STACK:

React (TypeScript), Django, Capas y PostgreSQL

Grupo 2

CONTENIDO

Definición, historia y evolución

Relación entre temas asignados

Situaciones donde se aplican

Ventajas y desventajas

Principios SOLID asociados

Atributos de Calidad asociados

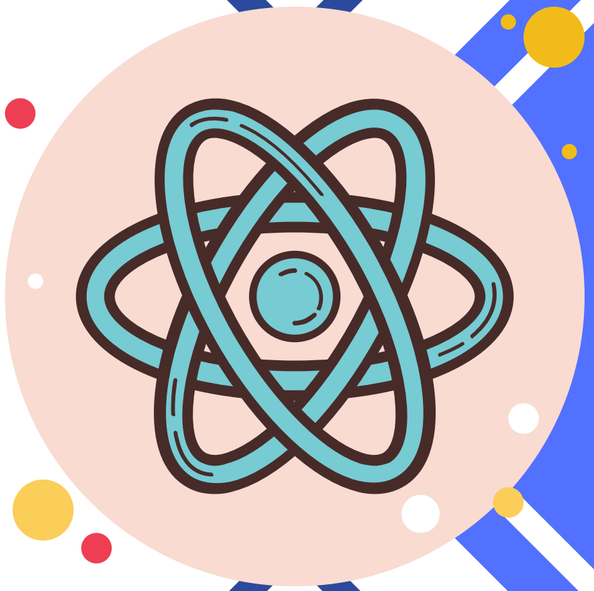
Casos de Estudio Relevantes

Ejemplo práctico y funcional

Definición, historia y evolución

REACT.JS

- Librería de JavaScript open-source usada para crear interfaces de usuario.
- Creado por Jordan Walke (ing. de software en Meta).
- En 2011 primer prototipo: "FaxJS".
- Inspiración en XHP (componente HTML de una librería para PHP).
- Se mostró como Open-Sourced en JSConf US en 2013.
- Segunda tecnología web más usada en 2022 (después de Node.js).
- Última versión: 18.2.0 (junio, 2022)



Definición, historia y evolución

TYPESCRIPT

- Lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft.
- Superconjunto de JavaScript: añade tipos estáticos y objetos basados en clases.
- Creado para aplicaciones de larga escala.
- Útil para refactorizar código debido al compilador.
- Ofrece code hinting y code completion.
- Publicado en octubre 2012.
- 5to lenguaje más usado en 2022.

Definición, historia y evolución

DJANGO

- Framework web basado en Python, gratuito y de código abierto, que sigue el patrón arquitectónico modelo-plantilla-vista (MTV).
- Creado 2003 por Adrian Holovaty y Simon Willison, programadores web del periódico Lawrence Journal-World.
- Es mantenido por Django Software Foundation (DSF).
- En 2022 el 14.65% de los encuestados lo usan.
- Última versión: 4.1.8 (abril, 2023)

Definición, historia y evolución

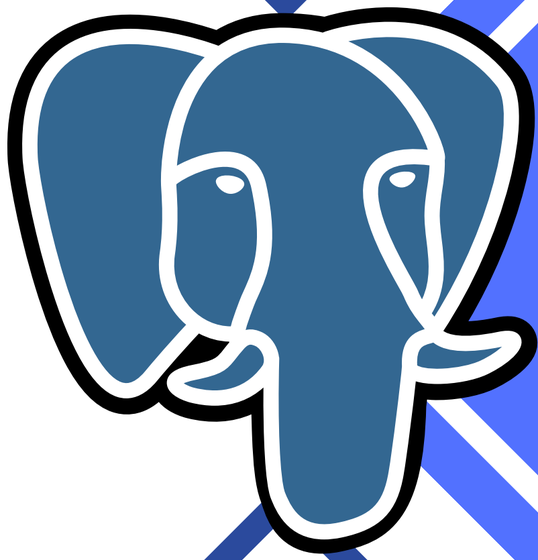
PYTHON

- Lenguaje de programación de código abierto y multiplataforma que se destaca por su código legible y limpio.
- Publicado en los años 90.
- Conocido por su versatilidad y facilidad de uso, lo que lo hace ideal para principiantes y expertos por igual.
- 4to lenguaje más usado en 2022.
- Última versión: 3.11.2 (febrero, 2023)

Definición, historia y evolución

POSTGRESQL

- Sistema de gestión de bases de datos objeto-relacional (ORDBMS) open source.
- Desarrollado por PostgreSQL Global Development Group.
- Inició en 1982 como un proyecto en la Universidad de Berkeley.
- Segunda base de datos más utilizada en 2022 (después de MySQL).



