***Conocimiento teórico***

1. ¿Qué es un patrón de diseño?
2. ¿Qué es Docker?
3. Por favor describa uno de los principios SOLID
4. ¿Qué es un servicio REST?
5. ¿Qué es Kanban?

***Respuesta***

**¿Qué es un patrón de diseño?**

Los patrones de diseño o design patterns, son una solución general, reutilizable y aplicable a diferentes problemas de [diseño de software](https://profile.es/que-hacemos/desarrollo-web/). Se trata de **plantillas que identifican problemas en el sistema y proporcionan soluciones apropiadas a problemas generales**a los que se han enfrentado los desarrolladores durante un largo periodo de tiempo, a través de prueba y error.

Estos patrones proporcionan un enfoque estructurado y reutilizable para resolver situaciones recurrentes en el desarrollo de software. Los patrones de diseño ayudan a los desarrolladores a comunicarse y compartir soluciones eficientes y efectivas que han demostrado ser exitosas en el pasado.

Cada patrón de diseño tiene un propósito específico y ofrece una solución a un problema de diseño común. Al utilizar un patrón de diseño, los desarrolladores pueden aprovechar la experiencia acumulada y evitar reinventar la rueda, lo que resulta en un software más eficiente, mantenible y escalable.

Es importante tener en cuenta que los patrones de diseño no son algoritmos o código listo para usar, sino más bien pautas y descripciones de soluciones de diseño. Los desarrolladores deben adaptar e implementar los patrones de diseño de acuerdo con los requisitos y características específicas de su proyecto de software.

Los patrones de diseño de software se basan en principios y prácticas de diseño que han evolucionado a lo largo del tiempo. A menudo, se describen utilizando terminología y notaciones específicas para facilitar su comprensión y aplicación.

**¿Por qué usar patrones de diseño?**

El gran crecimiento del sector de las tecnologías de la información ha hecho que las prácticas de desarrollo de software evolucionen. Antes se requería completar todo el software antes de realizar pruebas, lo que suponía encontrarse con problemas. Para ahorrar tiempo y evitar volver a la etapa de desarrollo una vez que este ha finalizado, se introdujo una práctica de prueba durante la fase de desarrollo. Esta práctica se usa para identificar condiciones de error y problemas en el código que pueden no ser evidentes en ese momento. En definitiva, los patrones de diseño te **ayudan a estar seguro de la validez de tu código, ya que son soluciones que funcionan y han sido probados por muchísimos desarrolladores** siendo menos propensos a errores.

**Tipos de patrones de diseño de software**

 Los patrones de diseño más utilizados se clasifican en tres categorías principales, cada patrón de diseño individual conforma un total de 23 patrones de diseño. **Las cuatro categorías principales son**:

* **Patrones creacionales**
* **Patrones estructurales**
* **Patrones de comportamiento**

**1. Patrones creacionales**

Estos patrones son utilizados para la creación de nuevos objetos que buscan incrementar la flexibilidad y reutilización del código existente:

-**Abstract Factory**: Este patrón se utiliza para crear familias de objetos que se relacionan sin necesidad de especificar clases.

-**Builder Patterns**: El patrón constructor se utiliza para crear diferentes objetos a partir de una serie de pasos reutilizando el código de construcción. Esto simplifica la necesidad de crear subclases para cada configuración posible de un objeto particular.

-**Factory Method:** Este patrón ofrece una interfaz para crear nuevos objetos dentro de una superclase permitiendo que las subclases cambien el tipo de objetos que van a crear.

-**Prototype:** Se utiliza para crear nuevos objetos a partir de objetos existentes. A través de la declaración de una interfaz (clonar) este patrón encarga a los propios objetos la acción de clonarse.

-**Singleton:** Se utiliza para restringir la creación de nuevas instancias de una clase a un objeto determinado.

**2. Patrones estructurales**

Los patrones estructurales buscan facilitar la acción de ensamblar objetos y clases dentro de clases estructurales más grandes sin perder la eficiencia y manteniendo la flexibilidad.

