FRONTEND ARCHITECTURE

Una introducción al mundo de la arquitectura software orientada al frontend.

2025
@iagolast

Frontend Architecture

Una introducción al mundo de la arquitectura software orientada al frontend.

Iago Lastra Rodríguez

Este libro está disponible en https://leanpub.com/frontend-architecture

Esta versión fue publicada el 2025-07-07 ISBN 9798289787552

¡Tuitea Este Libro!

¡Ayuda a Iago Lastra Rodríguez difundiendo la palabra sobre este libro en Twitter!

El hashtag sugerido para este libro es #frontend-architecture.

Descubre lo que otras personas están diciendo sobre el libro haciendo clic en este enlace para buscar este hashtag en Twitter:

#frontend-architecture

Índice general

1: Introducción

Existe una ley no escrita que establece que todo libro de programación que hable de arquitectura debe ofrecer su propia definición de dicho concepto. Como no quiero ser una excepción, mi definición será etimológica.

La palabra arquitectura proviene de añadir el sufijo *"-ura"*, que indica una actividad, a la palabra griega ἀρχιτέκτων, compuesta a su vez por dos partes: ἀρχι, que significa algo así como el que guía, el que manda, y τέκτων, que significa el que construye o, en este caso, el que programa.

Por ello, me gustaría proponer la siguiente definición de arquitectura de software:

> *"La arquitectura de software es la actividad que se encarga de guiar la construcción del software."*

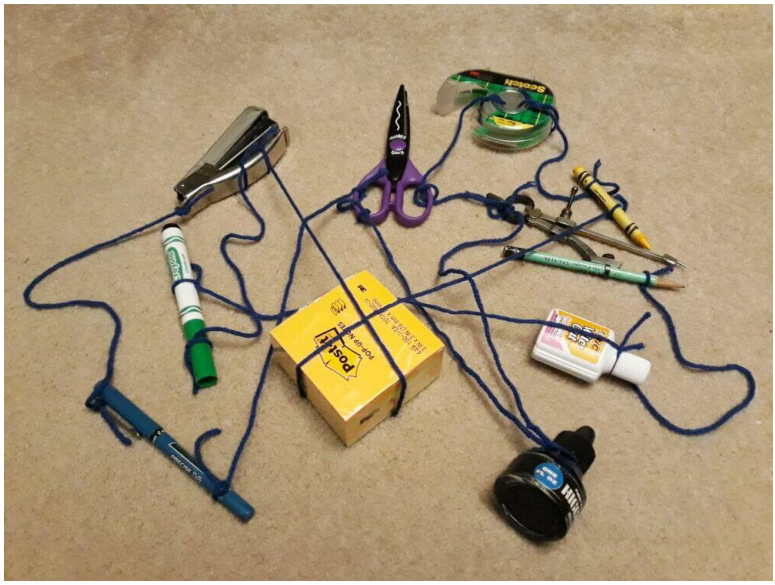
Esta definición es intencionadamente ambigua, pero tal y como dice Richard Feynman: *"Los nombres de las cosas no importan demasiado siempre que entendamos sus propiedades y cualidades"*. En el caso de la arquitectura de software, creo que lo fundamental es que entendamos su utilidad y su propósito en lugar de pelearnos por encontrar una definición perfecta. Quizá una mejor introducción para este libro sea: *"¿Para qué es necesaria una arquitectura?"*.

1.1: ¿Para qué es necesaria una arquitectura?

En mi opinión, **la gran ventaja de tener una buena arquitectura se reduce a** una cosa: **ser más productivos** y, por lo tanto, ahorrar nuestro tiempo y el dinero de la empresa. Un software con una buena arquitectura es más fácil de mantener; el proceso de incorporación de nuevos programadores se reduce considerablemente; las dependencias se actualizan sin demasiados problemas; las nuevas funcionalidades y correcciones de errores pueden realizarse en menos tiempo; y, en general, escribir tests será mucho más sencillo.