Definición, historia y evolución

SQL

- (Structured Query Language) Lenguaje de dominio específico diseñado para administrar y recuperar información de sistemas de gestión de bases de datos relacionales.
- Maneja álgebra y cálculo relacional para efectuar consultas.
- Se remonta a la década de 1970, cuando en los laboratorios de IBM se creó el nuevo software de base de datos System R.
- 3er lenguaje más usado en 2022.

Definición, historia y evolución

CAPAS

- Patrón arquitectónico común y ampliamente utilizado en el desarrollo de software.
- También conocido como arquitectura de n niveles.
- Se compone de varias capas horizontales separadas que funcionan juntas como una única unidad de software.
- Es flexible a diferentes necesidades, razón por la que no ha pasado por mucho cambio.
- Actualmente capas dedicadas a la seguridad y la integración se han hecho más comunes.
- Otros patrones como microservicios y serverless tienen como inspiración capas.

Definición, historia y evolución

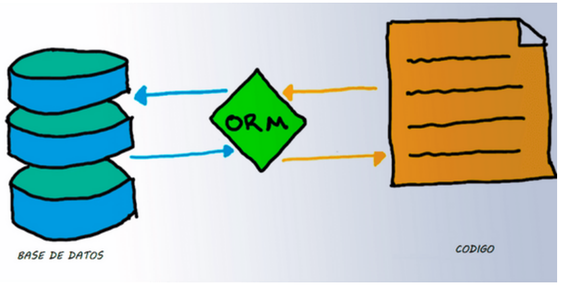
AXIOS

- Biblioteca de JavaScript basada en el cliente HTTP para el navegador y Node.js.
- Permite a los desarrolladores realizar fácilmente solicitudes HTTP asincrónicas y gestionar las respuestas
- Conocido por su API sencilla, manejo de errores y capacidad de interceptar solicitudes y respuestas.
- Creado en 2015, gana popularidad en 2016.

Definición, historia y evolución

ORM

- Técnica de programación para mapear objetos de una aplicación a tablas en una bd relacional.
- Elimina la necesidad de escribir en SQL.
- Se remonta a los años 80.
- Se hizo popular con el auge de la programación orientada a objetos.
- Sistemas ORM populares: Hibernate, Entity Framework y SQLAlchemy.
- Utilizamos Psycopg2 (adaptador de bases de datos).




```
String query = "INSERT INTO clientes (id,nombre,email,pais) VALUES (@id, @nombre, @email

command.Parameters.AddWithValue("@id","1")
command.Parameters.AddWithValue("@nombre","nombre")
command.Parameters.AddWithValue("@email","email")
command.Parameters.AddWithValue("@pais","pais")

command.ExecuteNonQuery();
```

```
var cliente = new Cliente();
cliente.Id = "1";
cliente.Nombre = "nombre";
cliente.Email = "email";
cliente.Pais = "pais";
session.Save(customer);
```

Relación entre temas

En una aplicación web:

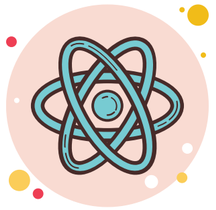
- React → construir interfaz de usuario.
- TypeScript → escribir el código del lado del cliente.
- Django → manejar la lógica del lado del servidor.
- Python → escribir el código del lado del negocio.
- PostgreSQL → base de datos para almacenar y recuperar datos.
- SQL →recuperar y administrar los datos.

