PRESENTACIÓN 1 – ARQUITECTURA DE SOFTWARE

Grupo 2

Nicolas Esteban Bayona Ordoñez

Diana Sofia Carrillo Gómez

Diana Natalia Chaparro Sanabria

FACULTAD DE INGENIERÍA
DEPARTAMENTO DE SISTEMAS
BOGOTÁ D.C. 2023

TABLA DE ILUSTRACIONES

TABLA DE CONTENIDO

1. Introducción

En este documento se presenta una explicación detallada de varias tecnologías. Su propósito es educar al(la) lector(a) sobre diferentes ítems relevantes a cada una como: su definición, sus ventajas, para qué usarlas, entre otros. También se expone un ejemplo de una aplicación web sencilla usando todas aquellas.

2. Arquitectura en Capas

2.1. Resumen

El estilo de arquitectura en capas busca estructurar una aplicación en varias capas, cada una de las cuales es responsable de un aspecto específico de la funcionalidad de una aplicación. Este modelo se usa a menudo en el desarrollo de aplicaciones a gran escala y es conocido por su modularidad, escalabilidad y mantenibilidad. El modelo de arquitectura en capas no es ajeno a las estructuras organizativas y de comunicación de TI tradicionales que se encuentran en la mayoría de las empresas, por lo que es una arquitectura usada en la mayoría de las aplicaciones empresariales, para entender esto podemos referirnos a Grady Booch quien afirma “Show me the organization of your team and I will show you the architecture of your system” [1].

2.2. Definición, historia y evolución

2.2.1. Definición.

La arquitectura en capas es un estilo arquitectural que organiza una aplicación en diferentes capas, cada una con su propio conjunto de responsabilidades y funciones de manera que se puedan separar responsabilidades y preocupaciones en una aplicación. Según Erich Gamma, "las capas proporcionan una forma efectiva de controlar la complejidad al establecer límites claros en las responsabilidades de cada componente". Esto significa que cada capa se encarga de una tarea específica, lo que reduce la complejidad de la aplicación en general.

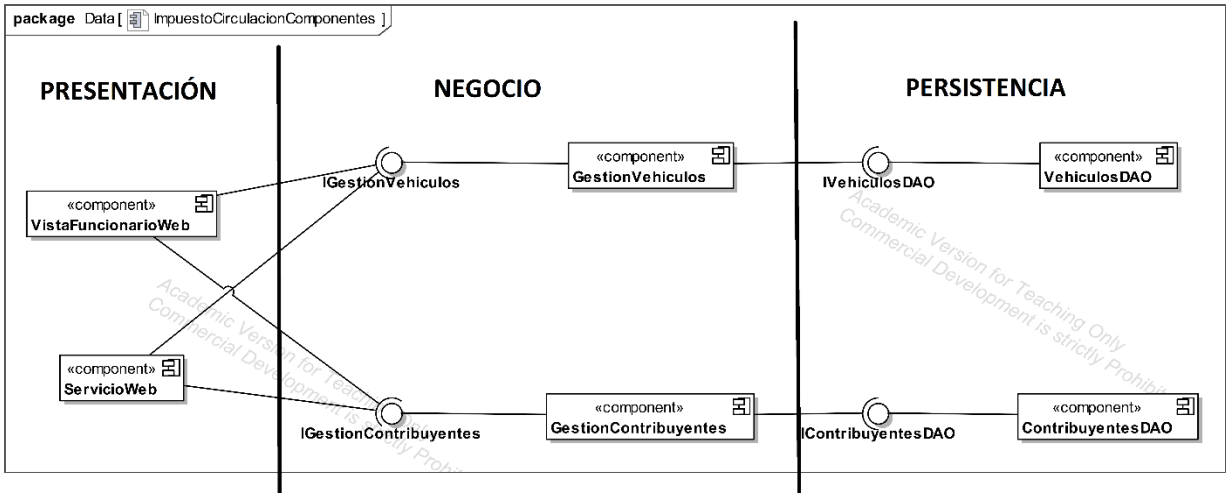

Ilustración 1. Ejemplo sencillo de arquitectura en capas (Tres capas)

“In a strictly layered structure, a layer can only use the services of the layer immediately below it. Many variations of this pattern, lessening the structural restriction, occur in practice.” [2]

2.2.2. Historia.

La historia del estilo por capas se remonta a la década de 1970, cuando se comenzó a desarrollar el concepto de arquitecturas de software en capas para sistemas de bases de datos. Este enfoque se popularizó en los años 80, con el advenimiento de las aplicaciones cliente-servidor.

En la década de 1990, el estilo por capas se consolidó como un estilo arquitectónico ampliamente utilizado en el diseño de software empresarial. En particular, el modelo de tres capas (presentación, lógica de negocio y acceso a datos) se convirtió en un estándar de facto para el desarrollo de aplicaciones web.

En los últimos años, el estilo por capas ha evolucionado hacia enfoques más sofisticados, como el estilo de arquitectura hexagonal (también conocido como puertos y adaptadores), que busca separar aún más las capas del sistema y reducir la dependencia de tecnologías específicas.

2.2.3. Evolución.

Con el tiempo, el estilo arquitectónico de capas ha evolucionado para satisfacer las cambiantes necesidades del desarrollo de software. Un cambio significativo ha sido la introducción de capas adicionales orientadas a manejar aspectos como la seguridad, la integración con otros sistemas y otras necesidades empresariales.

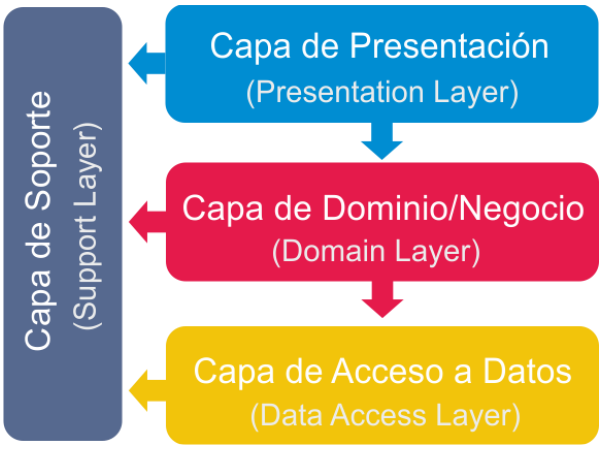

Ilustración 2. Ejemplo de arquitectura en capas usando una capa de soporte

Otro cambio importante que sufrió la arquitectura en capas se relaciona directamente con la creciente adopción de servicios web que generan la necesidad de una arquitectura más flexible, lo cual llevó a la aparición de la arquitectura SOA que se fundamenta en la separación de responsabilidades.

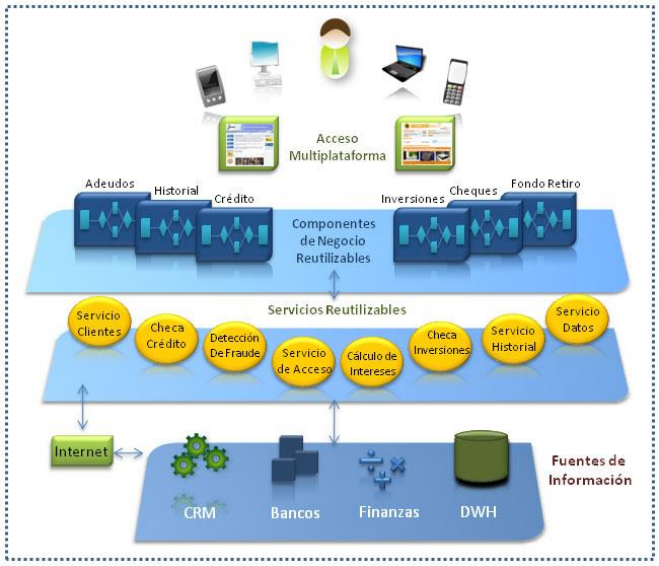

Ilustración 3. Ejemplo de una arquitectura SOA con inspiración en capas

Por otro lado, la arquitectura de microservicios también tiene una clara inspiración en la arquitectura de capas, en la cual cada capa se encarga de realizar una tarea. Y por último está la arquitectura sin servidor que tiene una clara inspiración en capas:

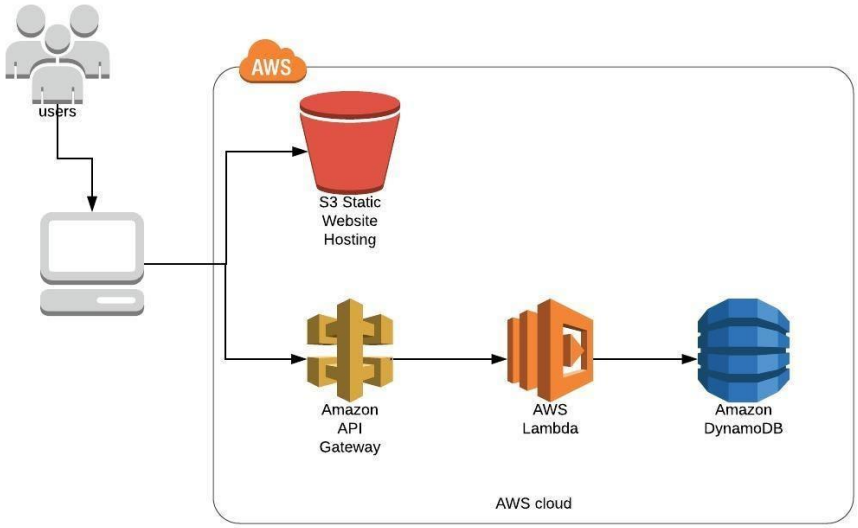

Ilustración 4. Ejemplo de una arquitectura serverless

2.3. Casos de uso relevantes

La arquitectura en capas es una opción ideal para muchos casos de uso en el desarrollo de software, esto se debe principalmente a la separación de responsabilidades que nos permite obtener la arquitectura en capas puesto que permite abstraer la complejidad del software de manera que las tareas necesarias para cumplir con una funcionalidad del negocio puedan encargarse a una capa que se encargue de cumplir con esta funcionalidad de la mejor manera.