**Adapter:** Adaptador es un patrón que se utiliza para que objetos con interfaces incompatibles colaboren entre sí.

**Bridge:** El patrón Bridge resuelve un problema habitual en la herencia de clases dividiendo clases relacionadas en dos jerarquías diferentes: implementación y abstracción, para que estas puedan desarrollarse independientemente.

**Composite:** Solo se recomienda utilizar Composite cuando el modelo de código está creado a partir de un sistema ramificado en forma de árbol.

**Decorator:** Se utiliza para extender el comportamiento de un objeto añadiendo funcionalidades al mismo a través de objetos encapsuladores que presentan dichas funcionalidades.

**Facade:** Una fachada proporciona una interfaz simplificada para un subsistema complejo. Por ejemplo, este patrón de diseño permite integrar una aplicación con una biblioteca que tiene muchas funciones de las que solo se ocupan unas pocas.

**Flyweight:** Ayuda a reducir el tamaño de los objetos almacenando en su interior solo el estado intrínseco (información constante) del mismo y compartiendo el resto de la información (estado extrínseco) entre varios objetos similares.

**Proxy:** Este patrón de diseño se utiliza para crear objetos sustitutos que trabajan como una interfaz hacia cualquier objeto determinado.

**3. Patrones de comportamiento**

Los patrones de comportamiento buscan resolver la comunicación entre diferentes áreas

**Chain of responsibility:** A través de este patrón podremos evitar que la petición emitida por un emisor sea acoplada a un solo receptor permitiendo que más de un objeto pueda responder a dicha petición.

**Command:** Se utiliza cuando es necesario encapsular dentro de un objeto todos los parámetros que una acción requiere para ejecutarse.

**Interpreter:** Utilizando Interpreter podremos evaluar un lenguaje a través de una interfaz que indique el contexto en el cual se interpreta.

**Iterator:** Este patrón de comportamiento se utiliza cuando necesitamos iterar en colecciones o conjuntos de objetos sin la necesidad de intercambiar información relevante.

**Mediator**: Se utiliza cuando necesitamos controlar las comunicaciones directas entre objetos y disminuir sus dependencias caóticas.

**Memento:** Este patrón es capaz de almacenar y restaurar la información de un objeto.

**Observer:** A través de este patrón de comportamiento varios objetos interesados (suscriptores) en un objeto en particular (notificador) pueden recibir notificaciones de su comportamiento mientras estén suscriptos a sus notificaciones.

**State:** Se utiliza para modificar el comportamiento de una clase de objetos dependiendo del estado actual (comportamiento interno) de dichos objetos.

**Strategy:** Permite separar todos los algoritmos de una clase específica en nuevas clases separadas donde los objetos pueden intercambiarse.

**Template method:** Este patrón define el esqueleto de un algoritmo y permite a las subclases variar la implementación del comportamiento del mismo.

**Visitor:** A través de este patrón podremos introducir nuevos algoritmos sin modificar la estructura de objetos que se utilizarán para ejecutarlos.

## Conclusión

En este post hemos hecho una breve **introducción a los patrones de diseño más populares** que fueron definidos en el libro de la Gang of Four, Design Patterns. Existen muchos tipos de patrones de diseño de software que nos permiten crear nuestro código de manera más fácil y con estructuras de código que ya han sido probadas y con garantías de que funcionan. Siempre que trabajas con un código que ha sido creado por otra persona, **utilizar patrones de diseño nos ayudará a descubrir problemas y a mejorar el código**.

<https://profile.es/blog/patrones-de-diseno-de-software/>

**¿Qué es Docker?**

Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas [contenedores](https://aws.amazon.com/es/containers/) que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

**Como funciona**

Docker le proporciona una manera estándar de ejecutar su código. Docker es un sistema operativo para contenedores. De manera similar a cómo una máquina virtual virtualiza (elimina la necesidad de administrar directamente) el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos que puede utilizar para crear, iniciar o detener contenedores.