Uno de los mejores ejemplos que ayuda a visualizar la importancia de la arquitectura y de tener un código bien organizado está extraído de un post de Pusher[1]. En este post se presenta la siguiente imagen junto al texto:

> *"En la imagen de abajo, si quieres reemplazar las tijeras por un cuchillo, ¿qué tienes que hacer? Tienes que desatar los hilos que van al bolígrafo, al tintero, a la cinta y al compás. Luego, tienes que volver a atar esos artículos a un cuchillo. Tal vez eso funcione para el cuchillo, pero ¿qué pasa si el bolígrafo y la cinta dicen: 'Espera, necesitábamos tijeras'? Así que ahora el bolígrafo y la cinta no funcionan y hay que cambiarlos, lo que a su vez afecta a los objetos atados a ellos. Es un desastre..."*



Y lo compara con esta otra imagen, que simboliza una arquitectura «limpia»:

> *"Ahora, ¿cómo reemplazamos las tijeras? Solo tenemos que sacar el hilo de las tijeras de debajo de los post-it y añadir un nuevo hilo que se ata a un cuchillo. Mucho más fácil. A las notas post-it no les importa porque la cuerda ni siquiera estaba atada."*


Post-it
POP-UP NOTES

1.1.1: Dependencias

A través de esta sencilla metáfora, podemos hacernos una idea intuitiva del concepto de **dependencia**, el cual mencionaremos en múltiples ocasiones a lo largo de este texto. Por el momento, basta con decir que existe una dependencia entre una pieza de software "A" y una pieza "B" cuando la pieza "A" necesita conocer de alguna forma a la pieza "B" para realizar su función. En nuestra metáfora, las dependencias son esos pequeños hilos que conectan cada uno de los objetos de oficina. De la misma forma que para cambiar las tijeras de la imagen necesitaríamos desatar nudos en el bolígrafo y la grapadora, si quisiéramos realizar cambios en una pieza "B" de un software, es posible que tengamos que "desatar nudos" en las piezas que dependan de ella.

1.1.2: Abstracciones, detalles e interfaces

Podemos decir que **una buena arquitectura busca reducir al mínimo la carga cognitiva que soportan los desarrolladores**. Para lograr este objetivo, **las arquitecturas**, **entre otras cosas**, **organizan las dependencias mediante la generación de interfaces y abstracciones que ocultan los detalles de implementación**.

Pero, ¿qué es una interfaz? ¿Qué son los detalles de implementación? ¿Qué es una abstracción? ¿Cómo se relacionan estos conceptos con la arquitectura de software?

Vamos intentar ilustrar estos conceptos con unos ejemplos que espero que todo el mundo entienda.

En primer lugar, el ejemplo de los grifos en nuestras casas, donde podemos aplicar el concepto de dependencia y la idea de interfaces simplificadas en la arquitectura de software. En nuestras viviendas, la mayoría de las personas desconoce por completo el complejo funcionamiento del ciclo del agua, lo que conocemos y utilizamos de manera intuitiva son las simples interfaces de los grifos: al girar el grifo rojo obtenemos agua caliente, y al girar el grifo azul obtenemos agua fría.

Esta interfaz oculta los llamados **detalles de implementación** y nos brinda una experiencia de uso intuitiva y sin complicaciones. No necesitamos saber cómo funciona el sistema de calentamiento del agua, cómo se distribuye a través de las tuberías ni qué procesos están involucrados. Simplemente interactuamos con la interfaz que se nos presenta, girando el grifo correspondiente según nuestras necesidades.

Otro ejemplo se puede observar en el contexto de los automóviles. Como conductores, no necesitamos conocer los detalles de implementación del motor, como la cilindrada, el funcionamiento de los inyectores o la aleación utilizada para el alternador. Simplemente interactuamos con la interfaz formada por tres pedales y un volante para acelerar, frenar y controlar la dirección del vehículo.