2.3.1. 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.

2.3.2. 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.

2.3.3. Aplicaciones móviles

La arquitectura en capas también es una buena opción para aplicaciones móviles. Al separar la capa de presentación de la lógica de negocio, se puede lograr una mayor reutilización del código y la creación de aplicaciones multiplataforma. La capa de presentación puede ser específica para cada plataforma, mientras que la capa de negocios y la capa de datos se pueden reutilizar en todas las plataformas.

2.4. Ventajas y desventajas

2.4.1. Ventajas.

- Modularidad: La arquitectura en capas permite una división clara y definida de la funcionalidad, lo que facilita la creación de componentes reutilizables y modulares [3]
- Flexibilidad: La separación de responsabilidades en diferentes capas simplifica la implementación de cambios en el software, ya que los cambios pueden realizarse en una capa sin afectar las demás [3]
- Facilita el mantenimiento: al separar las diferentes capas de un sistema, el mantenimiento se vuelve más sencillo, ya que las modificaciones en una capa no afectan a las demás [2]
- Escalabilidad: al separar las capas de un sistema, se pueden agregar o quitar capas según sea necesario, lo que permite adaptar el sistema a las necesidades cambiantes de los usuarios y de la empresa [2].
- Abstracción: La arquitectura en capas permite la creación de una abstracción clara de la funcionalidad, lo que puede facilitar la comprensión del software y reducir la complejidad [3]

2.4.2. Desventajas.

La arquitectura en capas es una solución popular para el diseño de software debido a su modularidad y facilidad de mantenimiento. Sin embargo, también presenta algunos desafíos que deben ser considerados.

Uno de los principales desafíos es la sobrecarga de comunicación entre las diferentes capas, lo que puede impactar negativamente en el rendimiento del software. Además, la rigidez de la arquitectura en capas

puede dificultar la adaptación del software a cambios en los requisitos del usuario. La arquitectura en capas también puede aumentar la complejidad del software y hacer que sea más difícil de entender y mantener. Además, la implementación de la arquitectura en capas puede ser costosa, ya que requiere la creación de componentes adicionales.

Otro desafío importante es la dificultad para escalar verticalmente el software, ya que cada capa puede requerir más recursos para funcionar correctamente. En general, la arquitectura en capas es una solución efectiva para el diseño de software, pero es importante considerar estos desafíos al elegir un estilo arquitectural para un proyecto específico.

2.5. Principios SOLID Asociados

A continuación, los principios SOLID asociados a la arquitectura en capas:

- Principio de responsabilidad única (SRP): La arquitectura en capas promueve la separación de responsabilidades al dividir la aplicación en capas lógicas con responsabilidades específicas. Esto es coherente con el SRP, que establece que una clase o módulo debe tener una única razón para cambiar.
- Principio abierto/cerrado (OCP): La arquitectura en capas puede facilitar la aplicación del OCP al permitir la extensión o modificación de la funcionalidad en una capa específica sin afectar las otras capas. Por ejemplo, se puede agregar una nueva funcionalidad a la capa de negocio sin cambiar la capa de presentación o de datos.
- Principio de inversión de dependencias (DIP): En una arquitectura en capas bien diseñada, las capas superiores dependen de abstracciones, no de implementaciones concretas en las capas inferiores. Esto es coherente con el DIP, que sugiere que los módulos de alto nivel no deben depender de los módulos de bajo nivel, sino de abstracciones.

2.6. Atributos de Calidad Asociados

Entre los atributos de calidad asociados con la arquitectura en capas se encuentran:

- Mantenibilidad: Al dividir la aplicación en capas lógicas, se promueve la separación de responsabilidades y se facilita la comprensión del código. Esto hace que la aplicación sea más fácil de mantener y actualizar en el futuro.
- Modularidad: La arquitectura en capas fomenta la modularidad, permitiendo a los desarrolladores dividir la aplicación en módulos independientes y enfocados en tareas específicas. Esto simplifica el desarrollo y facilita la reutilización de código.
- Extensibilidad: La separación de responsabilidades en diferentes capas permite una mayor extensibilidad de la aplicación, ya que es más fácil agregar o modificar funcionalidades en una capa específica sin afectar a las demás.

- Reemplazabilidad: La arquitectura en capas facilita la sustitución de componentes o servicios en una capa específica sin afectar a otras capas. Esto es útil cuando se desea cambiar una implementación o tecnología subyacente en una parte de la aplicación.
- Testabilidad: Al organizar la aplicación en capas, se simplifica el proceso de pruebas, ya que se pueden realizar pruebas unitarias y de integración en cada capa por separado, reduciendo la complejidad de las pruebas y mejorando la calidad del software.
- Desacoplamiento: La arquitectura en capas promueve el desacoplamiento entre componentes y servicios, lo que reduce las dependencias y facilita el desarrollo y mantenimiento de la aplicación.
- Interoperabilidad: La arquitectura en capas permite una mayor interoperabilidad entre sistemas, ya que las capas de servicios y de integración pueden exponer interfaces estandarizadas para interactuar con otros sistemas o servicios.
- Seguridad: Al separar la lógica de la aplicación en diferentes capas, se pueden implementar políticas de seguridad y control de acceso específicas para cada capa, mejorando la seguridad global de la aplicación.

3. React (TypeScript)

3.1. Definición, historia y evolución

3.1.1. Definición.

React es una biblioteca de JavaScript de código abierto desarrollada por Facebook para crear interfaces de usuario (UI) de aplicaciones web de una sola página (SPA), donde la interfaz de usuario se actualiza dinámicamente sin necesidad de cargar una página nueva. React se centra en la creación de componentes reutilizables y modulares, utilizando un modelo de programación declarativo y un enfoque basado en el estado y las propiedades para la gestión de datos.

3.1.2. Historia.

React fue creado por Jordan Walke, un ingeniero de software de Facebook, en 2011 y se hizo de código abierto en 2013. React fue diseñado para abordar las dificultades en la gestión del estado de la interfaz de usuario y el rendimiento en aplicaciones web de gran escala. Desde su lanzamiento, React ha ganado una amplia adopción en la industria y ha impulsado el desarrollo de una serie de bibliotecas y herramientas complementarias, como Redux, React Router y Next.js.

3.1.3. Evolución.

A lo largo de los años, React ha experimentado varias mejoras y cambios en su API y arquitectura. Algunas de las evoluciones clave incluyen:

- React Native (2015): React Native es un marco de desarrollo de aplicaciones móviles basado en React, que permite a los desarrolladores construir aplicaciones nativas para iOS y Android utilizando JavaScript y la misma arquitectura basada en componentes que React.

- Introducción de los componentes funcionales y Hooks (2019): Con la versión 16.8 de React, se introdujeron los Hooks, que permiten a los desarrolladores utilizar características de estado y ciclo de vida en componentes funcionales, en lugar de los componentes basados en clases. Esta actualización condujo a un cambio en la forma en que se crean y gestionan los componentes en React, favoreciendo una sintaxis más concisa y funcional.

Actualmente React corresponde a la segunda tecnología web más usada (42.62%) según la encuesta de stack overflow de 2022, después de Node.js.

3.2. Casos de uso Relevantes

React puede ser aplicado en diversas situaciones como:

- Desarrollo de aplicaciones web de una sola página (SPA): React permite desarrollar aplicaciones web ricas en interacción donde la navegación y actualización de la interfaz de usuario se realiza sin recargar la página completa.
- Aplicaciones con interfaces de usuario complejas y dinámicas: React se destaca en manejar aplicaciones con interfaces de usuario que cambian frecuentemente o que requieren una gran cantidad de interacciones por parte del usuario, como paneles de administración, herramientas de edición y aplicaciones de visualización de datos.
- Desarrollo de aplicaciones móviles con React Native: React Native, basado en React, permite desarrollar aplicaciones móviles para iOS y Android utilizando JavaScript y la misma arquitectura de componentes que React.
- Aplicaciones con rendimiento crítico: React utiliza un algoritmo de reconciliación eficiente y un modelo de actualización basado en el estado, lo que puede ayudar a mejorar el rendimiento de aplicaciones web con una gran cantidad de actualizaciones en tiempo real o manipulaciones del DOM.
- Desarrollo de componentes reutilizables: React fomenta la creación de componentes modulares y reutilizables, lo que permite a los desarrolladores compartir y reutilizar código en diferentes partes de una aplicación o entre aplicaciones.
- Integración en aplicaciones web existentes: React se puede utilizar de forma incremental en aplicaciones web existentes para mejorar la interacción del usuario o modernizar partes específicas de la interfaz de usuario.
- Aplicaciones basadas en la arquitectura de micro-frontends: React se adapta bien al enfoque de micro-frontends, donde una aplicación se divide en múltiples fragmentos de frontend más pequeños que pueden desarrollarse y desplegarse de forma independiente.

