INFORME: “Trabajo Practico Base de Datos”.

Materia: Base de datos (7515 / 9505 / TA 044).

Profesor: Hernan, Merlino.

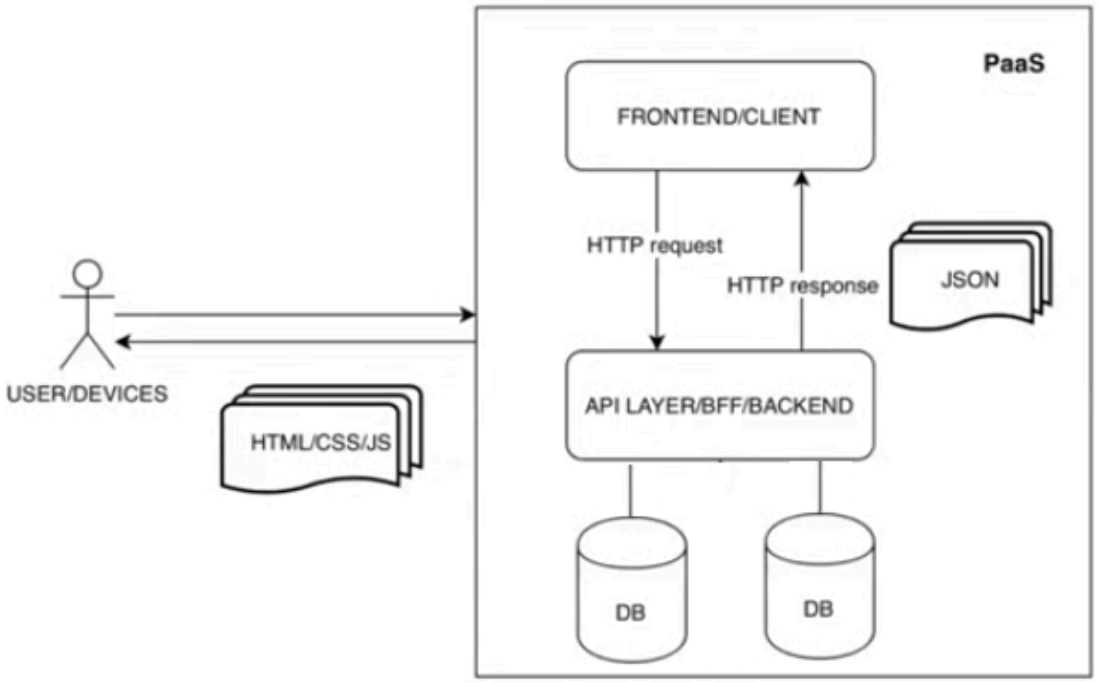
Integrantes:

- Iván, Maximoff - Padrón: 110868
- Matias, Bartellone - Padrón: 110484
- Martín, Castro - Padrón: 109807

Elección de las tecnologías.

El lenguaje de programación que elegimos fue **Python** ya que posee una gran comunidad y amplitud de librerías. Además de ser un lenguaje con el que ya teníamos experiencia previamente.
Como framework utilizamos **Flask**, el cuál tiene como ventajas principales ser liviano y flexible por lo que coincidia con las necesidades que teníamos para realizar este trabajo. Agregado a esto, Flask, tiene gran soporte y documentación para trabajar con base de datos.
Centrándonos en las bases de datos, la base de datos relacional que utilizamos fue **MySQL**. Elegimos esta opción ya que es una de las más tradicionales y nos pareció buena idea familiarizarnos con esta tecnología. Como ventajas tiene que al ser una de las más conocidas cuenta con una gran documentación.
Por otro lado, la base de datos no relacional que elegimos fue **MongoDB**. Como mencionamos anteriormente, teníamos la idea de trabajar con las bases de datos más tradicionales y MongoDB, dentro de las bases de datos noSQL, es de las más renombradas.
Por último, para el frontend, usamos **HTML**, **CSS** y **JavaScript** ya que teníamos experiencia previa en el desarrollo de interfaces web con estas tecnologías por lo que nos fue más fácil conectar las distintas partes del proyecto.

Diagrama de arquitectura.



Configuración y conexión a base de datos.

A la hora de conectar nuestro backend con las bases de datos seguimos los pasos de las librerías de Python las cuales hicieron mucho más sencilla la conexión ya que proveen funciones de alto nivel que facilitan la conexión.
Para probar que todo funcionaba correctamente hicimos pruebas manuales con Postman.
Una vez que todo funcionaba utilizamos una API para conectar el backend con el frontend.