Sin embargo, un mecánico que se encarga del mantenimiento y reparación de los automóviles necesita tener conocimientos diferentes sobre las características técnicas del motor, los sistemas de suspensión o los frenos, entre otros aspectos. Decimos que el mecánico se encuentra en otro "nivel de abstracción". Este otro nivel de abstracción tendrá también sus propias interfaces. Por ejemplo, si el mecánico se encuentra que la bombilla está

fundida puede pedir una pieza de repuesto y cambiarla simplemente conociendo su numero de serie y modelo, pero sin necesidad de conocer los detalles de implementación de la bombilla, como el proceso exacto de fabricación del vidrio o el tipo de gas que contiene. A su vez estos detalles de implementación si son conocidos por un ingeniero especializado en la fabricación de faros de automóviles, que necesita tener un conocimiento detallado sobre las propiedades de los materiales, la óptica, los requisitos reglamentarios de iluminación…

Vemos en todos estos ejemplos tres conceptos bien diferenciados: abstracciones, detalles e interfaces, que definimos a continuación:

- **Abstracción** (del latín *abstrahere*, 'alejar, sustraer, separar') es una operación mental destinada a aislar conceptualmente una propiedad o función concreta de un objeto, y pensar en qué consiste, ignorando otras propiedades del objeto en cuestión.
- **Detalle** es una parte, hecho o circunstancia que contribuye a formar o completar una cosa, pero no es indispensable en ella.
- **Interfaz** es la conexión funcional entre dos sistemas, programas, dispositivos o componentes de cualquier tipo, que proporciona una comunicación de distintos niveles de abstracción, permitiendo el intercambio de información.

Por lo general, cualquier conductor sabe que al girar el volante en el sentido de las agujas del reloj el coche girará a la derecha, y al presionar el pedal central el coche frenará. Para el nivel de abstracción "conductor", el volante y los pedales son una interfaz que expone el coche para poder conducirlo. Mientras que el modelo exacto de motor o el tamaño de las ruedas son detalles de implementación, ya que no es necesario conocer qué modelo de motor tiene un coche para conducirlo.

¿Qué pasaría si la industria automovilística hubiera decidido exponer interfaces erróneas? ¿Qué pasaría si para conducir un coche tuviéramos que conocer exactamente todos los detalles de implementación? En primer lugar, tendríamos que estudiar durante meses todos los detalles de cada uno de los modelos de coches existentes, y además, todo ese tiempo invertido no serviría de nada si cambiáramos de modelo de coche.

Ahora volvamos al *software*.

Como programadores, estaremos saltando continuamente entre diferentes niveles de abstracción. No es lo mismo la interfaz gráfica que la capa que

envía datos al servidor. Cuando hacemos un `console.log()`, estamos dando un salto de fe. Las funciones en sí mismas son una abstracción de la cual simplemente tenemos que conocer su interfaz (combinación de nombre y parámetros) para realizar una tarea. El cómo se realiza esa tarea es un detalle de implementación.

Por eso, uno de los objetivos de una buena arquitectura es organizar y definir correctamente las interfaces y los detalles que existirán en cada nivel de abstracción para gestionar las dependencias. Volviendo a nuestro ejemplo, ahora que conocemos la diferencia entre detalle e interfaz, podemos matizar y decir que **la dependencia entre dos piezas de *software* viene dada por la interfaz que las piezas exponen y el resto se considerarán detalles de implementación**.

Si logramos que la interfaz de una pieza se mantenga estable, actuará como una barrera que evita que los cambios se propaguen por el código cada vez que modifiquemos la implementación. Pongamos un ejemplo sencillo: Si consideramos una función que recibe un array de strings y devuelve el mismo array pero con todos los elementos en mayúsculas, podemos observar la distinción entre interfaz y implementación.

La interfaz de esta función se define por su nombre y sus parámetros: por ejemplo, podríamos llamarla `convertirAMayusculas(array)`. La interfaz establece qué se espera que haga la función, en este caso, convertir los elementos del array a mayúsculas y qué parámetros necesita para hacerlo, en este caso, un array de strings.

La implementación, por otro lado, se refiere a cómo se logra el resultado deseado. Hay diferentes enfoques posibles para implementar esta función. Una posible implementación podría hacer uso de la función `map`:

```
function convertirAMayusculas(array) {
  return array.map((elemento) => elemento.toUpperCase());
}
```

Otra posible implementación podría utilizar la función `forEach`:

```
function convertirAMayusculas(array) {
  const arrayEnMayusculas = [];
  array.forEach((elemento) => {
    arrayEnMayusculas.push(elemento.toUpperCase());
  });
  return arrayEnMayusculas;
}
```

En estos casos, la interfaz de la función `convertirAMayusculas` sigue siendo la misma: recibir un array de strings y devolver un array con los elementos en mayúsculas. La diferencia radica en la implementación, es decir, cómo se realiza internamente la transformación de los elementos.