3.3. Ventajas y desventajas

3.3.1. Ventajas.

React ofrece varias ventajas en el desarrollo de aplicaciones web y móviles, algunas de las cuales incluyen:

- Componentes reutilizables: React se basa en un enfoque de componentes modulares, lo que permite a los desarrolladores crear y reutilizar componentes en diferentes partes de la aplicación o entre aplicaciones. Esto facilita el mantenimiento y la coherencia en la interfaz de usuario.
- Rendimiento: React utiliza un algoritmo de reconciliación eficiente y un DOM virtual para minimizar las actualizaciones costosas del DOM real. Esto mejora el rendimiento de las aplicaciones, especialmente aquellas con una gran cantidad de actualizaciones en tiempo real o manipulaciones del DOM.
- Programación declarativa: React adopta un enfoque de programación declarativa, lo que facilita la lectura y el mantenimiento del código. En lugar de describir cómo se deben realizar los cambios en la interfaz de usuario, los desarrolladores especifican cómo debe verse la interfaz de usuario en función del estado de la aplicación.
- Ecosistema y comunidad: React cuenta con un amplio ecosistema de bibliotecas, herramientas y componentes de terceros, así como una comunidad activa de desarrolladores. Esto proporciona una gran cantidad de recursos y soluciones preexistentes que pueden facilitar y acelerar el desarrollo de aplicaciones.
- Flexibilidad: React es una biblioteca centrada en la vista y se puede utilizar con diferentes arquitecturas y soluciones de gestión de estado, como Redux, MobX o el propio Context API de React. Esto permite a los desarrolladores elegir la solución que mejor se adapte a sus necesidades.
- Compatibilidad con React Native: React Native, basado en React, permite a los desarrolladores crear aplicaciones móviles nativas para iOS y Android utilizando JavaScript y la misma arquitectura de componentes que React. Esto facilita la reutilización de lógica y componentes entre aplicaciones web y móviles.
- Desarrollo incremental: React se puede introducir en aplicaciones web existentes de forma incremental, lo que permite mejorar o modernizar partes específicas de la interfaz de usuario sin tener que reescribir toda la aplicación.
- Soporte de grandes empresas: React fue desarrollado y es mantenido por Facebook, lo que garantiza un soporte sólido y continuo. Además, muchas otras grandes empresas, como Airbnb, Netflix y Uber, también utilizan React en sus aplicaciones, lo que refuerza su relevancia y confiabilidad en la industria.

3.3.2. Desventajas.

A pesar de sus numerosas ventajas, React también tiene algunas desventajas que pueden afectar su adecuación para ciertos proyectos o equipos de desarrollo:

- Curva de aprendizaje: Aunque React en sí mismo no es difícil de aprender, la combinación de conceptos como componentes, estado, ciclo de vida y JSX puede ser confusa para los desarrolladores que se enfrentan a React por primera vez. Además, la elección de bibliotecas y patrones de arquitectura adicionales, como Redux o Context API, puede complicar aún más el proceso de aprendizaje.

- Verbosidad del código: En algunos casos, React puede resultar en un código más verboso en comparación con otras soluciones, especialmente cuando se trata de manejar el estado local y las interacciones de componentes. Sin embargo, la introducción de Hooks en React 16.8 ha reducido en gran medida esta verbosidad.
- Cambio frecuente de mejores prácticas: La comunidad y el ecosistema de React evolucionan rápidamente, y las mejores prácticas y patrones de diseño pueden cambiar con el tiempo. Esto puede ser desafiante para los desarrolladores que intentan mantenerse al día con las tendencias y actualizaciones.
- Ecosistema abrumador: Si bien el amplio ecosistema de React es una ventaja, también puede ser abrumador para los nuevos desarrolladores que deben tomar decisiones sobre qué bibliotecas y herramientas utilizar en sus proyectos. Además, algunas bibliotecas pueden quedar obsoletas o perder soporte con el tiempo, lo que podría afectar la mantenibilidad y la escalabilidad del proyecto.
- Inyección de dependencias limitada: React no proporciona un sistema de inyección de dependencias incorporado como otros frameworks, como Angular. Aunque esto puede ser una ventaja en términos de simplicidad, también puede dificultar la organización y gestión de dependencias en aplicaciones de gran tamaño.
- Tamaño de la biblioteca: Aunque React en sí mismo no es muy grande en términos de tamaño, el uso de bibliotecas y herramientas adicionales, como Redux y React Router, puede aumentar el tamaño final del paquete de la aplicación. Esto puede afectar el rendimiento y la velocidad de carga de la aplicación, especialmente en conexiones lentas.

3.4. Principios SOLID Asociados

A continuación, algunas conexiones entre los principios SOLID y React:

- Principio de responsabilidad única (SRP): En React, este principio se puede aplicar al diseño de componentes. Cada componente debe tener una única responsabilidad y estar enfocado en una sola tarea. Esto facilita la reutilización, el mantenimiento y la comprensión de los componentes individuales.

- Principio abierto/cerrado (OCP): Los componentes de React se pueden considerar "abiertos para extensión" pero "cerrados para modificación" en el sentido de que se pueden extender mediante composición en lugar de modificar el componente original. Por ejemplo, es posible envolver un componente existente con otro componente de mayor orden (HOC) para agregar nueva funcionalidad sin cambiar el código del componente original.

- Principio de sustitución de Liskov (LSP): Aunque el LSP se aplica específicamente a la herencia en programación orientada a objetos, en React, se puede considerar en términos de composición y uso de props. Al utilizar la composición y pasar props a los componentes, es posible crear componentes más genéricos que pueden ser fácilmente sustituidos por otros componentes con interfaces de props similares sin afectar la funcionalidad del sistema.

3.5. Atributos de Calidad Asociados

React se ha ganado una buena reputación por su capacidad de mejorar atributos de calidad en aplicaciones que lo utilizan, entre estos se encuentran:

- Mantenibilidad: React fomenta la creación de componentes modulares y reutilizables, lo que facilita la comprensión, modificación y extensión del código en el futuro. Además, React adopta un enfoque de programación declarativa que mejora la legibilidad y comprensión del código.
- Rendimiento: React utiliza un DOM virtual y un algoritmo de reconciliación eficiente para minimizar las actualizaciones costosas en el DOM real. Esto puede mejorar significativamente el rendimiento de aplicaciones web, especialmente aquellas con una gran cantidad de actualizaciones en tiempo real o manipulaciones del DOM.
- Escalabilidad: La arquitectura basada en componentes y el enfoque modular de React facilitan la escalabilidad de aplicaciones. Los desarrolladores pueden agregar, modificar o eliminar fácilmente componentes y características sin afectar otras partes del sistema.
- Reutilización de código: React permite a los desarrolladores crear componentes reutilizables que se pueden compartir y utilizar en diferentes partes de la aplicación o entre aplicaciones. Esto puede mejorar la coherencia de la interfaz de usuario y reducir la duplicación de código y esfuerzo de desarrollo.
- Flexibilidad: React es una biblioteca centrada en la vista y se puede utilizar con diferentes arquitecturas y soluciones de gestión de estado, como Redux, MobX o el propio Context API de React. Esto permite a los desarrolladores elegir la solución que mejor se adapte a sus necesidades y proporciona una base sólida para adaptarse a los cambios en los requisitos del proyecto.
- Portabilidad: Con React Native, los desarrolladores pueden crear aplicaciones móviles nativas para iOS y Android utilizando JavaScript y la misma arquitectura de componentes que React. Esto facilita la reutilización de lógica y componentes entre aplicaciones web y móviles, mejorando la coherencia y reduciendo el tiempo de desarrollo.
- Interoperabilidad: React se puede utilizar de forma incremental en aplicaciones web existentes, lo que permite a los desarrolladores mejorar o modernizar partes específicas de la interfaz de usuario sin tener que reescribir toda la aplicación.

4. TypeScript

4.1. Definición, historia y evolución

4.1.1. Definición.

TypeScript es un lenguaje de programación de código abierto desarrollado por Microsoft que se basa en JavaScript y agrega tipado estático opcional. TypeScript extiende la sintaxis de JavaScript, lo que permite a los desarrolladores definir y usar tipos de datos, interfaces y clases en su código. El código TypeScript

se compila en JavaScript, lo que significa que puede ser ejecutado en cualquier entorno compatible con JavaScript, como navegadores web, Node.js y dispositivos IoT.

4.1.2. Historia y evolución.

- 2010: Anders Hejlsberg, conocido por su trabajo en Turbo Pascal, Delphi y C#, comenzó a trabajar en TypeScript como un proyecto en Microsoft. Su objetivo era abordar las limitaciones de JavaScript en cuanto a escalabilidad y mantenibilidad en proyectos grandes y complejos.
- Octubre de 2012: Microsoft anunció oficialmente TypeScript y lanzó la versión 0.8 como un proyecto de código abierto en GitHub.
- 2013-2014: TypeScript continuó evolucionando, con varias actualizaciones que mejoraron el rendimiento, la compatibilidad con ECMAScript y la experiencia del desarrollador. La versión 1.0 se lanzó en abril de 2014.
- 2015: TypeScript 1.5 introdujo soporte para módulos ES6 y decoradores, lo que permitió a los desarrolladores adoptar las características más recientes de JavaScript en sus proyectos TypeScript. También marcó el comienzo de un ciclo de lanzamiento más rápido para el lenguaje.
- 2016: TypeScript 2.0 fue lanzado, introduciendo características significativas como control de flujo de tipos, tipado no nulo y tipos de utilidad, lo que mejoró la capacidad del lenguaje para detectar y prevenir errores en tiempo de compilación.
- 2017-2019: TypeScript siguió evolucionando, lanzando versiones con mejoras en el rendimiento, la ergonomía del lenguaje y la compatibilidad con las últimas características de JavaScript. Algunas de las características notables incluyen el soporte para módulos dinámicos, asignaciones de tipos condicionales y operadores de propagación.
- 2020: TypeScript 4.0 fue lanzado, mejorando la experiencia del desarrollador con refinamientos en la inferencia de tipos, mensajes de error más claros y nuevas capacidades de refactorización.

Desde su lanzamiento en 2012, TypeScript ha experimentado un crecimiento constante en adopción y popularidad. Hoy en día, TypeScript es ampliamente utilizado en la industria del software y es el lenguaje de elección para muchos proyectos y frameworks, como Angular y NestJS. La comunidad de TypeScript sigue creciendo y evolucionando, y se espera que el lenguaje siga mejorando y expandiéndose en el futuro.

Según la encuesta de stack overflow de 2022, TypeScript es el quinto lenguaje de programación más usado después de JavaScript, HTML/CSS, SQL y Python.

4.2. Ventajas y desventajas

4.2.1. Ventajas.

- Tipos estáticos: TypeScript introduce tipos estáticos opcionales, lo que mejora la detección de errores en tiempo de compilación y facilita la detección de problemas antes de que el código se ejecute en el navegador o en el servidor.