Para configurar cada base de datos seguimos la documentación que estaba disponible en la web. Por ejemplo, para MySQL la conexión se hacía de la siguiente forma.

```
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'tu_usuario'
app.config['MYSQL_PASSWORD'] = 'tu_contraseña'
app.config['MYSQL_DATABASE'] = 'nombre_base_de_datos'

def get_db_connection():
    conn = mysql.connector.connect(
        host=app.config['MYSQL_HOST'],
        user=app.config['MYSQL_USER'],
        password=app.config['MYSQL_PASSWORD'],
        database=app.config['MYSQL_DATABASE']
    )
    return conn
```

Para el caso de MongoDB:

```
app.config['MONGO_URI'] = 'mongodb://localhost:27017/nombre_base_de_datos'

def get_db_connection():
    client = MongoClient(app.config['MONGO_URI'])
    db = client.get_database()
    return db
```

Para trabajar con la base relacional MySQL utilizamos la librería de **SQLAlchemy** la cual proporciona un conjunto de herramientas y un ORM (Object Relational Mapping) para trabajar con distintas bases de datos relacionales como MySQL, PostgreSQL, entre otras. SQLAlchemy permite mapear tablas de bases de datos a objetos en Python, lo que simplifica las operaciones CRUD complejas sin tener que escribir SQL manualmente.
Por otro lado, utilizamos **Mongoose** para la base de datos no relacional. Mongoose es una librería de JavaScript que nos otorga una interfaz sencilla para interactuar con MongoDB y facilita el manejo de los datos.

Esta librería actúa como un ODM (Object Data Modeling) para MongoDB, lo que facilita las interacciones entre la base de datos y el código JavaScript.

Descripción de funcionalidades CRUD.

Las funciones CRUD (Create, Read, Update, Delete) son las que nos permiten interactuar con nuestras bases de datos. En particular, en nuestra implementación nos fue bastante sencillo su uso ya que las librerías cuentan con funciones que resuelven gran parte de la lógica.
Para nuestra implementación seguimos estos ejemplos y los adecuamos a nuestras necesidades.
En el caso de SQLAlchemy:

Create

```python
def crear_usuario(nombre, correo, edad):
    nuevo_usuario = Usuario(nombre=nombre, correo=correo, edad=edad)
    try:
        session.add(nuevo_usuario)
        session.commit()
    except Exception as e:
        session.rollback()
        print("Error al crear el usuario:", e)
```

Read

```python
def obtener_usuarios():
    try:
        usuarios = session.query(Usuario).all()
    except Exception as e:
        print("Error al obtener los usuarios:", e)
```

Update

```python
def actualizar_usuario(correo, nueva_edad):
    try:
        usuario = session.query(Usuario).filter_by(correo=correo).first()
        if usuario:
            usuario.edad = nueva_edad
            session.commit()
        else:
            print("Usuario no encontrado")
    except Exception as e:
        session.rollback()
        print("Error al actualizar el usuario:", e)
```

Delete

```python
def eliminar_usuario(correo):
    try:
        usuario = session.query(Usuario).filter_by(correo=correo).first()
        if usuario:
            session.delete(usuario)
            session.commit()
        else:
            print("Usuario no encontrado")
    except Exception as e:
        session.rollback()
        print("Error al eliminar el usuario:", e)
```

En el caso de Mongoose:

Create

```javascript
async function crearUsuario() {
    const nuevoUsuario = new Usuario({
      nombre: "nombre",
      apellido: "apellido",
      edad: 30
    });

    try {
      const usuarioGuardado = await nuevoUsuario.save();
      console.log("Usuario creado:", usuarioGuardado);
    } catch (error) {
      console.error("Error al crear el usuario:", error);
    }
  }
```

Read

```javascript
async function obtenerUsuarios() {
  try {
    const usuarios = await Usuario.find();
    console.log("Usuarios encontrados:", usuarios);
  } catch (error) {
    console.error("Error al obtener usuarios:", error);
  }
}
```

Update

```
async function actualizarUsuario() {
  try {
    const usuarioActualizado = await Usuario.findOneAndUpdate(
      { apellido: "apellido" },
      { edad: 31 },
      { new: true }
    );
    console.log("Usuario actualizado:", usuarioActualizado);
  } catch (error) {
    console.error("Error al actualizar el usuario:", error);
  }
}
```

Delete

```
async function eliminarUsuario() {
  try {
    const usuarioEliminado = await Usuario.findOneAndDelete({ apellido: "apellido" });
    if (usuarioEliminado) {
      console.log("Usuario eliminado:", usuarioEliminado);
    } else {
      console.log("No se encontró el usuario.");
    }
  } catch (error) {
    console.error("Error al eliminar el usuario:", error);
  }
}
```

