

PROYECTO DE SOFTWARE

Cursada 2022

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 extensiones 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)
```

- Modifiquemos la app y veamos de nuevo...

[http://localhost:5000/iniciar sesion](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`

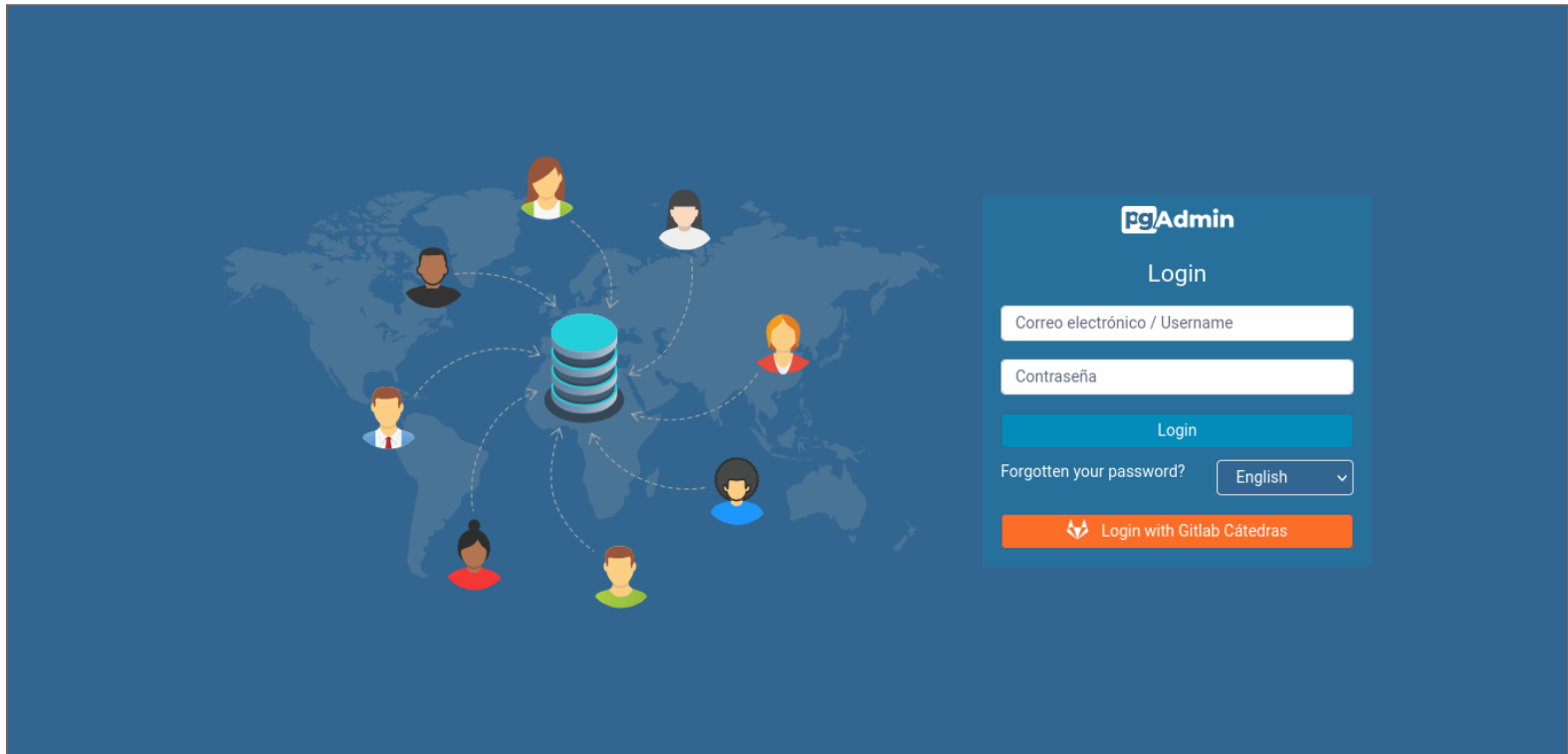
IMPORTANTE

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



- Sitio oficial: <https://www.pgadmin.org/>

- pgAdmin de la cátedra:
<https://sql.proyecto20212.linti.unlp.edu.ar/>

ACCESO A BBDD – POSTGRESQL

- Vamos a acceder a través de psycopg2.

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