- Mejor autocompletado e información sobre herramientas: El soporte de tipos estáticos en TypeScript facilita la experiencia de desarrollo al proporcionar autocompletado inteligente y documentación en línea en entornos de desarrollo integrados (IDE) y editores de código.
- Compatibilidad con JavaScript: TypeScript es un superconjunto de JavaScript, lo que significa que cualquier código JavaScript válido también es válido en TypeScript. Esto permite a los desarrolladores adoptar TypeScript gradualmente e integrarlo en proyectos existentes de JavaScript.
- Soporte para características futuras de JavaScript: TypeScript admite características propuestas para futuras versiones de JavaScript antes de que estén ampliamente disponibles en los navegadores, lo que permite a los desarrolladores utilizar las últimas características del lenguaje sin preocuparse por la compatibilidad.
- Herramientas y ecosistema: TypeScript es compatible con muchas bibliotecas y marcos populares de JavaScript, y tiene un ecosistema en crecimiento de herramientas y recursos de la comunidad.

4.2.2. Desventajas.

- Curva de aprendizaje: Aprender TypeScript puede llevar tiempo, especialmente para aquellos que no están familiarizados con la programación orientada a objetos y los tipos estáticos.
- Tiempo de compilación: TypeScript requiere un paso adicional de compilación para convertir el código TypeScript en JavaScript, lo que puede aumentar el tiempo de desarrollo y ralentizar el proceso de construcción.
- Verbosidad: El uso de tipos estáticos y anotaciones en TypeScript puede resultar en un código más largo y verboso en comparación con JavaScript puro.
- Actualizaciones de bibliotecas y marcos: Aunque TypeScript es compatible con muchas bibliotecas y marcos populares, puede haber retrasos en la actualización de las definiciones de tipos cuando las bibliotecas o marcos se actualizan, lo que puede causar problemas de compatibilidad.

5. Axios

5.1. Definición, historia y evolución

5.1.1. Definición.

Axios es una biblioteca de JavaScript promesa basada en el cliente HTTP para el navegador y Node.js. Permite a los desarrolladores realizar fácilmente solicitudes HTTP asincrónicas y gestionar las respuestas. Axios es conocido por su API sencilla, manejo de errores y capacidad de interceptar solicitudes y respuestas.

5.1.2. Historia y evolución.

- Bla 2014-2015: Axios fue creado por Matt Zabriskie como una alternativa a la API Fetch nativa de JavaScript, que en ese momento era nueva y carecía de algunas características como la cancelación de solicitudes y el soporte para el progreso de las solicitudes.
- 2016-2017: Axios gana popularidad en la comunidad de desarrolladores de JavaScript debido a su facilidad de uso y características adicionales en comparación con la API Fetch. Además, Axios se convierte en la biblioteca recomendada para realizar solicitudes HTTP en proyectos basados en Vue.js, lo que aumenta aún más su adopción.
- 2018 en adelante: A medida que el ecosistema JavaScript sigue creciendo y evolucionando, Axios se mantiene como una de las principales bibliotecas para realizar solicitudes HTTP. La biblioteca sigue siendo actualizada y mantenida activamente, y ha ampliado su soporte a entornos como Node.js.

5.2. Ventajas y desventajas

5.2.1. Ventajas.

- Facilidad de uso: Axios tiene una API sencilla y fácil de entender, lo que facilita su adopción y uso para realizar solicitudes HTTP.
- Soporte para promesas: Axios se basa en promesas, lo que facilita la gestión de solicitudes y respuestas asincrónicas, y permite un mejor manejo de errores.
- Intercepción de solicitudes y respuestas: Axios permite interceptar y modificar solicitudes y respuestas antes de que se procesen, lo que puede ser útil para casos de uso como la autenticación o el registro de datos.
- Cancelación de solicitudes: Axios proporciona una forma de cancelar solicitudes en curso, lo cual es útil cuando se desea detener una solicitud debido a cambios en el estado de la aplicación o eventos del usuario.
- Soporte para navegadores y Node.js: Axios funciona tanto en navegadores como en entornos Node.js, lo que permite a los desarrolladores utilizar una única biblioteca para realizar solicitudes HTTP en diferentes plataformas.
- Amplia adopción: Axios es ampliamente utilizado en la comunidad de desarrolladores de JavaScript y es compatible con muchos marcos populares, como Vue.js y React.

5.2.2. Desventajas.

- Tamaño de la biblioteca: Aunque Axios no es una biblioteca particularmente grande, es más pesada que la API Fetch nativa de JavaScript. Para aplicaciones que buscan minimizar el tamaño del paquete, esto puede ser una desventaja.

- Redundancia con Fetch API: La API Fetch nativa de JavaScript se ha vuelto más capaz y madura con el tiempo, y en algunos casos, puede ser suficiente para las necesidades de una aplicación. En tales casos, Axios puede ser redundante.
- Mantenimiento y evolución: Aunque Axios es una biblioteca popular y ampliamente utilizada, su evolución y mantenimiento dependen en gran medida de la comunidad de desarrolladores y sus contribuciones.

6. Django

6.1. Definición, historia y evolución

6.1.1. Definición.

Django (Python) es un framework de desarrollo web de alto nivel escrito en Python que se utiliza para construir aplicaciones web escalables y seguras. Django proporciona una estructura de proyecto y una API que facilitan el desarrollo de aplicaciones web complejas, así mismo, este se enfoca en la reutilización de componentes y el mantenimiento de la estructura de una aplicación.

6.1.2. Historia.

Django fue creado en 2003 mientras sus creadores trabajaban en el sitio web 'Lawrence Journal-World'. El código fuente de Django se publicó bajo la licencia BSD de código abierto en 2005. Desde entonces, Django ha sido utilizado por una amplia variedad de organizaciones y sitios web, desde pequeñas empresas hasta grandes empresas como Instagram y Pinterest.

6.1.3. Evolución.

Django es un marco de desarrollo web de Python lanzado en 2005. Ha evolucionado constantemente para adaptarse a las necesidades de los desarrolladores, mejorando la seguridad, el rendimiento y las características. Con el tiempo, ha ganado soporte para múltiples bases de datos, Python 3, operaciones asíncronas y más. Hoy en día, Django es un marco popular y estable respaldado por una sólida comunidad de desarrolladores.

Según la encuesta de stack overflow de 2022, Django es la novena tecnología web más usada (14.65%).

6.2. Casos de uso relevantes

Django es un marco de desarrollo web versátil que se puede utilizar en una amplia variedad de casos de uso. Algunos de los casos de uso relevantes de Django incluyen:

- Sitios web y aplicaciones de contenido: Django es especialmente útil para crear sitios web basados en contenido, como blogs, revistas en línea, periódicos y portales de noticias, gracias a su potente sistema de plantillas y su interfaz de administración incorporada.
- Aplicaciones de comercio electrónico: Django se puede utilizar para desarrollar plataformas de comercio electrónico completas, desde la gestión de productos hasta la integración de pagos, utilizando paquetes como Django Oscar o Saleor.

- Aplicaciones de redes sociales: Django es adecuado para crear aplicaciones de redes sociales con características como la autenticación de usuarios, la gestión de perfiles, la publicación de contenido y las interacciones entre usuarios.
- Sistemas de gestión de relaciones con el cliente (CRM): Django puede ser la base para construir soluciones CRM personalizadas, que permiten a las empresas administrar sus interacciones con clientes y prospectos.
- Aplicaciones empresariales: Django es una opción sólida para el desarrollo de aplicaciones empresariales, incluyendo sistemas de gestión de proyectos, planificación de recursos empresariales (ERP) y sistemas de información de gestión (MIS).
- Aplicaciones de análisis de datos y visualización: Django puede ser utilizado para crear aplicaciones que procesan, analizan y visualizan datos, al integrarse con bibliotecas de análisis de datos de Python como Pandas, NumPy y Matplotlib.
- APIs RESTful: Django es ideal para crear APIs RESTful utilizando el paquete Django Rest Framework, que facilita el desarrollo de interfaces de programación de aplicaciones para aplicaciones web y móviles.

6.3. Ventajas y Desventajas

6.3.1. Ventajas.

- Desarrollo rápido de aplicaciones web: Django se enfoca en la reutilización de componentes y el mantenimiento de la estructura de una aplicación, lo que permite a los desarrolladores construir aplicaciones web rápidamente y con menos errores.
- Seguridad: Django incluye medidas de seguridad integradas, como protección contra ataques de CSRF y XSS, autenticación y autorización de usuarios, y verificación de contraseñas seguras.
- Escalabilidad: Django está diseñado para manejar grandes volúmenes de tráfico y datos, lo que lo convierte en una buena opción para aplicaciones web empresariales y de alta demanda.
- Comunidad: Django tiene una gran comunidad de desarrolladores activos que contribuyen a su desarrollo y proporcionan soporte y recursos para otros desarrolladores.

6.3.2. Desventajas.

- Curva de aprendizaje: Aunque Django está diseñado para ser fácil de usar, puede requerir de un proceso de aprendizaje para los desarrolladores que no tienen experiencia en el desarrollo web o en Python.
- Flexibilidad limitada: Django se enfoca en la reutilización de componentes y la estructura de la aplicación, lo que puede limitar la flexibilidad en algunas áreas del desarrollo de aplicaciones web.
- Base de datos única: Django está diseñado para trabajar con una única base de datos relacional, esto limita la opción de trabajar con bases de datos no relacionales, además de trabajar con varias bases de datos.

6.4. Principios SOLID Asociados

A continuación, la relación de Django con los principios SOLID:

- Principio de responsabilidad única (SRP): Django fomenta la separación de responsabilidades utilizando el patrón Modelo-Vista-Controlador (MVC). En este patrón, los modelos manejan la lógica de la base de datos, las vistas se encargan de la lógica del negocio y las plantillas manejan la presentación.
- Principio abierto/cerrado (OCP): Django permite extender y modificar fácilmente el comportamiento de las aplicaciones a través de la herencia de clases y la composición de componentes, como vistas y modelos, lo que facilita la aplicación del principio OCP.
- Principio de sustitución de Liskov (LSP): Django utiliza la herencia en sus clases base, como las vistas genéricas y los modelos, lo que permite a los desarrolladores crear clases derivadas que pueden sustituir a las clases base sin afectar la funcionalidad general de la aplicación.
- Principio de segregación de interfaces (ISP): Aunque Django no utiliza interfaces en el sentido estricto de la programación orientada a objetos, los desarrolladores pueden aplicar el principio ISP dividiendo la funcionalidad en vistas y clases más pequeñas y específicas, lo que reduce las dependencias innecesarias entre componentes.
- Principio de inversión de dependencias (DIP): Django promueve la inversión de dependencias al permitir la inyección de dependencias en la configuración de la aplicación, como el uso de motores de base de datos y servicios de correo electrónico intercambiables.