Capa de Presentación

Capa de Negocio

Capa de Acceso a Datos

Situaciones



- Crear aplicaciones web de una sola página (SPA).
- Desarrollar aplicaciones móviles (React Native).
- Crear componentes reutilizables para interfaces de usuario.

- Desarrollar apps web y móviles.
- Crear bibliotecas y frameworks.

- Ciencia de datos.
- Desarrollo web (django, flask, pyramid).

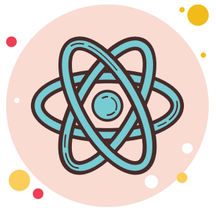
Crear aplicaciones web que necesiten:

- funcionalidades precontruidas (autenticación de usuario, formularios).
- alta escalabilidad y rendimiento

- Almacenar y analizar grandes cantidades de datos.
- Apps que necesiten alto nivel de seguridad y precisión en los datos.
- Apps que necesiten bd robusta y escalable

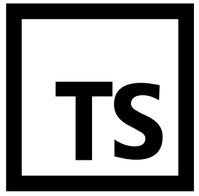
- Crear sistemas que requieran una alta escalabilidad y rendimiento.
- Apps sencillas de entender y de mantener por otros desarrolladores.

Ventajas

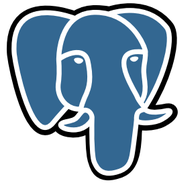


- Reusabilidad.
- Desempeño.
- Flexibilidad.
- Gran comunidad.

- Desarrollo rápido.
- Escalabilidad.
- Seguridad.
- Gran comunidad.



- Type checking.
- Improved tootling.
- Compatibilidad con JavaScript.

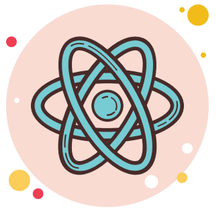


- Funciones avanzadas
- Fiabilidad.
- Escalabilidad.
- Gran comunidad.

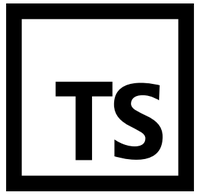
- Mantenibilidad.
- Escalabilidad.
- Facilidad para testing.
- Flexibilidad.

- Variedad bibliotecas, frameworks.
- Portabilidad.
- Eficiente.

Desventajas



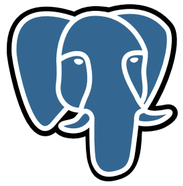
- Curva de aprendizaje.
- Complejidad.
- Nuevas versiones.
- Añade al tamaño de la aplicación.



- Curva de aprendizaje.
- Tiempo de compilación.
- Verbosidad.
- Actualización de bibliotecas y marcos.

- Comunicación entre diferentes capas.
- Dificultad de escalar verticalmente el software.

- Flexibilidad limitada.
- No recomendado en aplicaciones pequeñas.



- Configuración.
- Requisitos de Hardware.

- No útil en aplicaciones móviles.
- Cambios recientes de versión.

Principios SOLID

Single Responsability Principle (SRP)

- Capas: promueve separación de responsabilidades.
- React: componentes.
- Django: separación de responsabilidades (MVC/MTV).

Open/Closed Principle (OCP)

- Capas: se puede agregar funciones a una capa sin afectar las demás.
- React: extender componentes.
- Django: herencia de clases y composición de componentes.

Liskov Substitution Principle (LSP)

- React: composición y props para crear componentes genéricos.
- Django: herencia en sus clases base.

Principios SOLID

Interface Segregation Principle (ISP)

- Django: dividiendo la funcionalidad en vistas y clases más pequeñas.

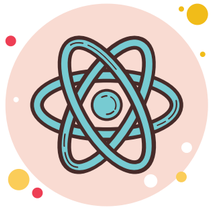
Dependency Inversion Principle (DIP)

- Capas: las capas superiores dependen de abstacciones.
- Django: permite inyección de dependencias.

Atributos de Calidad

Capas

- Mantenibilidad
- Modularidad
- Extensibilidad
- Testabilidad
- Interoperabilidad
- Seguridad

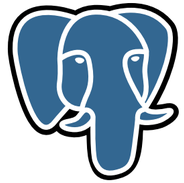


React

- Mantenibilidad
- Rendimiento
- Escalabilidad
- Flexibilidad
- Portabilidad
- Interoperabilidad.

