Proyecto de Software

Temas de hoy

- Repaso Sesiones
- Repaso Acceso a Bases de Datos.
- ORM

Repasando: Manejo de sesiones

- ¿Qué es una sesión?
- ¿Porqué son necesarias?
- ¿Qué son las cookies?
- ¿Para qué sirven las cookies?
- ¿Dónde se alojan las cookies?
- ¿Dónde se aloja la sesión?

De nuevo: ¿Dónde se aloja la sesión en flask?

- Depende...
- Tradicionalmente (en lenguajes como PHP) la sesión es almacenada en un archivo en el servidor, el cliente guarda una cookie que SÓLO posee el sessionID para identificarla.

Sesiones en flask

- Por defecto Flask usa sesiones basadas en cookies (**session cookie**).
- La información de sesión se almacena **en el cliente** en una cookie firmada con una **secret key**.
- Cualquier modificación a la cookie queda invalidada por su firma. Pero es **visible en todo momento** en el cliente.
- No es aconsejable guardar información sensible en una session cookie.

Veamos una de estas sesiones en http://localhost:5000/iniciar_sesion decodificadas con https://github.com/noraj/flask-session-cookie-manager

Sesiones en flask - Flask-Session

- Flask posee extrensiones como **Flask-Session** que permiten un mejor manejo de las sesiones.
- Con Flask-Session podemos elegir diferentes lugares donde almacenar la sesión en el servidor:
  - redis
  - memcached
  - **filesystem**
  - mongodb
  - sqlalchemy
- Instalamos: **poetry add Flask-Session**.

Uso de Flask-Session

```
from flask_session import Session
# Configuración inicial de la app
app = Flask(__name__)
app.config.from_object(Config)
#Server Side session
app.config['SESSION_TYPE'] = 'filesystem'
Session(app)
```

- Modifiquenos la app y veamos de nuevo... http://localhost:5000/iniciar_sesion

Referencias:

- Sesiones en flask: https://overiq.com/flask-101/sessions-in-flask/
- Flask-Session: https://flask-session.readthedocs.io/en/latest/

Accediendo a Bases de Datos

Lenguaje SQL (Structured Query Language)

- Sentencias insert, update, select, etc....
- Ejemplos:
  - select * from tabla where condición
  - insert into tabla (campos) values (valores)
  - update tabla set campo1='valor1' where condición

Importante

MySQL, PostgreSQL: motores de base de datos

SQL: lenguaje de consulta

pgAdmin

- Interfaz de Administración de la Base de Datos PostgreSQL
- Podemos exportar e importar a varios formatos

Acceso a BBDD – PostgreSQL

- Vamos a acceder a través de psycopg2.

```
sudo apt-get install libpq-dev
```

```
poetry add psycopg2-binary
```

Acceso a BBDD – PostgreSQL

```
import psycopg2, psycopg2.extras

class Issue:
    def getAll(self):
        conn = psycopg2.connect(
            host="localhost",
            database="proyecto_db",
            user="proyecto_db",
            password="proyecto_db")

        cur = conn.cursor(cursor_factory=psycopg2.extras.RealDictCursor)

        cur.execute('select * from issues')
        issues = cur.fetchall()
        print(issues)
        cur.close()
        conn.close()
        return issues

    def insert (self, issue):
        conn = psycopg2.connect(
                host="localhost",
                database="proyecto_db",
                user="proyecto_db",
                password="proyecto_db")

        cur = conn.cursor()
        cur.execute(f"insert into issues (\"user\", title, description, status) values ('{issue['user']}', '{issue['title']}', '{issue['description']}', '{issue['status']}')")
        conn.commit()
        cur.close()
        conn.close()
```

Acceso a BBDD – PostgreSQL

```
# src/core/db.py

import psycopg2, psycopg2.extras

def get_conn():
    conn = psycopg2.connect(
        host="localhost",
        database="proyecto_db",
        user="proyecto_db",
        password="proyecto_db")

    return conn
```

Acceso a BBDD – PostgreSQL

- Antes de continuar, usamos config.py

```
# src/web/config.py

....

class DevelopmentConfig(Config):
    """Development configuration."""

    DEBUG = True
    DB_SERVER = "localhost"
    DB_DATABASE = "proyecto_db"
    DB_USER = "proyecto_db"
    DB_PASSWORD = "proyecto_db"
    DB_PORT = "5432"
    DATABASE_URI = f"postgresql://{DB_USER}:{DB_PASSWORD}@{DB_SERVER}:{DB_PORT}/{DB_DATABASE}"

....
```

