemos a ejecutar docker-compose.

Si por alguna razón se realiza una modificación en los contenedores y la aplicación deja de funcionar, se recomienda utilizar el siguiente comando para limpiar el entorno:

docker rm -f $(docker ps -aq) && docker volume rm decide_static decide_db && docker network rm docker_decide

Este comando eliminará todos los contenedores en ejecución, eliminará los volúmenes asociados con la aplicación y también eliminará la red Docker utilizada. Esto puede ser útil si el error que se está experimentando se debe a que se está utilizando una imagen antigua en lugar de la nueva. Este procedimiento garantiza que se comience desde un estado limpio, lo que puede ayudar a solucionar problemas relacionados con versiones antiguas de imágenes o configuraciones obsoletas.

En caso de que la terminal nos indique que el puerto de la máquina Host ya está en uso, podemos cambiarlo dentro del archivo docker-compose.yaml, concretamente en **nginx ports**, sustituyendo el puerto ocupado por otro que esté libre.

Si, habiendo construido el proyecto, no nos permite acceder, debemos de comprobar que nuestra ip de acceso esta añadida en **CSRF_TRUSTED_ORIGINS** dentro del archivo settings.py que encontramos en la carpeta Docker.

En caso de que por algún motivo no se hayan realizado las migraciones, ejecutamos los comandos: **python manage.py makemigrations** y **python manage.py migrate**.

9. Pruebas y Validación:

En primer lugar, si estamos construyendo el sistema con Docker, y damos por hecho que ya se ha construido, deberíamos de poder observar tres contenedores mediante el comando **docker ps**, en caso contrario algo ha ido mal.
Para comprobar que el sistema se ha construido correctamente, podemos lanzar el servidor, si no está lanzado aún, y trata de acceder a él, mediante http://localhost:8000.

A su vez, deberíamos de ejecutar, dentro de la carpeta decide, python manage.py test, lo que nos permitirá realizar todos los test presentes en la aplicación.