Django

- Mantenibilidad
- Seguridad
- Escalabilidad
- Portabilidad
- Extensibilidad



PostgreSQL

- Fiabilidad
- Escalabilidad
- Rendimiento
- Seguridad
- Flexibilidad

CASOS DE USO RELEVANTES

APLICACIONES EMPRESARIALES

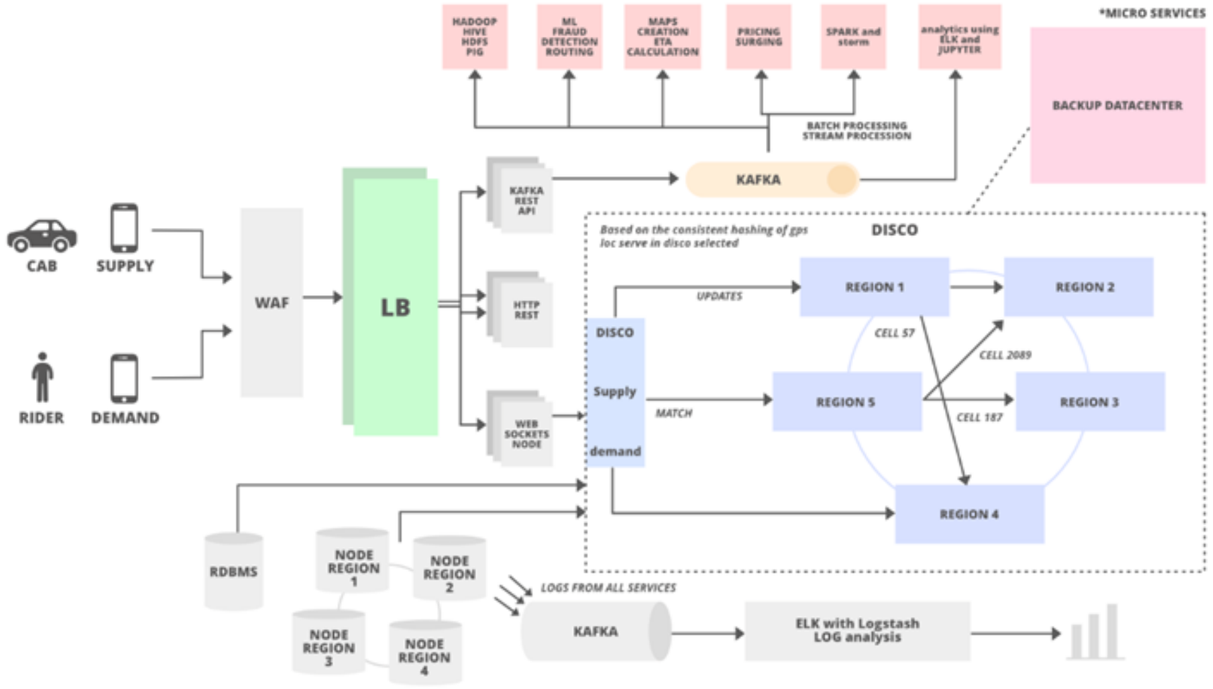
La arquitectura en capas es una buena opción para aplicaciones empresariales que necesitan una estructura organizada y escalable. Al dividir la aplicación en capas, se puede lograr una separación clara y definida entre la lógica de negocio y la interfaz de usuario. Además, esto permite que diferentes equipos se concentren en diferentes capas sin afectar la funcionalidad del resto de la aplicación. Esto puede ser especialmente útil en grandes empresas con equipos de desarrollo distribuidos en diferentes lugares geográficos.