Los servicios de AWS como [AWS Fargate](https://aws.amazon.com/es/fargate/), [Amazon ECS](https://aws.amazon.com/es/ecs/), [Amazon EKS](https://aws.amazon.com/es/eks/) y [AWS Batch](https://aws.amazon.com/es/batch/) hacen que resulte sencillo ejecutar y administrar contenedores de Docker a escala.

**Por qué usar Docker**

Docker le permite entregar código con mayor rapidez, estandarizar las operaciones de las aplicaciones, transferir el código con facilidad y ahorrar dinero al mejorar el uso de recursos. Con Docker, obtiene un solo objeto que se puede ejecutar de manera fiable en cualquier lugar. La sintaxis sencilla y simple de Docker le aporta un control absoluto. La amplia adopción significa que existe un gran ecosistema de herramientas y aplicaciones listas para su uso que puede utilizar con Docker.

**ENVÍE MÁS SOFTWARE, MÁS RÁPIDO**

La frecuencia media de envío de software de los usuarios de Docker es siete veces superior a la de aquellos que no lo usan. Docker le permite entregar servicios aislados con la frecuencia necesaria.

**ESTANDARICE LAS OPERACIONES**

Las aplicaciones con contenedores facilitan la implementación, la identificación de problemas y el retorno a una fase anterior para remediarlos.

**TRANSFIERA DE MANERA SENCILLA**

Las aplicaciones basadas en Docker pueden transferirse a la perfección desde equipos de desarrollo locales a implementaciones de producción en AWS.

**AHORRE DINERO**

Los contenedores de Docker facilitan la ejecución de más código en cada servidor, mejorando su uso y ahorrándole dinero.

<https://aws.amazon.com/es/docker/>

**Por favor enliste uno de los principios SOLID**

Los principios SOLID

Los 5 principios SOLID de diseño de aplicaciones de software son:

**S** – Single Responsibility Principle (SRP)

**O** – Open/Closed Principle (OCP)

**L** – Liskov Substitution Principle (LSP)

**I** – Interface Segregation Principle (ISP)

**D** – Dependency Inversion Principle (DIP)

Entre los objetivos de tener en cuenta estos 5 principios a la hora de escribir código encontramos:

Crear un software eficaz: que cumpla con su cometido y que sea robusto y estable.

Escribir un código limpio y flexible ante los cambios: que se pueda modificar fácilmente según necesidad, que sea reutilizable y mantenible.

Permitir escalabilidad: que acepte ser ampliado con nuevas funcionalidades de manera ágil.

En definitiva, desarrollar un software de calidad.

En este sentido la aplicación de los principios SOLID está muy relacionada con la comprensión y el uso de patrones de diseño, que nos permitirán mantener una alta cohesión y, por tanto, un bajo acoplamiento de software.

¿Qué son la cohesión y el acoplamiento?

Son dos conceptos muy relevantes a la hora de diseñar y desarrollar software. Veamos en qué consisten.

**Acoplamiento**

El acoplamiento se refiere al grado de interdependencia que tienen dos unidades de software entre sí, entendiendo por unidades de software: clases, subtipos, métodos, módulos, funciones, bibliotecas, etc.

Si dos unidades de software son completamente independientes la una de la otra, decimos que están desacopladas.

**Cohesión**

La cohesión de software es el grado en que elementos diferentes de un sistema permanecen unidos para alcanzar un mejor resultado que si trabajaran por separado. Se refiere a la forma en que podemos agrupar diversas unidades de software para crear una unidad mayor.

1. Principio de Responsabilidad Única

“A class should have one, and only one, reason to change.”

La S del acrónimo del que hablamos hoy se refiere a Single Responsibility Principle (SRP). Según este principio “una clase debería tener una, y solo una, razón para cambiar”. Es esto, precisamente, “razón para cambiar”, lo que Robert C. Martin identifica como “responsabilidad”.

El principio de Responsabilidad Única es el más importante y fundamental de SOLID, muy sencillo de explicar, pero el más difícil de seguir en la práctica.