Acceso a BBDD – PostgreSQL

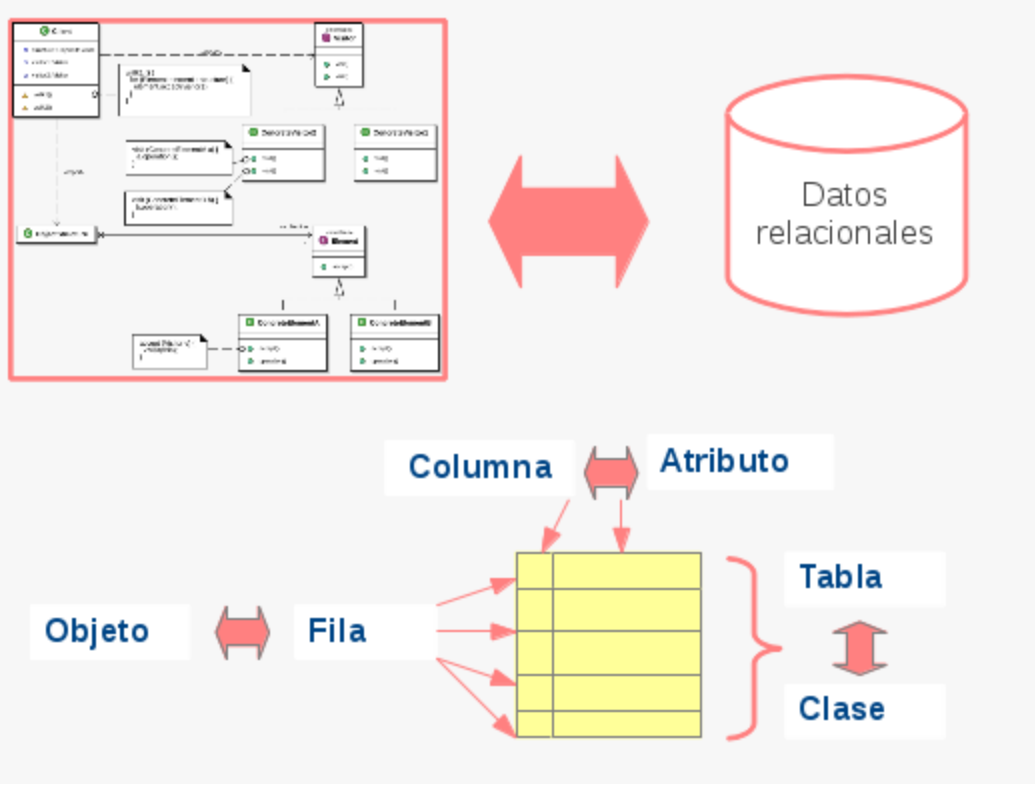
- Antes de continuar, usamos config.py

```
# src/core/db.py

import psycopg2, psycopg2.extras
from flask import current_app

def get_conn():
    conn = psycopg2.connect(
        host=current_app.config.get("DB_SERVER"),
        database=current_app.config.get("DB_DATABASE"),
        user=current_app.config.get("DB_USER"),
        password=current_app.config.get("DB_PASSWORD"))

    return conn
```

Ahora bien ...

- ¿Qué pasa si queremos migrar de motor de BDD?
- ¿Qué pasa si queremos tener múltiples BDD conectadas?

Una vuelta de rosca más -> ORM



ORM - Object-Relational Mapping

- Mapeo de objetos a base de datos relacionales.
- Permite acceder a una base de datos relacional como si fuera orientada a objetos.
- Transforma las llamadas a los objetos en consultas SQL, que permiten independizar el motor de BD utilizado en la aplicación.
- De acuerdo a la implementación se utiliza una sintaxis diferente.

¿Por qué?

- BBDD relacionales
  - Datos escalares: números, cadenas de texto, etc...
- Aplicaciones orientadas a objetos.
  - Objetos con propiedades y métodos.
- ¿Problema?
  - Convertir los objetos en datos escalares para grabar los datos en la base de datos y volver a convertirlos en objetos al recuperarlos.

