Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)

Implementación patrón Singleton

En el contexto del proyecto la gestión de la base de datos se realiza a través del framework de Django.

La configuración de la conexión de Django a la base de datos se encuentra en la ruta Proyecto/Conedus/settings.py, donde encontramos la línea DATABASES como:

```python
DATABASES = {
   "default": {
      "ENGINE": os.environ.get("SQL_ENGINE", "django.db.backends.sqlite3"),
                  "NAME":   os.environ.get("SQL_DATABASE",   os.path.join(BASE_DIR,
'db.sqlite3')),
      "USER": os.environ.get("SQL_USER", "myuser"),
      "PASSWORD": os.environ.get("SQL_PASSWORD", "myuserpassword"),
      "HOST": os.environ.get("SQL_HOST", "localhost"),
      "PORT": os.environ.get("SQL_PORT", "3306"),
   }
}
```

de forma que se utilizan variables de entorno para configurar los parámetros de conexión del proyecto a la base de datos.

Luego, estas son utilizadas en un entorno en Docker, definido en el archivo docker-compose.yml en la raíz del proyecto, de forma que este describe los servicios que componen la aplicación. En particular, define dos contenedores, uno de los cuales corresponde a la aplicación, mientras el otro contendrá la ejecución de la base de datos de la siguiente manera:

```
version: '3.8'

services:
```

```
db:
 image: mysql:8.0
 restart: always
 command: mysqld --default-authentication-plugin=mysql_native_password
 container_name: mysql_db
 env_file:
  - ./.env.dev
 environment:
  - MYSQL_ROOT_PASSWORD=${SQL_ROOT_PASSWORD}
  - MYSQL_USER=${SQL_USER}
  - MYSQL_PASSWORD=${SQL_PASSWORD}
  - MYSQL_DATABASE=${SQL_DATABASE}
 ports:
  - '3306:3306'

web:
   container_name: django-web
   build:
     context: ./Proyecto
     dockerfile: Dockerfile
   command: python manage.py runserver 0.0.0.0:8000
   volumes:
     - ./Proyecto/:/usr/src/app/
   env_file:
     - ./.env.dev
   ports:
     - 8000:8000
   depends_on:
     - db
   links:
     - db
```

De esta forma, al realizarse una conexión al proyecto, también se realizará una única conexión a la base de datos, la cuál será utilizada por el framework de Django, re-utilizándola para cada petición.

Implementación de patrón Memento

El patrón Memento se utiliza para capturar y externalizar el estado interno de un objeto sin violar su encapsulación, de modo que el objeto pueda ser restaurado a este estado más tarde.
En el caso de este proyecto se podría implementar al momento de crear y editar **preguntas,** de forma que se puedan revertir cambios, descartando modificaciones

previas, pues una pregunta puede tener varios detalles que modificar. Con esto, se tendría una copia de todos los atributos de una pregunta

De esta forma se requerirían tres roles:
El originador, el cuál sería la pregunta “original” cuyo estado tenemos que almacenar, y que tendría definidos dos métodos: guardar() y restaurar(memento), de forma que el primero crea un memento con el estado actual del originador, y el segundo restaura su estado a partir de un memento.
el memento, el cuál almacenará el estado actual del originador y que será inmutable (garantizando una copia fiel previo a modificar la pregunta), y
un cuidador, el cuál define cuándo se debe crear un memento y almacena los mementos creados a partir del originador como una pila, de tal modo que el memento más reciente será cabeza de la pila, con dos métodos encargados de utilizar los métodos del originador para que, al guardar un memento, este se agregue a la pila y, para restaurar un memento, se llame al método restaurar del originador con la cabeza de la pila como argumento.

Así, se tendrá un historial de modificaciones de una pregunta, de forma que la pregunta se puede modificar, y retornar a versiones anteriores en caso de alguna modificación erronea durante la edición.