El propio Bob resume cómo hacerlo: “Gather together the things that change for the same reasons. Separate those things that change for different reasons”, es decir: “Reúne las cosas que cambian por las mismas razones. Separa aquellas que cambian por razones diferentes”.

2. Principio de Abierto/Cerrado

“You should be able to extend a classes behavior, without modifying it.”

El segundo principio de SOLID lo formuló Bertrand Meyer en 1988 en su libro “Object Oriented Software Construction” y dice: “Deberías ser capaz de extender el comportamiento de una clase, sin modificarla”. En otras palabras: las clases que usas deberían estar abiertas para poder extenderse y cerradas para modificarse.

En su blog Robert C. Martin defendió este principio que a priori puede parecer una paradoja. Es importante tener en cuenta el Open/Closed Principle (OCP) a la hora de desarrollar clases, librerías o frameworks.

3. Principio de Sustitución de Liskov

“Derived classes must be substitutable for their base classes.”

La L de SOLID alude al apellido de quien lo creó, Barbara Liskov, y dice que “las clases derivadas deben poder sustituirse por sus clases base”.

Esto significa que los objetos deben poder ser reemplazados por instancias de sus subtipos sin alterar el correcto funcionamiento del sistema o lo que es lo mismo: si en un programa utilizamos cierta clase, deberíamos poder usar cualquiera de sus subclases sin interferir en la funcionalidad del programa.

Según Robert C. Martin incumplir el Liskov Substitution Principle (LSP) implica violar también el principio de Abierto/Cerrado.

4. Principio de Segregación de la Interfaz

“Make fine grained interfaces that are client specific.”

En el cuarto principio de SOLID, el tío Bob sugiere: “Haz interfaces que sean específicas para un tipo de cliente”, es decir, para una finalidad concreta.

En este sentido, según el Interface Segregation Principle (ISP), es preferible contar con muchas interfaces que definan pocos métodos que tener una interface forzada a implementar muchos métodos a los que no dará uso.

5. Principio de Inversión de Dependencias

“Depend on abstractions, not on concretions.”

Llegamos al último principio: “Depende de abstracciones, no de clases concretas”.

Así, Robert C. Martin recomienda:

Los módulos de alto nivel no deberían depender de módulos de bajo nivel. Ambos deberían depender de abstracciones.

Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones.

El objetivo del Dependency Inversion Principle (DIP) consiste en reducir las dependencias entre los módulos del código, es decir, alcanzar un bajo acoplamiento de las clases.

**https://profile.es/blog/principios-solid-desarrollo-software-calidad/**

**¿Qué es un servicio REST?**

## Qué significa REST

**REST es una interfaz para conectar varios sistemas basados en el protocolo HTTP** (uno de los protocolos más antiguos) y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON.

El formato más usado en la actualidad es el formato JSON, ya que es más ligero y legible en comparación al formato XML. Elegir uno será cuestión de la lógica y necesidades de cada proyecto.

**REST se apoya en HTTP**, los verbos que utiliza son exactamente los mismos, con ellos se puede hacer GET, POST, PUT y DELETE. De aquí surge una alternativa a SOAP.

Cuando hablamos de SOAP hablamos de una arquitectura divididas por niveles que se utilizaba para hacer un servicio, es más complejo de montar como de gestionar y solo trabajaba con XML.

Ahora bien, **REST llega a solucionar esa complejidad que añadía SOAP, haciendo mucho más fácil el desarrollo de una API REST**, en este caso de un servicio en el cual nosotros vamos a almacenar nuestra lógica de negocio y vamos servir los datos con una serie de recursos URL y una serie de datos que nosotros los limitaremos, es decir, será nuestro BACKEND nuestra lógica pura de negocios que nosotros vamos a utilizar.