ORM

- Ventajas
  - Abstracción a la BBDD.
  - Reutilización de código.
- Desventajas
  - Otra capa intermedia.
  - Otro lenguaje de consulta.
- Algunos ORMs
  - SQLAlchemy
  - Peewee
  - Django ORM

SQLAlchemy: agregamos una capa de abstracción

- - Ver: SQLAlchemy
- Herramientas de Abstracción y ORM (mapea objeto a relacional)
- Múltiples motores de bases de datos: https://docs.sqlalchemy.org/en/14/dialects/index.html

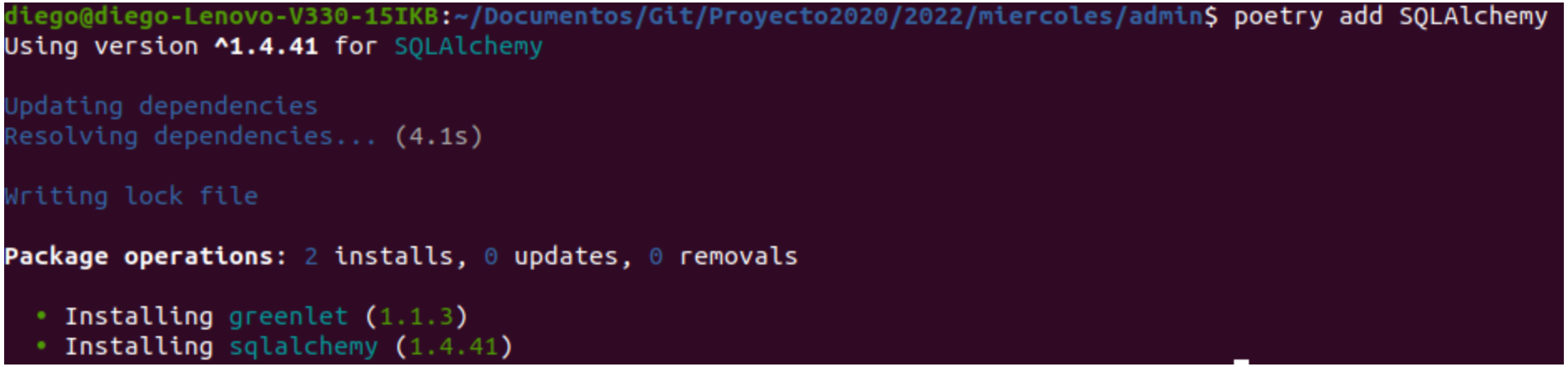
Comparando

- A mayor abstracción menos control .
- Al abstraernos es posible que se compliquen algunas funciones propias del motor que queremos usar.
- ¿La abstracción es necesaria?

ORM - SQLAchemy

- Agregamos SQLAlchemy

```
poetry add SQLAlchemy
```



Ejemplo SQLAlchemy

```
# src/core/db.py

from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import scoped_session, sessionmaker

engine = create_engine(f"postgresql://proyecto_db:proyecto_db@localhost:5432/proyecto_db", echo=True, future=True)

db_session = scoped_session(sessionmaker(autocommit=False,
                                         autoflush=False,
                                         bind=engine))
Base = declarative_base()
Base.query = db_session.query_property()


def init_db():
    # import all modules here that might define models so that
    # they will be registered properly on the metadata.  Otherwise
    # you will have to import them first before calling init_db()

    Base.metadata.create_all(bind=engine)
```

Ejemplo SQLAlchemy

```python
# src/core/board/issue.py

from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import String
from src.core.db import Base

class Issue(Base):
    __tablename__ = "issues"
    id = Column(Integer, primary_key=True)
    title = Column(String, nullable=False)
    description = Column(String, nullable=False)
    status = Column(String, nullable=False)
    user = Column(String, nullable=False)

    def __init__(self, title=None, description=None, status=None, user=None):
        self.title = title
        self.status = status
        self.description = description
        self.user = user

    def __repr__(self):
        return f"Issue(id={self.id!r}, title={self.title!r})"
```

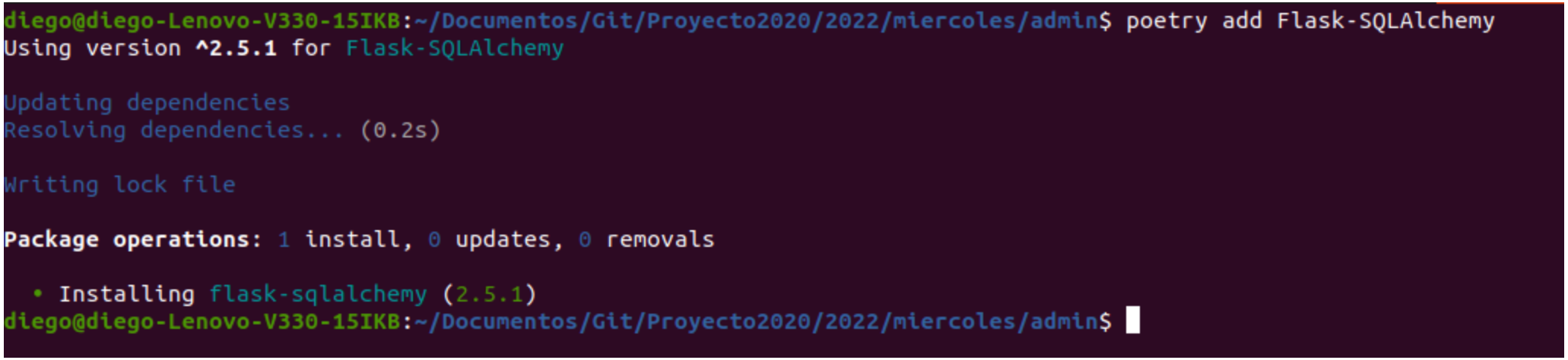
Ejemplo SQLAlchemy

```
# src/web/__init__.py
....
from src.core.db import db_session

def create_app(env="development", static_folder="static"):
    ....
    @app.teardown_appcontext
    def shutdown_session(exception=None):
        db_session.remove()

    return app

....
```