6.5. Atributos de Calidad Asociados

Atributos de Calidad asociados específicamente con Django incluyen:

- Mantenibilidad: Gracias al patrón Modelo-Vista-Controlador (MVC), Django promueve una estructura de código organizada y una separación clara de responsabilidades, lo que facilita la comprensión y la modificación del código a lo largo del tiempo.
- Reutilización de código: Django fomenta la reutilización de código a través de aplicaciones modulares y componentes como vistas genéricas y mixins. Esto permite a los desarrolladores construir aplicaciones más rápidamente y con menos duplicación de código.
- Seguridad: Django incluye varias características de seguridad integradas, como protección contra ataques de inyección SQL, falsificación de solicitudes entre sitios (CSRF) y protección contra ataques de secuencia de comandos entre sitios (XSS). Esto ayuda a garantizar que las aplicaciones desarrolladas con Django sean más seguras por defecto.
- Escalabilidad: Django es adecuado para aplicaciones web de todos los tamaños y puede manejar un gran número de solicitudes simultáneas mediante la implementación de técnicas de almacenamiento en caché y balanceo de carga.

- Portabilidad: Al ser un marco basado en Python, Django es altamente portable y puede ejecutarse en una amplia variedad de sistemas operativos y entornos de ejecución, lo que facilita la implementación y migración de aplicaciones.
- Compatibilidad con bases de datos: Django proporciona un sistema de abstracción de base de datos que permite a los desarrolladores trabajar con diferentes sistemas de bases de datos (como PostgreSQL, MySQL y SQLite) sin tener que cambiar significativamente el código de la aplicación.
- Extensibilidad: Django es altamente extensible y ofrece una amplia gama de paquetes y aplicaciones de terceros que pueden integrarse fácilmente en proyectos existentes para ampliar su funcionalidad.

7. Python

7.1. Definición, historia y evolución

7.1.1. Definición.

Python es un lenguaje de programación de alto nivel, interpretado y orientado a objetos. El ecosistema de Python presenta una gran cantidad de librerías que permiten realizar tareas en múltiples áreas.

7.1.2. Historia.

Fue creado en 1991 por Guido van Rossum en los Países Bajos. Su nombre proviene de la afición de van Rossum por el grupo cómico británico Monty Python.

7.1.3. Evolución.

Python, creado por Guido van Rossum en 1989, es un lenguaje de programación de alto nivel. Su evolución incluye el lanzamiento de Python 1.0 en 1994, Python 2.0 en 2000 y Python 3.0 en 2008. Desde entonces, Python 3 ha sido la única rama activa y el lenguaje ha crecido en popularidad, convirtiéndose en uno de los lenguajes más utilizados en todo el mundo.

Según la encuesta de stack overflow de 2022, Python es el cuarto lenguaje de programación más usado (48.07%) después de JavaScript, HTML/CSS y SQL.

7.2. Ventajas y Desventajas

7.2.1. Ventajas.

- Python es fácil de aprender y leer
- Python cuenta con una gran biblioteca estándar
- Python tiene una amplia comunidad de desarrolladores y una sintaxis clara y concisa.
- Python es multiplataforma y se puede utilizar en diferentes sistemas operativos.

7.2.2. Desventajas.

- Python puede ser más lento que otros lenguajes de programación

- A veces puede haber problemas de compatibilidad entre diferentes versiones de Python

8. REST

8.1. Definición, historia y evolución

8.1.1. Definición

REST es un estilo de arquitectura para sistemas distribuidos que se utiliza en aplicaciones web para transmitir datos a través de un protocolo que provea una interfaz uniforme como HTTP.

8.1.2. Historia

REST fue propuesto por primera vez por Roy Fielding en su tesis doctoral en 2000. Desde entonces, REST ha sido ampliamente adoptado como un estándar para la comunicación de datos en aplicaciones web.

8.1.3. Evolución

REST, propuesto por Roy Fielding en 2000, es un estilo arquitectónico para sistemas distribuidos. Ganó popularidad en la década de 2000, y su adopción creció rápidamente en la década de 2010, impulsada por aplicaciones móviles y marcos de API RESTful. Aunque REST sigue siendo popular, en la década de 2020 enfrenta competencia de alternativas como GraphQL.

8.2. Ventajas y Desventajas

8.2.1. Ventajas

- REST es fácil de entender y de implementar.
- REST es muy escalable, lo que significa que puede manejar grandes cantidades de tráfico y datos.
- REST es independiente del lenguaje de programación y del sistema operativo, lo que lo hace muy versátil.

8.2.2. Desventajas

- REST puede ser menos eficiente que otros protocolos de comunicación puesto que REST se ubica encima de otro protocolo como puede ser HTTP
- REST puede ser menos seguro que otros protocolos de comunicación, ya que la información se transmite en texto plano.
- REST puede ser menos adecuado para aplicaciones que requieren una transmisión de datos en tiempo real.

9. ORM

9.1. Definición, historia y evolución.

9.1.1. Definición.

ORM es una técnica de programación que permite mapear los objetos de un lenguaje de programación a las tablas de una base de datos relacional, eliminando así la necesidad de escribir consultas SQL en el código.

9.1.2. Historia.

ORM se originó a mediados de los años 90 como una solución para el mapeo de objetos en lenguajes de programación orientados a objetos y bases de datos relacionales, no obstante, fue en la década de los 2000 cuando su uso se popularizó gracias al uso de Hibernate en Java.

9.1.3. Evolución.

ORM (Object-Relational Mapping) es una técnica de programación que facilita la interacción entre sistemas orientados a objetos y bases de datos relacionales. La evolución de ORM incluye su aparición en la década de 1990, la creación de diversos frameworks ORM, como Hibernate y SQLAlchemy, y su adopción generalizada en el desarrollo de aplicaciones modernas.

9.2. Ventajas y desventajas

9.2.1. Ventajas.

- ORM hace que el código sea más fácil de leer y escribir, ya que elimina la necesidad de escribir consultas SQL en el código.
- ORM es útil para el mantenimiento de la base de datos, ya que puede automatizar las tareas de creación de tablas, actualización de esquemas, y otros procesos que normalmente son tediosos.
- ORM también puede mejorar el rendimiento, ya que puede optimizar automáticamente las consultas que se realizan a la base de datos.

9.2.2. Desventajas.

- ORM puede ser más lento que escribir consultas SQL a mano, especialmente cuando se trata de operaciones complejas en grandes conjuntos de datos.
- La implementación de ORM puede requerir una curva de aprendizaje, ya que es necesario entender cómo se mapean los objetos a las tablas de la base de datos.
- ORM puede ser menos flexible que escribir consultas SQL a mano, ya que a veces puede requerir una sintaxis específica para realizar ciertas operaciones.

10.PostgreSQL

10.1. Definición, historia y evolución

10.1.1. Definición

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto que se utiliza comúnmente en aplicaciones web. PostgreSQL proporciona funciones avanzadas de seguridad y escalabilidad que lo hacen adecuado para aplicaciones empresariales de alto nivel. PostgreSQL es conocido por su capacidad para manejar grandes conjuntos de datos y su soporte para características avanzadas como transacciones y funciones almacenadas.

10.1.2. Historia

PostgreSQL fue creado en 1986 como un proyecto de investigación en la Universidad de California en Berkeley. A lo largo de los años, ha evolucionado para convertirse en uno de los sistemas de gestión de bases de datos relacionales más avanzados y ampliamente utilizados disponibles. PostgreSQL es mantenido por una comunidad global de desarrolladores, y su código fuente está libremente disponible para su uso, modificación y distribución.

10.1.3. Evolución

PostgreSQL es un sistema de gestión de bases de datos relacionales de código abierto que se originó en la década de 1980 como un proyecto de investigación en la Universidad de California, Berkeley. Desde entonces, ha evolucionado hasta convertirse en uno de los sistemas más populares y avanzados, con una arquitectura modular y extensible, mejoras de rendimiento y escalabilidad, y soporte para una amplia variedad de lenguajes de programación y plataformas.

Según la encuesta de stack overflow de 2022, PostgreSQL es la segunda base de datos más usada (43.59%).

10.2. Ventajas y desventajas

10.2.1. Ventajas.

- Escalabilidad: PostgreSQL puede manejar grandes conjuntos de datos y es conocido por su capacidad para escalar horizontalmente, lo que significa que puede manejar un mayor volumen de datos a medida que crece una aplicación.
- Fiabilidad: PostgreSQL es conocido por su confiabilidad y estabilidad, lo que lo hace popular entre las empresas y organizaciones que necesitan una base de datos que funcione de manera consistente y sin fallas.
- Conformidad con los estándares: PostgreSQL cumple con los estándares ANSI y SQL, lo que significa que es compatible con una amplia variedad de herramientas y sistemas que también cumplen con estos estándares.

- Funciones avanzadas: PostgreSQL es conocido por su soporte para características avanzadas como transacciones y funciones almacenadas, que lo hacen popular entre los desarrolladores que necesitan una base de datos poderosa y escalable.

10.2.2. Desventajas.

- Curva de aprendizaje: PostgreSQL es un sistema de gestión de bases de datos avanzado, lo que significa que puede requerir una curva de aprendizaje para los desarrolladores que no tienen experiencia en el uso de bases de datos relacionales.
- Configuración: PostgreSQL puede ser más difícil de configurar que otros sistemas de gestión de bases de datos, lo que puede requerir más tiempo y recursos para su configuración inicial.
- Requisitos de hardware: PostgreSQL puede ser más exigente en términos de requisitos de hardware que otros sistemas de gestión de bases de datos, lo que puede requerir una inversión mayor en infraestructura.