```
poetry add psycopg2
```

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+psycopg2://{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

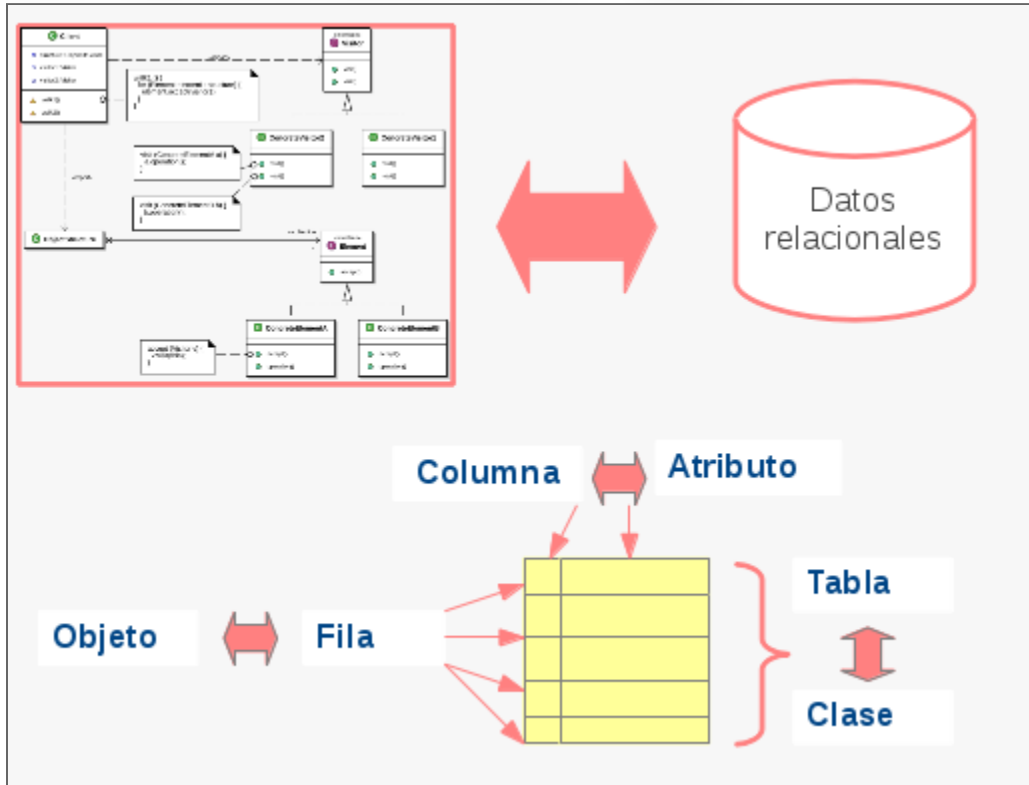
def get_conn():
    conn = psycopg2.connect(
        host=current_app.config.get("DB_SERVER"),
        database=current_app.config.get("DB_DATABASE"),
        user=current_app.config.get("DB_USER"),
        password=current_app.config.get("DB_PASSWORD"))

    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 - SQLALCHEMY

- Agregamos SQLAlchemy

```
poetry add SQLAlchemy
```

```
diego@diego-Lenovo-V330-15IKB:~/Documentos/Git/Proyecto2020/2022/miercoles/admin$ poetry add SQLAlchemy
Using version ^1.4.41 for SQLAlchemy

Updating dependencies
Resolving dependencies... (4.1s)

Writing lock file

Package operations: 2 installs, 0 updates, 0 removals

  • Installing greenlet (1.1.3)
  • Installing sqlalchemy (1.4.41)
```

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+psycopg2://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

```
# 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
```

```
diego@diego-Lenovo-V330-15IKB:~/Documentos/Git/Proyecto2020/2022/miercoles/admin$ poetry add Flask-SQLAlchemy
Using version ^2.5.1 for Flask-SQLAlchemy

Updating dependencies
Resolving dependencies... (0.2s)

Writing lock file

Package operations: 1 install, 0 updates, 0 removals

  • Installing flask-sqlalchemy (2.5.1)
diego@diego-Lenovo-V330-15IKB:~/Documentos/Git/Proyecto2020/2022/miercoles/admin$
```

- Breve diferencia entre SQLAlchemy y Flask-SQLAlchemy.

EJEMPLO FLASK-SQLACHEMY

- Antes de continuar, modificamos 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"
    SQLALCHEMY_DATABASE_URI =
f"postgresql+psycopg2://{DB_USER}:
{DB_PASSWORD}@{DB_SERVER}:{DB_PORT}/{DB_DATABASE}"

...
```

EJEMPLO FLASK-SQLALCHEMY

```
# src/core/db.py

from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

def init_db(app):
    db.init_app(app)
    db.create_all()
```

EJEMPLO FLASK-SQLACHEMY

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

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

class Issue(db.Model):
    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 FLASK-SQLACHEMY

```
# 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

...
```

EJEMPLO FLASK-SQLACHEMY

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

from flask import render_template, request, Blueprint
from src.core.board.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("/add", methods=["POST"])
def issues_add():
    data_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(**data_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