Las diferencias claves entre el manejo de los datos entre un modelo relacional y un modelo no relacional son que en un modelo relacional tenemos tablas con filas y columnas, un esquema fijo, ACID, consultas mediante SQL, relaciones entre tablas mediante PK Y FK, entre otras.
Por otro lado, en una base de datos no relacional, tenemos un modelo de datos que depende de la orientación de la base de datos NoSQL (en nuestro caso orientado a documentos), esquema flexible, consistencia eventual (no instantánea), las consultas dependen de cada base de datos (no son generales como los modelos SQL), entre otras.

Comparación entre bases de datos relacionales y NoSQL.

Las principales ventajas que encontramos en el modelo relacional fueron las siguientes:

- Estructura organizada. Nos permite organizar los datos de manera eficiente.
- Datos auto-incrementales. Esta característica que podemos añadir a ciertos datos es la que nos evita cometer errores y tener que enviar parámetros manualmente. Un ejemplo de esto, suelen ser los ids.
- PK y FK. Nos ayuda a prevenir errores como introducir varias veces un dato que debería ser único.
- Consultas complejas mediante SQL. Si bien no profundizamos por el alcance del trabajo, es una de sus grandes ventajas
- Transacciones ACID. Es una característica de las bases de datos relacionales que si bien no la usamos directamente, nos ayudó saber de su existencia a la hora de realizar el trabajo. Principalmente por la atomicidad y consistencia.

Por otro lado, sus desventajas son:

- Rigidez del esquema. Por ejemplo, sí tenemos un campo “ID” el cual es de tipo *int* pero queremos, por cierta regla de negocio, que si no tiene “ID” diga en ese campo “Sin ID” no lo podemos hacer ya que la columna solo acepta un tipo de dato.
- Haciendo pruebas, nos dimos cuenta que no siempre son la mejor opción, ya que para datos desestructurados, por ejemplo, conviene otro enfoque como una base de datos NoSQL orientada a documentos.

Las ventajas que encontramos al trabajar con un modelo NoSQL fueron:

- Flexibilidad. Nos permite guardar datos no estructurados o semiestructurados. Es útil cuando los datos cambian frecuentemente como las preferencias en un sitio web.
- Rendimiento. Al hacer pruebas notamos que este tipo de base de datos funcionaba con una mayor performance. En un principio no estábamos seguros de esto, ya que podría deberse a como estábamos ejecutando el proyecto pero luego de investigar descubrimos que las bases de datos no relacionales suelen funcionar con mayor eficiencia que las relacionales.
- Distintos objetivos de las bases NoSQL. Si bien este punto no fue específicamente del desarrollo del trabajo. Durante la investigación de qué base de datos NoSQL usar a partir de lo visto en las clases, vimos que hay distintos tipos de bases de datos no relacionales que cubren distintos objetivos como las bases de datos NoSQL orientadas a grafos, clave-valor, entre otras.

Las desventajas que vimos fueron las siguientes:

- Falta de consistencia. En algunas de nuestras pruebas nos ocurrió perder consistencia.

- Consultas limitadas. Al momento de realizar consultas a la base de datos, estas no ofrecen la misma versatilidad que las bases de datos relacionales.

En conclusión, utilizaremos una base de datos SQL cuando los datos sean altamente estructurados y tengan relaciones claras, necesite de ACID (ej: sistemas financieros), la aplicación requiera consultas complejas, entre otras.
Por otro lado, se suele utilizar un modelo NoSQL en circunstancias donde el volumen de los datos es muy grande (ya que tiene un alto rendimiento), los datos son desestructurados o semi-desestructurados, en situaciones donde necesito flexibilidad, etc.

Dificultades y aprendizajes.

Una de las principales dificultades que tuvimos a la hora de hacer este proyecto fue empezar a trabajar con las bases de datos, ya que teníamos una noción teórica de cómo funcionaba pero nunca habíamos trabajado con una. La forma de solucionar este problema fue leer la documentación y realizar pruebas pequeñas para ir progresando paso a paso hasta lograr un backend funcional.
Por último, nos llevamos como aprendizaje el haber trabajado con bases de datos que son muy solicitadas en el ámbito profesional, como pueden ser MongoDB o MySQL.
Además de practicar el desarrollo de una aplicación web que cuente con distintas bases de datos, realizar sus conexiones, crear la interfaz, entre otras cosas que aprendimos.