10.3. Casos de uso relevantes

- Aplicaciones web: PostgreSQL es una opción popular para aplicaciones web debido a su capacidad de manejar grandes cantidades de datos, su flexibilidad y escalabilidad. Es utilizado por numerosos frameworks web como Django y Ruby on Rails.

- Aplicaciones móviles: PostgreSQL es una opción popular para el almacenamiento y la gestión de datos en aplicaciones móviles. Las aplicaciones móviles que utilizan PostgreSQL incluyen aplicaciones de redes sociales, aplicaciones de navegación y aplicaciones de comercio electrónico.

- Análisis de datos: PostgreSQL es capaz de manejar grandes conjuntos de datos y proporciona soporte para análisis de datos avanzados y herramientas de minería de datos. Es utilizado por empresas y organizaciones en sectores como finanzas, salud y tecnología para el análisis de datos.

- Sistemas de gestión de contenido: PostgreSQL se utiliza en sistemas de gestión de contenido como Drupal, WordPress y Joomla. Estos sistemas requieren una base de datos potente y escalable para almacenar y gestionar grandes cantidades de contenido y datos relacionados.

- Sistemas de información geográfica: PostgreSQL tiene soporte integrado para datos geoespaciales y es utilizado por sistemas de información geográfica para almacenar y analizar datos geográficos.

10.4. Principios SOLID Asociados

- Principio de responsabilidad única (SRP): PostgreSQL es una base de datos y, por lo tanto, su responsabilidad principal es almacenar y gestionar datos. Al aplicar el principio SRP, los desarrolladores deben asegurarse de que PostgreSQL solo se utilice para su propósito principal y se evite la sobrecarga de funciones adicionales que pueden reducir su rendimiento.

- Principio de abierto/cerrado (OCP): El principio OCP se refiere a la capacidad de extender el software sin modificar su código fuente original. En el caso de PostgreSQL, esto significa que los

desarrolladores pueden agregar nuevas funcionalidades y capacidades a través de extensiones sin tener que modificar directamente el código fuente del sistema.

- Principio de sustitución de Liskov (LSP): El principio LSP se refiere a la capacidad de reemplazar una clase con su subclase sin afectar el comportamiento del sistema. En el caso de PostgreSQL, esto significa que las nuevas versiones del sistema deben ser compatibles con versiones anteriores para que los usuarios puedan actualizar sin problemas.
- Principio de segregación de la interfaz (ISP): El principio ISP se refiere a la necesidad de dividir las interfaces en partes más pequeñas y cohesivas. En el caso de PostgreSQL, esto significa que las interfaces de usuario deben ser diseñadas para ser simples y cohesivas, y para satisfacer las necesidades específicas de los usuarios.
- Principio de inversión de dependencia (DIP): El principio DIP se refiere a la necesidad de que los módulos de un sistema dependan de abstracciones en lugar de implementaciones concretas. En el caso de PostgreSQL, esto significa que los desarrolladores deben depender de abstracciones de la base de datos en lugar de depender de una implementación concreta, lo que permite una mayor flexibilidad y modularidad.

10.5. Atributos de Calidad Asociados

- Fiabilidad: PostgreSQL es un sistema de gestión de bases de datos muy confiable. Ofrece características como la recuperación ante fallos y la replicación de datos para garantizar que los datos estén disponibles y accesibles en todo momento.
- Escalabilidad: PostgreSQL es muy escalable y puede manejar grandes cantidades de datos y transacciones simultáneas. Es capaz de crecer y adaptarse a las necesidades cambiantes de la aplicación y el negocio.
- Rendimiento: PostgreSQL es un sistema de gestión de bases de datos muy rápido y eficiente. Utiliza técnicas avanzadas de optimización de consultas y soporta la ejecución de múltiples consultas en paralelo.
- Seguridad: PostgreSQL ofrece múltiples capas de seguridad para proteger los datos almacenados en la base de datos. Ofrece funciones avanzadas de autenticación, cifrado de datos y auditoría de seguridad.
- Flexibilidad: PostgreSQL es muy flexible y puede ser utilizado en una amplia variedad de aplicaciones y proyectos. Es compatible con una amplia variedad de lenguajes de programación y plataformas, lo que lo hace muy adaptable a las necesidades de la aplicación.

11.SQL

11.1. Definición, historia y evolución

11.1.1. Definición.

SQL es un lenguaje de programación utilizado para gestionar y manipular bases de datos relacionales.

11.1.2. Historia.

SQL fue desarrollado por IBM en la década de 1970, y en 1986, SQL se convirtió en un estándar ANSI.

11.2. Ventajas y desventajas

11.2.1. Ventajas.

- SQL es un lenguaje muy potente y versátil que puede utilizarse para gestionar y manipular grandes cantidades de datos.
- SQL es fácil de aprender y de utilizar.
- SQL es independiente del sistema operativo.

11.2.2. Desventajas.

- SQL puede ser menos eficiente que otros lenguajes de programación para ciertas tareas.
- SQL puede ser más difícil de utilizar para tareas complejas que requieren una gran cantidad de código.
- SQL puede ser menos adecuado para aplicaciones que requieren una gestión de datos no estructurados o de grandes cantidades de datos no relacionales.

12.Estadísticas de uso

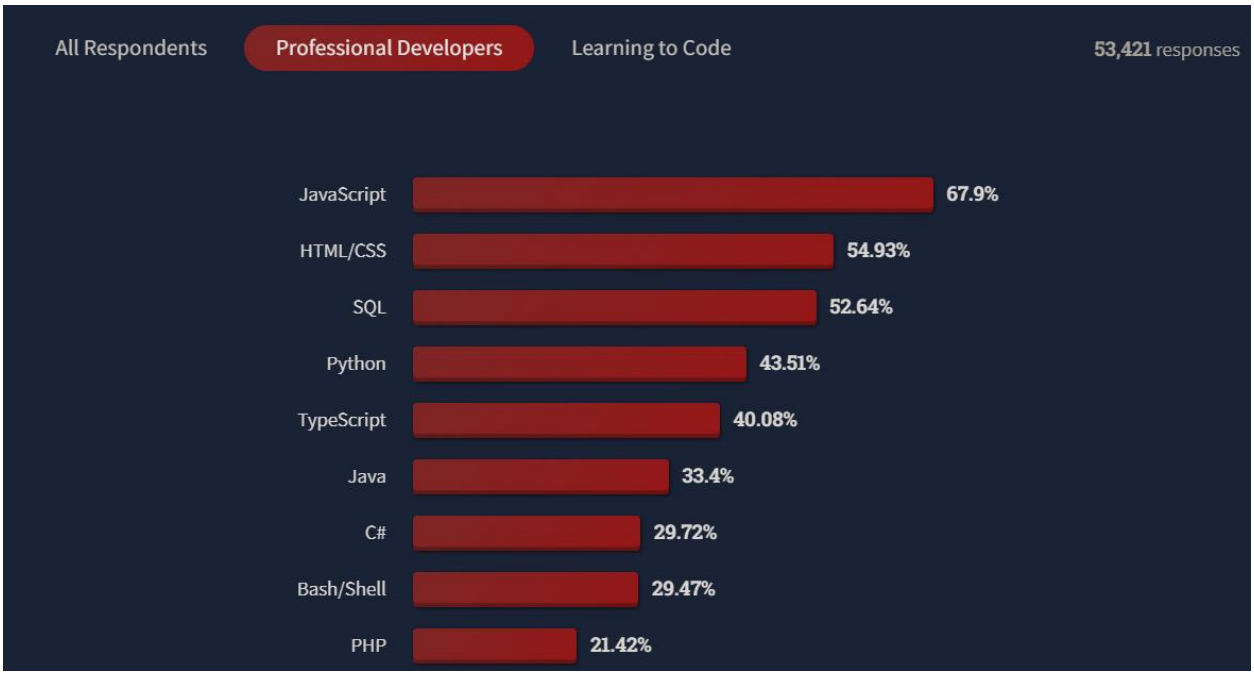

Ilustración 5. Imagen del uso de TypeScript, Python y SQL en el mundo tecnológico

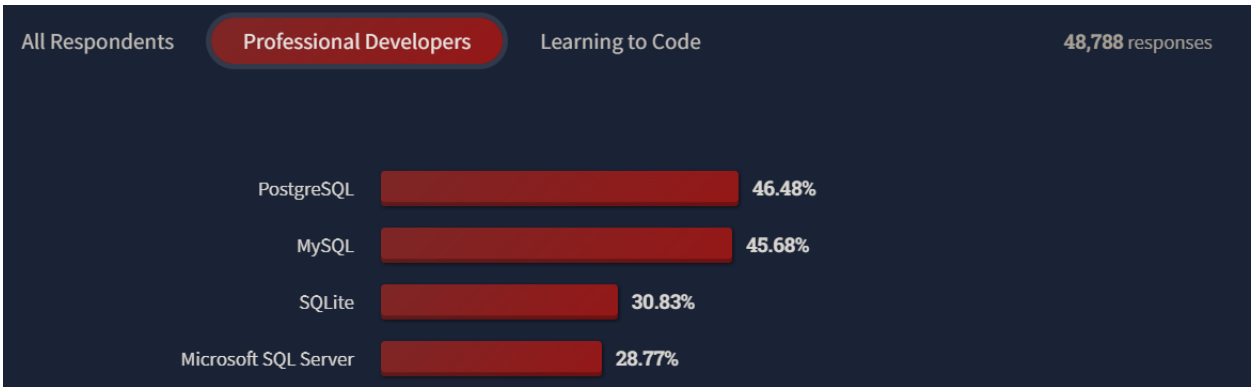

Ilustración 6. Imagen del uso de PostgreSQL en el mundo tecnológico

Ilustración 7.Imagen del uso de frameworks (Django) y tecnologías web (React) en el mundo tecnológico

Las imágenes mostradas anteriormente, muestran resultados de la encuesta desarrollada por stackoverflow en el año 2022 a diferentes programadores alrededor del mundo, y en diferentes áreas. Podemos notar que las tecnologías que conforman el stack tecnológico conformado por (React, Django, PostgreSQL) si bien no siempre aparecen en las primeras posiciones son ampliamente usadas en el mundo tecnológico.