APLICACIONES WEB

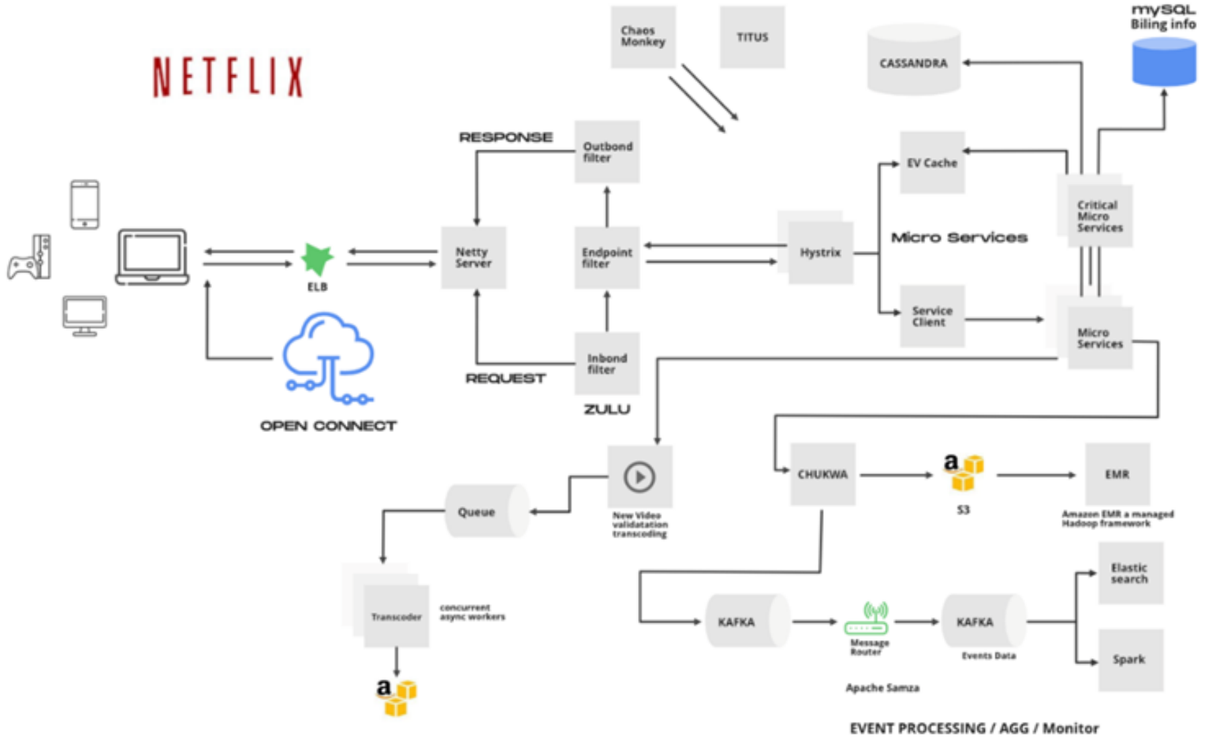
La arquitectura en capas es una buena opción para aplicaciones web. Al separar la lógica del servidor y la interfaz de usuario en capas diferentes, se puede lograr una mayor seguridad y escalabilidad. La capa de presentación se puede diseñar para manejar la interacción con el usuario, mientras que la capa de negocios maneja la lógica de negocio y la capa de datos maneja la persistencia de datos. Además, facilita la integración de nuevas funcionalidades y mejoras sin afectar la funcionalidad existente.

CASOS DE ESTUDIO

UBER



NETFLIX



CASO DE USO PROPUESTO

nclsbayona/
presentacion1-arqui

Implementación del patrón arquitectural en capas. Frontend en React, Backend en DJango y BBDD PostgreSQL