Al mantener la interfaz constante, podemos cambiar la implementación según nuestras necesidades o preferencias, sin afectar el código que utiliza esta función. Esto nos puede llevar a la siguiente conclusión: **una interfaz estable actúa como una barrera que permite cambios en la implementación, sin afectar a otros componentes que dependen de ella**.

1.2: Otros aspectos de una buena arquitectura

La labor de un arquitecto, de acuerdo con Robert C. Martin, consiste en reducir al mínimo la cantidad de recursos humanos necesarios para mantener un sistema de *software*. En mi experiencia, el aspecto que más recursos consume es la gestión de dependencias, y es a esto a lo que dedicaremos más tiempo en este texto. No obstante, hay numerosos otros aspectos de una buena arquitectura que también resultaría beneficioso mencionar.

1.2.1: Desarrollo.

Una buena arquitectura prescribe una serie de herramientas y prácticas que ayudan a los desarrolladores a ser más productivos. Aunque sea una afirmación bastante genérica, intentaremos ilustrarla con algunos ejemplos.

¿Cómo de difícil es arrancar el proyecto después de clonar el repositorio? ¿Cuánto tiempo se tarda en instalar las dependencias? ¿Cuánto tiempo tarda un desarrollador en poder "montar" el proyecto? Conectarse a una base de datos de pruebas, tener usuarios de prueba, etc. ¿Cuánto tiempo se tarda en ejecutar las pruebas? ¿Existe un estándar de código? ¿Tiene herramientas que permitan el autoformateo? ¿Tiene pruebas que nos ayuden a encontrar errores? ¿Hay una política de nombrado de ramas? ¿Existe un flujo de trabajo con GIT? ¿Están actualizadas las dependencias?

Todos estos aspectos deben ser considerados por el equipo y deben optimizarse para conseguir una buena arquitectura para el proyecto.

1.2.2: Despliegue

Relacionado con los aspectos previamente mencionados, una buena arquitectura garantiza que el despliegue de un proyecto sea sencillo y eficiente. He presenciado proyectos en los que realizar un despliegue en producción es tan fácil como mezclar un commit en la rama principal. Sin embargo, también he visto casos en los que una persona debe ejecutar scripts manualmente durante horas, conectándose a un servidor a través de SSH y esperando no cometer errores en ningún paso.

La efectividad de un sistema de software está intrínsecamente ligada a su capacidad de ser desplegado fácilmente. Cuanto más alto es el costo de despliegue, menos útil resulta el sistema. Por tanto, un objetivo clave de la arquitectura de software debe ser facilitar un despliegue que pueda realizarse sin esfuerzo, idealmente en un único paso.

Desafortunadamente, la estrategia de despliegue rara vez se considera durante el desarrollo inicial. Esto conduce a arquitecturas que pueden facilitar el desarrollo, pero que complican significativamente el despliegue. Un ejemplo clásico es el uso de una arquitectura de microservicios en las etapas iniciales. Aunque esta arquitectura puede simplificar el desarrollo gracias a límites de componente claros y interfaces estables, el despliegue puede volverse un desafío debido al gran número de microservicios y la complejidad en configurar sus conexiones e iniciación.

Si los arquitectos hubieran considerado los problemas de despliegue desde el principio, podrían haber optado por menos servicios, una combinación de servicios y componentes en proceso, y métodos más integrados para gestionar las interconexiones. Por lo tanto, es esencial que la facilidad de despliegue sea una prioridad desde el inicio de cualquier proyecto de software, asegurando que la arquitectura no solo sea favorable para el desarrollo, sino también para una implementación eficiente y libre de errores.

1.2.3: Operación

Una arquitectura de software bien diseñada es esencial para una operación eficiente del sistema. Las preguntas clave que se deben considerar incluyen: ¿Cuánto tiempo se tarda en identificar y corregir un error? ¿Estamos al tanto del consumo de memoria y del número de clientes afectados por fallos? ¿Cuán ágil es el proceso para revertir cambios problemáticos? ¿Contamos con una observabilidad clara de los eventos del sistema? ¿Disponemos de métricas precisas sobre el comportamiento de los usuarios?