12.1. Ejemplos de uso del stack tecnológico

12.1.1. Instagram

Instagram usa un stack conformado por múltiples tecnologias, entre las cuales podemos identificar React, PostgreSQL y Django. [4].

13. Ejemplo de uso práctico

Como ejemplo de uso, se plantea el desarrollo de una plataforma muy simple para administrar personas, donde se pueden agregar, buscar y editar algunos datos básicos de personas. Para ello, se decidió utilizar una arquitectura en capas, donde la capa de presentación sería desarrollada en React, la capa de aplicación en Django y la capa de datos en PostgreSQL.

La capa de presentación en React se enfocó en proporcionar una interfaz simple y fácil de usar. Se desarrollaron componentes reutilizables para construir la interfaz de usuario, que se comunica con la capa de aplicación en Django a través de una API RESTful. La capa de aplicación en Django proporciona las funciones necesarias para gestionar la información de los usuarios. Utilizando el ORM de Django, se implementó una estructura de modelos para manejar las operaciones de la base de datos y los datos del servidor. La capa de datos en PostgreSQL se diseñó para almacenar los datos de los usuarios, incluyendo nombre, edad, género y altura Se configuraron los modelos de Django para que interactuaran con la base de datos PostgreSQL de manera que los mismos pudiesen mapearse directamente a los objetos usados en Python. Es importante destacar que, en un inicio, se había considerado utilizar Oracle Express como base de datos para la plataforma. Sin embargo, se encontraron problemas de compatibilidad al intentar utilizarla con Django. Debido a esto, se decidió migrar a PostgreSQL para evitar estos problemas y garantizar la compatibilidad entre las diferentes capas de la aplicación.

13.1. Diseño

13.1.1. Diagrama de arquitectura de alto nivel

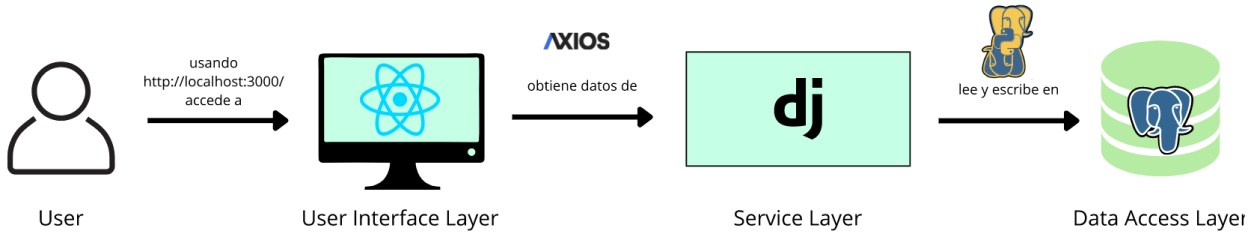

Ilustración 8. Diagrama de arquitectura de alto nivel.

En la ilustración anterior se evidencia la arquitectura de alto nivel del ejemplo. El usuario interactúa con la aplicación ingresando http://localhost:3000/ en el navegador, esto lo redirige a la capa de presentación donde se le permite al usuario usar los servicios de la capa de servicio. La capa de presentación consume a su capa inferior: la capa de servicio usando la librería axios, en esta capa se encuentra la lógica de las operaciones CRUD que se ofrecen al usuario en el front. Finalmente, la capa de servicio consume a la capa de acceso de datos usando psycopg2 para leer y escribir datos.

Sobre los lenguajes y frameworks, la capa de presentación está hecha con el framework React y el lenguaje TypeScript; la capa de servicio con Django y Python y; la capa de acceso de datos en PostgreSQL.

13.1.2. Diagrama de Arquitectura de Bajo Nivel

La arquitectura de bajo nivel se representa en los siguientes diagramas los cuales corresponden a los 4 niveles del modelo c4.

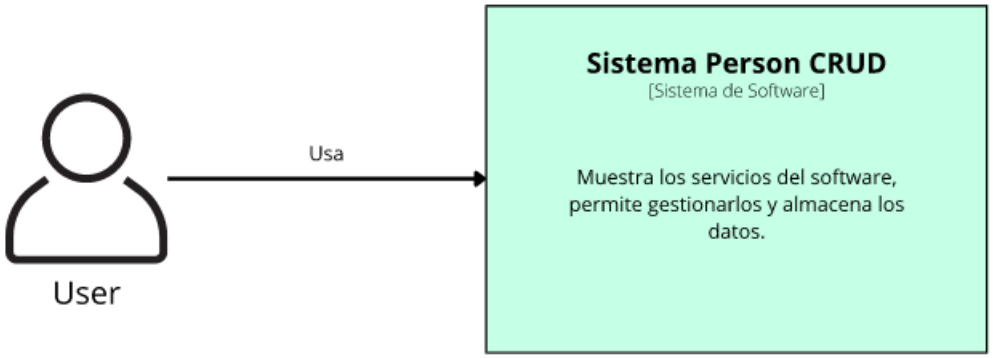

Ilustración 9. Diagrama de Contexto de Sistema de Software.

El diagrama de contexto de sistema de software muestra al único usuario que usa el sistema de software person crud, esta muestra los servicios, permite gestionarlos y almacena los datos relevantes para su funcionamiento.

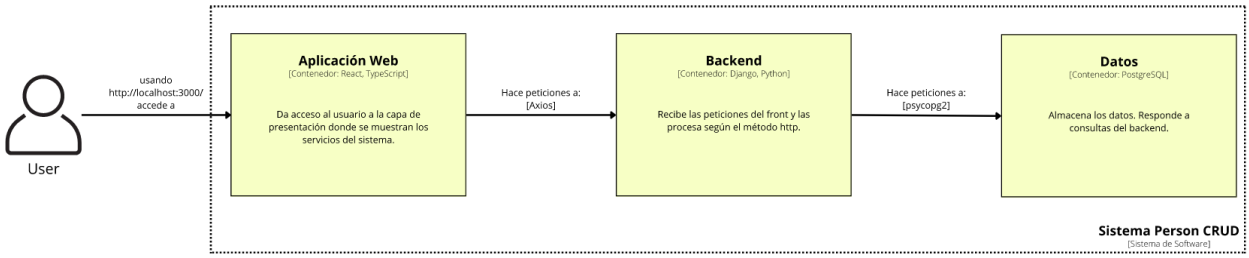

Ilustración 10. Diagrama de Contenedores.

Al hacer un *zoom* en el sistema de software podemos ver con claridad el estilo N-Capas. En este caso se tienen 3 capas, la primera corresponde al frontend de la aplicación, la segunda al backend y la tercera a los datos. El estilo se cumple al ver cómo cada capa solo consume a su capa inferior. El detalle del funcionamiento de cada componente y su interacción con otros se encuentra en la arquitectura de alto nivel.

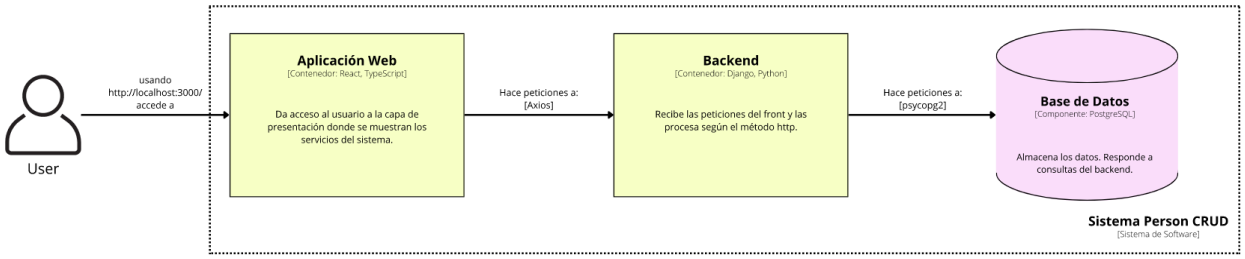

Ilustración 11. Diagrama de Componentes.

El diagrama de componentes se especifica en el contenedor de datos, el cual se compone de una base de datos relacional hecha con PostgreSQL.

Para mejor visualización de los diagramas visitar el siguiente enlace.

13.1.3. Diagrama de despliegue

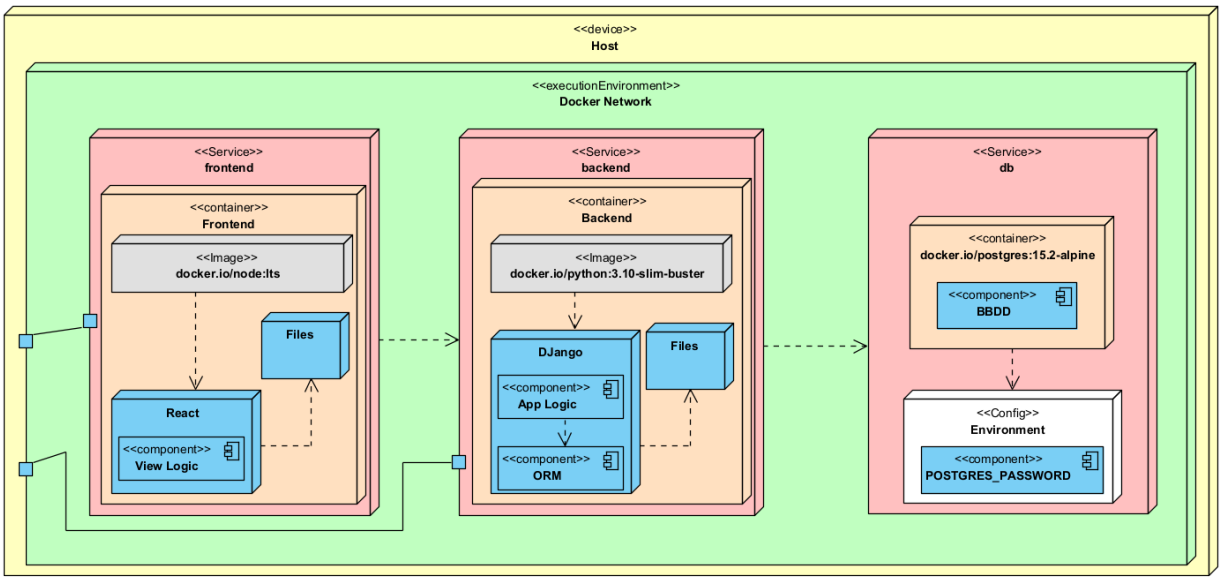

Ilustración 12. Diagrama de despliegue del caso de estudio propuesto

En este diagrama de despliegue, podemos ver tres capas que juntas funcionan como una sola aplicación. Cada capa tiene un propósito específico en la funcionalidad global de la aplicación. La capa de presentación es responsable de la interfaz de usuario y la interacción con el usuario final. La capa de aplicación es responsable de la lógica de negocios y la funcionalidad principal de la aplicación. Y la capa de datos es responsable del almacenamiento y la recuperación de datos para la aplicación.