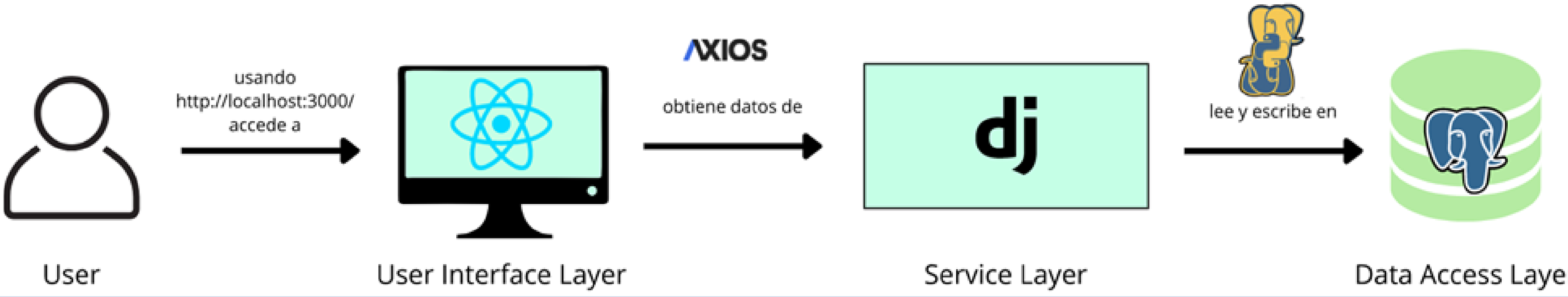
1 Contributor 0 Issues 0 Stars 0 Forks

nclsbayona/presentacion1-arqui: Implementación del patrón arquitectural en capas. Frontend en React, Backend en DJango y...

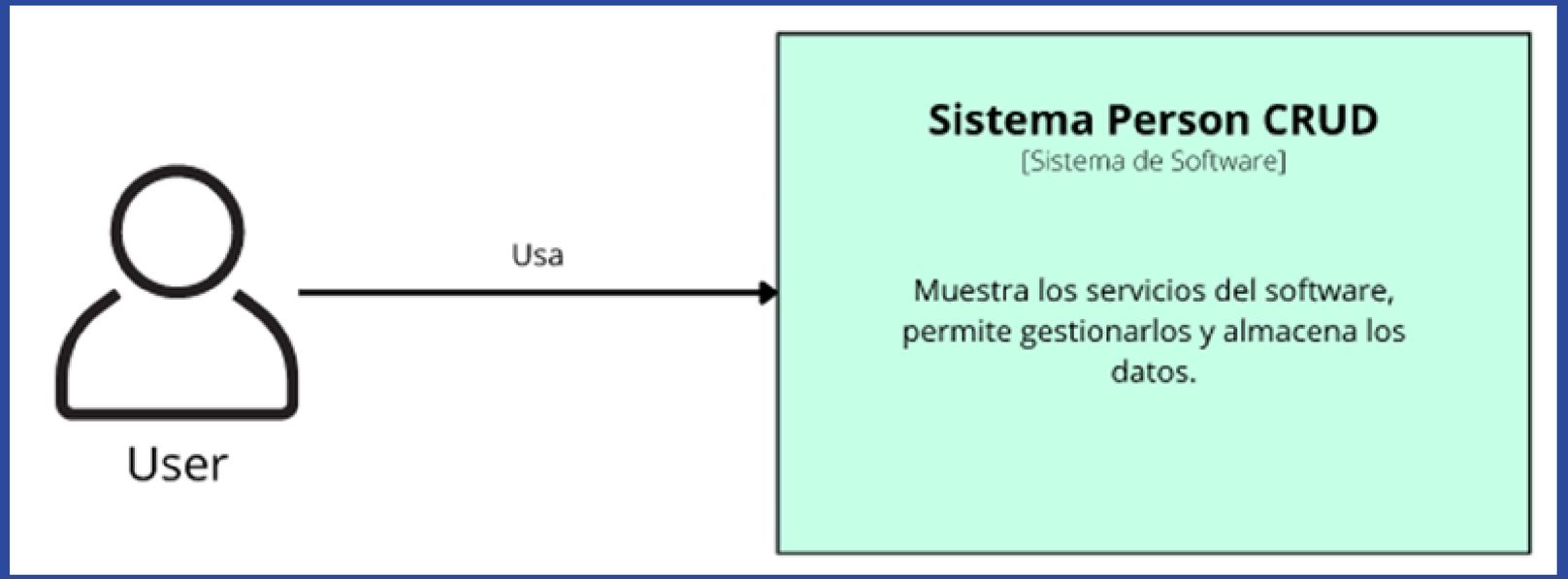
Implementación del patrón arquitectural en capas. Frontend en React, Backend en DJango y BBDD PostgreSQL - GitHub - nclsbayona/presentacion1-arqui: Implementación del patrón arquitectural en capas....