Si bien el impacto de la arquitectura en la operación del sistema suele ser menos dramático que en el desarrollo, despliegue y mantenimiento, su importancia no debe subestimarse. A menudo, los problemas operativos se solucionan incrementando los recursos de hardware, lo que puede ser efectivo a corto plazo. Sin embargo, este enfoque puede no ser sostenible a largo plazo, especialmente considerando que el hardware, aunque más económico que el personal, sigue representando un costo.

Una arquitectura de software eficaz facilita enormemente la operación del sistema, y su eficiencia se ve potenciada mediante el uso de herramientas especializadas. Herramientas como Datadog[2], Clarity[3] y Sentry[4] son interesantes para proporcionar una visión clara de lo que está sucediendo dentro del sistema. Estas plataformas permiten monitorear el rendimiento en tiempo real, rastrear errores y comprender la interacción del usuario, ofreciendo una información que ayude a la toma de decisiones operativas. Además, sistemas de gestión de tickets como Jira[5] son interesantes para identificar y gestionar eficientemente los problemas reportados por los clientes. Estas herramientas no solo ayudan a diagnosticar y resolver problemas rápidamente, sino que también facilitan la comunicación entre los equipos de desarrollo y operaciones, mejorando la agilidad y eficiencia del proceso de mantenimiento y operación del sistema. En conjunto, estas herramientas transforman la arquitectura de un sistema en una entidad más dinámica y receptiva, ajustándose continuamente a las necesidades operativas y mejorando la experiencia general del usuario.

1.2.4: Mantenimiento

De todos los elementos que conforman un sistema de software, el mantenimiento suele ser el más costoso[6]. Esta realidad se manifiesta en la constante adición de nuevas funcionalidades y en la inevitable secuencia de defectos y correcciones, los cuales consumen una cantidad significativa de recursos humanos. El costo principal del mantenimiento radica en la exploración y el riesgo. La exploración implica el desafío de investigar dentro del software existente para determinar el lugar y la estrategia más adecuados para añadir una nueva característica o reparar un defecto. Al realizar estos cambios, siempre existe la probabilidad de generar defectos involuntarios, incrementando así el costo asociado al riesgo.

Una arquitectura bien planificada puede mitigar enormemente estos costos. Al dividir el sistema en componentes y aislar estos componentes a través de interfaces estables, es posible clarificar las rutas para futuras funcionalidades y reducir significativamente el riesgo de fallos inadvertidos. Por lo tanto, el mantenimiento no es solo una fase más del ciclo de vida del software, sino un aspecto crítico que debe considerarse desde el inicio de la arquitectura de cualquier sistema de software.

1.2.5: Mantener opciones abiertas

Una arquitectura de software eficaz se caracteriza por mantener abiertas todas las opciones posibles durante el mayor tiempo posible. Este enfoque se alinea con el principio de "Last Responsible Moment" del movimiento Lean[7], que aboga por retrasar las decisiones hasta el momento más oportuno, cuando se dispone de la máxima información posible, enfatizando la importancia de la flexibilidad en el diseño del software para adaptarse a cambios futuros.

En la práctica, esto significa que algunas decisiones, como la elección de un framework específico, deben ser abordadas con cautela, ya que son difíciles de cambiar una vez implementadas. Por ejemplo, migrar un proyecto de React a Vue puede resultar complejo. Sin embargo, una arquitectura bien diseñada facilita la modificación de ciertas decisiones en el futuro.

Tomemos, por ejemplo, un sistema de análisis: la elección entre Google Analytics y Heap.io podría abstraerse dentro de una interfaz analytics. Esto permite un cambio en el proveedor (implementación concreta) en el futuro sin mayores complicaciones. Este enfoque de diseño mantiene abiertas las opciones y permite una mayor agilidad y adaptabilidad.