ORM - Flask-SQLAlchemy

- Agregamos Flask-SQLAlchemy

```
poetry add Flask-SQLAlchemy
```



- Breve diferencia entre SQLAlchemy y Flask-SQLAlchemy

Ejemplo Flask-SQLAlchemy

- Antes de continuar, creamos nuestro .env

```
# .env

SQLALCHEMY_DATABASE_URI=sqlite:///site.db

o

SQLALCHEMY_DATABASE_URI = postgresql://misuario:mipass@localhost:5432/
```

Ejemplo Flask-SQLAlchemy

```python
# src/core/db.py

from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass


db = SQLAlchemy(model_class=Base)


def init_db(app):
    print("init_db")
    db.init_app(app)
    config_db(app)


def config_db(app):
    @app.teardown_request
    def close_db_session(exception=None):
        db.session.remove()


def reset_db(app):
    with app.app_context():
        db.drop_all()
        db.create_all()
```

Ejemplo Flask-SQLAlchemy

```python
# src/core/board/issue.py

from datetime import datetime, timezone

from src.core.db import db


class Issue(db.Model):
    # __tablename__ = "issues"
    id = db.Column(db.Integer, primary_key=True)
    user = db.Column(db.String(255))
    title = db.Column(db.String(255))
    description = db.Column(db.Text)
    status = db.Column(db.String(255))
    issue_type_id = db.Column(db.Integer, db.ForeignKey("issue_type.id"))
    created_at = db.Column(db.DateTime, default=datetime.now(timezone.utc))
    updated_at = db.Column(
        db.DateTime,
        default=datetime.now(timezone.utc),
        onupdate=datetime.now(timezone.utc),
    )

    def __init__(self, user, title, description, status):
        self.user = user
        self.title = title
        self.description = description
        self.status = status

    def __repr__(self):
        return f"<Issue {self.id}>"


class IssueType(db.Model):

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(255))
    issues = db.relationship("Issue", backref="issue_type", lazy=True)

    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return f"<IssueType {self.id}>"
```

Ejemplo Flask-SQLAlchemy

```
# src/web/__init__.py
....
from src.core.db import db
from dotenv import load_dotenv
import os


def create_app(env="development", static_folder="static"):
    ....

    app.config["SQLALCHEMY_DATABASE_URI"] = os.getenv("SQLALCHEMY_DATABASE_URI")

    init_app(app)

    ....
    @app.teardown_appcontext
    def shutdown_session(exception=None):
        db.session.remove()

    @app.cli.command('init-db')
    @click.option('--drop-tables', is_flag=True, help='Elimino las tablas antes de crearlas.')
    def init_db_command(drop_tables):
        """Inicializo la BBDD"""
        if drop_tables:
            db.drop_all()
            click.echo('Tablas eliminadas')
        db.create_all()
        click.echo('BBDD creada')


    return app

....
```

Ejemplo Flask-SQLAlchemy

```python
# src/web/controllers/issue.py

from flask import Blueprint, render_template, request
from src.core.issue import Issue
from src.core.db import db

issue_blueprint = Blueprint("issues", __name__, url_prefix="/issues")


@issue_blueprint.route("/")
def issues_index():
    issues = Issue.query.all()
    return render_template("issues/index.html", issues=issues)


@issue_blueprint.route("/issues/add", methods=["POST"])
def issues_add():
    issue = {
        # "id": request.form.get("id"),
        "user": request.form.get("user"),
        "title": request.form.get("title"),
        "description": request.form.get("description"),
        "status": request.form.get("status"),
    }
    issue = Issue(**issue)
    db.session.add(issue)
    db.session.commit()
    issues = Issue.query.all()

    return render_template("issues/index.html", issues=issues)
```

Lo que viene....

- Integrar sesiones, ORM, login

Fin