GitHub

ARQUITECTURA DE ALTO NIVEL

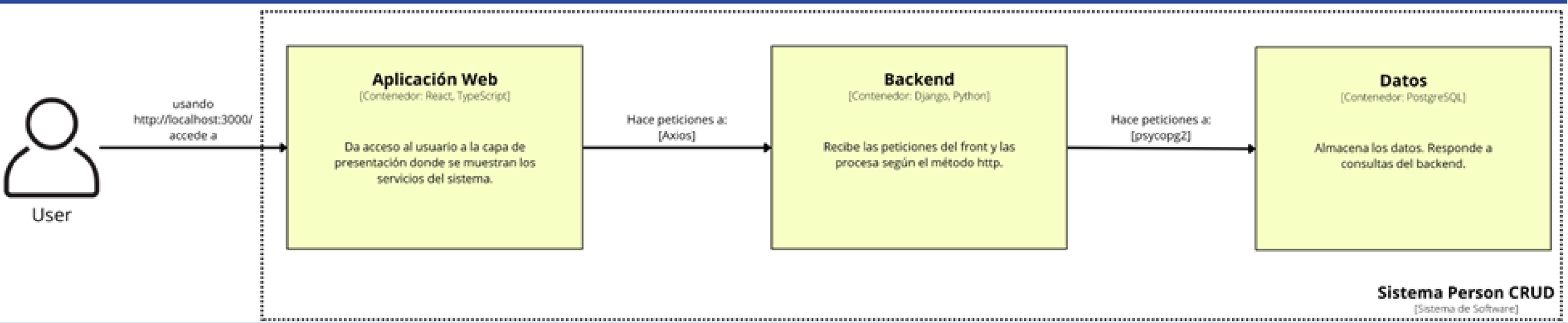


ARQUITECTURA DE BAJO NIVEL



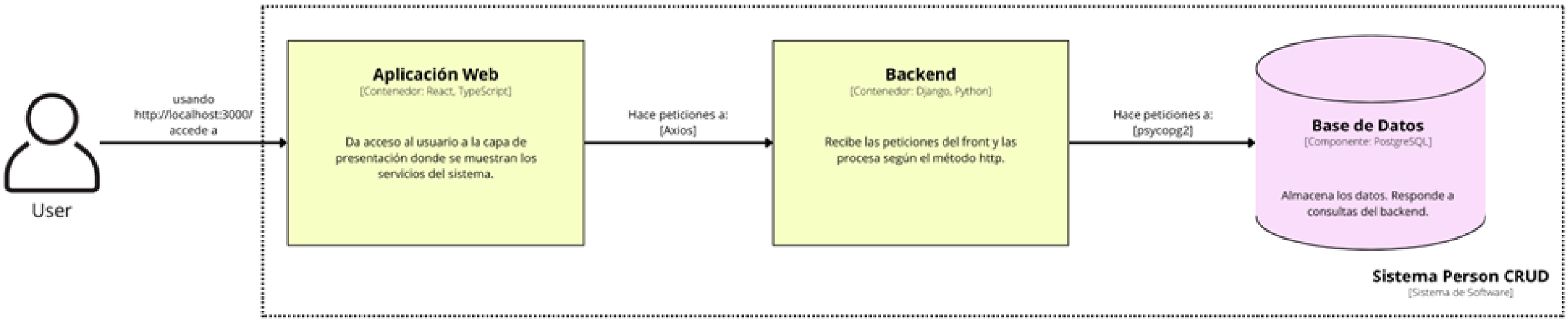
CONTEXTO

ARQUITECTURA DE BAJO NIVEL



CONTENEDORES

ARQUITECTURA DE BAJO NIVEL



COMPONENTES

DIAGRAMA DE DESPLIEGUE