Al priorizar la política (reglas y procedimientos del negocio) sobre los detalles técnicos (como dispositivos de IO, bases de datos, servidores y protocolos de comunicación), la arquitectura del software se vuelve más resistente a los cambios en estos últimos. La clave está en reconocer que los detalles son facilitadores necesarios, pero no deben influir ni limitar la política central del sistema. De esta manera, se pueden posponer las decisiones sobre detalles específicos, como la persistencia de datos o el proveedor de autenticación, hasta que sea absolutamente necesario hacerlas, maximizando así la flexibilidad y la capacidad de respuesta del software a las necesidades cambiantes.

Un buen arquitecto maximiza el número de decisiones no tomadas, manteniendo la arquitectura abierta a cambios y adaptaciones futuras. Esto no solo facilita el desarrollo inicial, sino que también garantiza una mayor longevidad y relevancia del software en un entorno tecnológico en constante evolución.

1.3: Resumen

En este primer capítulo hemos visto una definición de *arquitectura de software* y hemos reflexionado sobre los aspectos fundamentales de una buena arquitectura. En general, podemos decir que una arquitectura ayuda a reducir el esfuerzo cognitivo que soportan los programadores al trabajar en un proyecto, y una de sus características fundamentales es la organización de las dependencias. Las arquitecturas organizan las dependencias de un proyecto definiendo interfaces que permiten abstraer detalles de implementación.

También hemos visto otros aspectos importantes de una buena arquitectura, como la facilidad de desarrollo, despliegue, operación y mantenimiento de un sistema de software, así como la capacidad de mantener todas las opciones abiertas tanto tiempo como sea posible.

2: Medir para mejorar

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.1: Dependencias

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.2: Acoplamiento y Cohesión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.2.1: Acoplamiento

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.2.2: Cohesión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3: *Conacimiento*

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.1: *Conacimiento* estático

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.1.1: *Conacimiento* de nombre

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.1.2: *Conacimiento* de tipo

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.1.3: *Conacimiento* de significado

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.1.4: *Conacimiento* de posición

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.1.5: *Conacimiento* de algoritmo

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.2: *Conacimiento* dimámico

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.2.1: *Conacimiento* de ejecución

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.2.2: *Conacimiento* de tiempo

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.2.3: *Conacimiento* de valor

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.2.4: *Conacimiento* de identidad

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.3: Reglas del conacimiento

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.3.1: Regla de Fuerza (RoS)

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.3.2: Regla de Localidad (RoL)

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.3.3: Regla de Grado (RoD)

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.3.3.4: Regla de Estabilidad (RoS)

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.4: Diagramas de dependencias

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

2.5: Resumen

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

3: Arquitecturas limpias

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

3.1: La regla de la dependencia

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

3.2: Entidades

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

3.3: Reglas de negocio

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

3.3.1: ¿Existe lógica de negocio en el frontend?

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

3.4: Detalles

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

3.5: Resumen

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4: Paradigmas de programación

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.1: Un poco de historia

4.2: Programación imperativa clásica

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.3: Programación estructurada

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.3.1: Conclusión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.4: Programación funcional

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.4.1: Funciones como objetos de primera clase

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.4.2: Funciones puras y efectos secundarios

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.4.3: Inmutabilidad

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.5: Programación orientada a objetos

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.5.1: Clases y Objetos

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.5.2: Herencia

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.5.3: Encapsulación

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.5.4: Abstracción

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.6: Typescript, el lenguaje del *frontend*

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.6.1: Utilizar POJOS y funciones puras en lugar de clases

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.6.2: Aprovecharse de las funciones como objetos de primera clase

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.7: Resumen

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

4.7.1: Puntos clave

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5: Principios

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.1: Coherencia y cohesión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.2: Single Responsibility - SRP

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.2.1: ¿Qué es un módulo?

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.2.2: ¿Qué es una responsabilidad?

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.2.3: ¿Qué es un actor?