Como herramienta para facilitar el despliegue se decidió usar Docker y específicamente se decidió usar Docker Compose puesto que con ayuda de esta herramienta pueden desplegarse múltiples servicios de manera coordinada. Otra ventaja que nos proporciona Compose son las redes con registros DNS puesto que permiten comunicar los contenedores de manera fácil, también el uso de las redes en Compose resulta particularmente importante cuando se ejecutan múltiples servicios en una misma red, ya que puede haber conflictos de puertos y direcciones IP [5].

En esta aplicación en particular, la capa de presentación está construida en React, la capa de aplicación en Django y la capa de datos en PostgreSQL. La combinación de estas tres capas permite que la aplicación funcione de manera fluida y coherente, con una interfaz de usuario atractiva y fácil de usar, una lógica de negocios eficiente y un almacenamiento y recuperación de datos confiable. Al utilizar Docker Compose, se puede asegurar que cada una de estas capas esté configurada y ejecutándose de manera correcta y coordinada, lo que garantiza la seguridad y la eficiencia en la comunicación entre contenedores. En resumen, el uso de Docker Compose y la combinación de estas tres capas permiten crear una aplicación efectiva y confiable que cumple con las necesidades del usuario final.

13.2. Implementación

Links a los repositorios y a los tags:

- Base de datos:
  - Repositorio: https://github.com/nclsbayona/presentacion1-arqui
  - Release: https://github.com/nclsbayona/presentacion1-arqui/releases/tag/release1
- Backend:
  - Repositorio: https://github.com/nclsbayona/presentacion1-arqui
  - Release: https://github.com/nclsbayona/presentacion1-arqui/releases/tag/release1
- Frontend:
  - Repositorio: https://github.com/nclsbayona/presentacion1-arqui
  - Release: https://github.com/nclsbayona/presentacion1-arqui/releases/tag/release1
- General:
  - Repositorio: https://github.com/nclsbayona/presentacion1-arqui
  - Release: https://github.com/nclsbayona/presentacion1-arqui/releases/tag/release1

Nota: Para correr el repositorio general, es necesario tener docker compose instalado, abrir el proyecto en Visual Studio Code y abrir dos terminales bash dentro de este.

1. Primero para subir el docker compose, se utiliza docker-compose up –build y hay esperar que cargue la base de datos y el backend (localhost:8000).
2. Luego, se ejecuta docker ps, se copia el id del frontend y se usa el comando docker exec -it [idCopiado] bash
3. Por último, se hace npm install y npm start para correrlo. Para poder visualizarlo en el navegador, hay que abrir una pestaña y escribir localhost:3000/api.

13.3. Conclusiones

Después de revisar la información general de React, TypeScript, Django, Axios, REST, Python, PostgreSQL, SQL y la arquitectura en capas, podemos concluir lo siguiente:

- React es una biblioteca de JavaScript utilizada para crear interfaces de usuario interactivas y eficientes, que se puede integrar en una arquitectura en capas para construir aplicaciones escalables y mantenibles.
- TypeScript es un lenguaje de programación que se basa en JavaScript, pero que agrega características de orientación a objetos y tipos estáticos, y puede utilizarse en conjunto con una arquitectura en capas para crear aplicaciones más seguras y mantenibles.
- Django es un framework de Python que se utiliza para crear aplicaciones web de alta calidad y escalables, que se pueden diseñar utilizando una arquitectura en capas para separar la lógica de negocio de la presentación y el almacenamiento de datos.
- Axios es una biblioteca de JavaScript utilizada para realizar solicitudes HTTP desde una aplicación web, que se puede integrar en una arquitectura en capas para separar la lógica de la interfaz de usuario de las operaciones de red.
- REST es un estilo arquitectónico para la construcción de servicios web que utiliza el protocolo HTTP para la transmisión de datos, y que se puede utilizar como parte de una arquitectura en capas para separar la lógica de negocio de la presentación y la gestión de recursos.
- Python es un lenguaje de programación de alto nivel y multipropósito utilizado en una amplia variedad de aplicaciones y proyectos, que se puede utilizar en una arquitectura en capas para crear aplicaciones escalables y mantenibles.
- PostgreSQL es un sistema de gestión de bases de datos relacionales de código abierto y muy versátil, que se puede utilizar en una arquitectura en capas para separar la lógica de negocio de la gestión de datos y mejorar la escalabilidad y el rendimiento.
- SQL es un lenguaje de programación utilizado para administrar y manipular datos en bases de datos relacionales, que se puede utilizar en una arquitectura en capas para separar la lógica de negocio de la gestión de datos y mejorar la escalabilidad y el rendimiento.

13.4. Lecciones aprendidas

- Si bien en un inicio se intentó usar Oracle Express como base de datos, la misma presentaba algunos problemas de compatibilidad con Django, por lo que se decidió usar PostgreSQL.
- La combinación de estas tecnologías puede ser muy poderosa y puede ayudar a construir aplicaciones robustas y escalables.
- La separación de la lógica de negocio, la presentación y el almacenamiento de datos en diferentes capas puede mejorar la eficiencia y la mantenibilidad de la aplicación.
- El uso de TypeScript puede mejorar la seguridad y la mantenibilidad de la aplicación, ya que permite la detección de errores de manera temprana y el control de tipos de datos.
- El uso de React puede mejorar la eficiencia y la experiencia de usuario de la aplicación, ya que permite la construcción de interfaces de usuario interactivas y eficientes.

- El uso de Django y PostgreSQL puede mejorar la escalabilidad y la eficiencia de la aplicación, ya que son sistemas altamente escalables y confiables para la gestión de bases de datos y aplicaciones web.
- El uso de REST y Axios puede mejorar la eficiencia de la aplicación en la transmisión de datos y la realización de solicitudes HTTP.
- El uso de SQL puede mejorar la eficiencia en la manipulación y gestión de datos en bases de datos relacionales.
- Se aprendió a usar docker compose para ejecutar la aplicación.
- Este stack no es común en situaciones de la vida real.

14. Bibliografía

[1] G. Booch, «A thread regarding the architecture of software-intensive systems,» Twitter.

[2] P. Bass, P. Clements y R. Kazman, Software architecture in practice, Addisson-Wesley.

[3] M. Richards, «Layered architecture,» de *Software architecture patterns*, O'Reilly Media, Inc, 2015.

[4] stackshare.io, «Instagram,» [En línea]. Available: https://stackshare.io/instagram/instagram.

[5] Docker, «Networking in Compose,» [En línea]. Available: https://docs.docker.com/compose/networking/.

[6] Microsoft, «Principios de la arquitectura,» 28 11 2022. [En línea]. Available: https://learn.microsoft.com/es-es/dotnet/architecture/modern-web-apps-azure/architectural-principles.

[7] C. Macias, «Principios SOLID,» 03 04 2019. [En línea]. Available: https://www.enmilocalfunciona.io/principios-solid/ .

[8] AWS, «AWS Architecture Center,» [En línea]. Available: https://aws.amazon.com/architecture/.

[9] reactiveprogramming.io, «Arquitectura en capa,» [En línea]. Available: https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/capas.

[10] R. C. advance, «Arquitectura en Capas – Análisis completo + Tradicional vs Modernas, DDD, DIP (Cap 5),» [En línea]. Available: https://rjcodeadvance.com/patrones-de-software-arquitectura-en-capas-analisis-completo-ejemplo-ddd-parte-5/.

[11] A. e. t. capas, «Spuzipedia,» [En línea]. Available: https://spuzi.github.io/Spuzipedia/arquitectura3capas/arquitectura3capas.html.

[12] R. F. C. Polini, «Metodologías Ágiles, Patrones de Diseño y Arquitectura de Software,» LinkedIn, [En línea]. Available: https://www.linkedin.com/pulse/metodolog%C3%ADas-%C3%A1giles-patrones-de-dise%C3%B1o-y-arquitectura-ricardo-fabio. .

[13] Barcelona Geeks, «System design netflix: Una Arquitectura Completa,» [En línea]. Available: https://barcelonageeks.com/diseno-de-sistemas-netflix-una-arquitectura-completa/.

[14] Platzi, «Patrón arquitectónico de capas/layers,» [En línea]. Available: https://platzi.com/tutoriales/1248-pro-arquitectura/5439-patron-arquitectonico-de-capas-layers/.

[15] Microsoft, «Estilos de arquitectura,» [En línea]. Available: https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/..

[16] J. Savolainen y V. Myllarniemi, «Layered architecture revisited — Comparison of research and practice,» 2009.

[17] Barcelona Geeks, «Diseño del sistema de la aplicación Uber: arquitectura del sistema Uber,» [En línea]. Available: https://barcelonageeks.com/diseno-del-sistema-de-la-aplicacion-uber-arquitectura-del-sistema-uber/.