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.2.4: Conclusión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.3: Open-Closed - OCP

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.3.1: Conclusión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.4: Liskov Substitution - LSP

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.4.0.1: Conclusión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.5: Interface Segregation - ISP

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.6: Dependency Inversion - DIP

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.6.1: Cómo usar `FooComponent` con inyección de dependencias

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

5.7: Conclusión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6: Patrones

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.1: Patrón Observador

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.2: MVC & MVVM

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.2.1: MVC (Model-View-Controller)

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.2.2: MVVM (Model-View-ViewModel)

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.3: Patrón repositorio

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.3.1: Objetivos Principales:

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.4: Patrón singleton

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.5: Patrón CQRS

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.5.1: CQRS en el Contexto de React con Tanstack Query

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.5.2: Ejemplo: Aplicación de Tareas con CQRS usando Tanstack Query

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

6.6: Conclusiónes

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7: Tipos de arquitecturas frontend

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.1: Introducción

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.2: MPA

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.2.1: Características de las MPA

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.2.1.1: Enrutado y navegación

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.2.1.2: Persistencia de datos

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.2.1.2.1: Formularios

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.2.1.3: Autenticación

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.2.1.4: Vista

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.2.2: Conclusión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.3: AJAX

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.4: SPA

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.4.1: AngularJS

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.4.1.1: Enrutado y navegación

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.4.1.2: Modelo de Datos

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.4.1.3: Componentes y reactividad

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.4.1.4: Persistencia de datos

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.4.1.5: Conclusión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.4.2: React

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.5: HSPA / PESPA

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6: SSC

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6.1: ¿Qué son exactamente los Server Side Components?

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6.2: El problema de la complejidad cognitiva

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6.3: Un ejemplo que me hizo reflexionar

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6.4: El problema del testing y las herramientas

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6.5: Cuando los SSC sí funcionan

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6.6: El problema de las aplicaciones “reales”

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6.7: La cuestión del vendor lock-in

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6.8: ¿Es solo resistencia al cambio?

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6.9: Mi recomendación personal

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

7.6.10: ¿Qué nos depara el futuro?

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8: Gestión de Estado

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.1: ¿Qué es el Estado?

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.2: Tipos de variables de estado

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.2.1: Valores estáticos

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.2.1.1: Constantes de Dominio

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.2.1.2: Código fuente

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.2.2: Valores Dinámicos

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.2.2.1: Variables de sistema

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.2.2.2: Variables de servidor

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.2.2.3: Variables de usuario

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.3: ¿Por qué analizar el estado?

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.3.1: Conclusión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.4: Ejemplos de Gestión de Estado

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.4.1: Ejemplo 0: Variables de solo lectura

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.4.2: Ejemplo 1: Variables de lectura y escritura

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.4.3: Ejemplo 2: Flotando el estado

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.4.4: Ejemplo 3: Añadiendo persistencia en el navegador (localstorage)

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.4.5: Ejemplo 4: Persistiendo el estado en la URL.

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.4.5.1: Entonces, ¿debemos persistir todo el estado en la URL?

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.4.6: Ejemplo 5: Interactuando con el servidor.

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.4.6.1: Condiciones de carrera

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.4.7: Ejemplo 6: Añadiendo autenticación y autorización

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.5: Flotación de estado

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.5.1: Flujo de datos unidireccional

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.5.2: Flotando el estado

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.5.2.1: Estado reactivo global

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.6: Estado derivado

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

8.7: Conclusión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9: Arquitecturas recursivas

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.1: Introducción y principios

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.2: Estructura recursiva de componentes

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.2.1: Componentes globales

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.3: Organización de carpetas

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.3.1: Páginas

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4: Tipos de archivos en detalle

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.1: Core

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.1.1: Entidades: Interfaces y tipos

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.1.2: Constantes

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.1.3: Reglas de negocio: Servicios

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.2: Aplicación

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.2.1: Cliente

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.2.2: Repositorios

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.2.3: Queries y mutaciones

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.2.4: Controladores

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.2.5: Vistas y componentes

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.2.5.1: Componentes con estado

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

9.4.2.5.2: Componentes puros

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

10: Conclusión

Este contenido no está disponible en el libro de muestra. El libro puede comprarse en Leanpub en https://leanpub.com/frontend-architecture.

Notas

1 https://pusher.com/tutorials/clean-architecture-introduction/

2 https://www.datadoghq.com/

3 https://clarity.microsoft.com/

4 https://sentry.io/

5 https://www.atlassian.com/software/jira

6 https://www.oreilly.com/library/view/97-things-every/9780596805425/ch34.html

7 https://www.oreilly.com/library/view/software-architects-handbook/9781788624060/a844b94f-be9e-456d-8ef0-cd9b46b41c33.xhtml