**Por qué debemos utilizar REST**

REST no es solo una moda, y es por las siguientes razones que esta interfaz está teniendo tanto protagonismo en los últimos años:

* Crea una petición HTTP que contiene toda la información necesaria, es decir, **un REQUEST a un servidor tiene toda la información necesaria y solo espera una RESPONSE**, ósea una respuesta en concreto.
* **Se apoya sobre un protocolo que es el que se utiliza para las páginas web, que es HTTP**, es un protocolo que existe hace muchos años y que ya está consolidado, no se tiene que inventar ni realizar cosas nuevas.
* Se apoya en los métodos básicos de HTTP, como son:
  + **Post**: Para crear recursos nuevos.
  + **Get**: Para obtener un listado o un recurso en concreto.
  + **Put**: Para modificar.
  + **Patch**: Para modificar un recurso que no es un recurso de un dato, por ejemplo.
  + **Delete**: Para borrar un recurso, un dato por ejemplo de nuestra base de datos.
* Todos**los objetos se manipulan mediante URI, por ejemplo**, si tenemos un recurso usuario y queremos acceder a un usuario en concreto nuestra URI seria /user/identificadordelobjeto, con eso ya tendríamos un servicio USER preparado para obtener la información de un usuario, dado un ID.

## Ventajas de REST

* Nos **permite separar el cliente del servidor**. Esto quiere decir que nuestro servidor se puede desarrollar en Node y Express, y nuestra API REST con Ver por ejemplo, no tiene por qué estar todos dentro de un mismo.
* En la actualidad tiene una **gran comunidad como proyecto** en [GitHub](https://developer.github.com/v3/).
* Podemos crear un**diseño de un microservicio orientado a un dominio** (DDD)
* Es totalmente independiente de la plataforma, así que **podemos hacer uso de REST tanto en Windows, Linux, Mac** o el sistema operativo que nosotros queramos.
* Podemos hacer nuestra **API pública**, permitiendo darnos visibilidad si la hacemos pública.
* Nos da **escalabilidad**, porque tenemos la separación de conceptos de CLIENTE y SERVIDOR, por tanto, podemos dedicarnos exclusivamente a la parte del servidor.

## Quién usa REST

Muchas empresas como Twitter, Facebook, Google, Netflix, LinkedIn y miles de startups y empresas usan REST. **Todas estas empresas y servicios tienen su API REST por un lado con su lógica de negocio y por otro lado su parte frontend**, con lo cual nos permite centrarnos también un poco más en lo que es nuestra lógica de negocio haciendo una API REST potente.

**Estas API pueden ser públicas y lo pueden consumir otros usuario**s, con lo cual tenemos una forma de dar visibilidad a nuestra API y de testearla, no podemos olvidar que la parte de TEST es una de la parte más importantes, pues ¿por qué no nos serviría como TEST que otros usuarios prueben nuestra API REST y nos den Feedback?, pues aquí lo tenemos solo hay que segurizarla y es lo que ocurre con estas empresas.

**https://openwebinars.net/blog/que-es-rest-conoce-su-potencia/**

## ¿Qué es Kanban?

Kanban es una forma de ayudar a los equipos a encontrar un equilibrio entre el trabajo que necesitan hacer y la disponibilidad de cada miembro del equipo. La metodología Kanban se basa en una filosofía centrada en la mejora continua, donde las tareas se “extraen” de una lista de acciones pendientes en un flujo de trabajo constante.

La metodología Kanban se implementa por medio de tableros Kanban. Se trata de un método visual de gestión de proyectos que permite a los equipos visualizar sus flujos de trabajo y la carga de trabajo. En un tablero Kanban, el trabajo se muestra en un proyecto en forma de tablero organizado por columnas. Tradicionalmente, cada columna representa una etapa del trabajo. El tablero Kanban más básico puede presentar columnas como Trabajo pendiente, En progreso y Terminado. Las tareas individuales —representadas por tarjetas visuales en el tablero— avanzan a través de las diferentes columnas hasta que estén finalizadas.

**Cómo funciona la metodología Kanban**

Hoy en día, los tableros Kanban son en su mayoría tableros virtuales con columnas que representan las etapas del trabajo (¡aunque aun se pueden dibujar tableros Kanban en pizarras blancas y dar seguimiento al trabajo con post-it!). En un tablero, una “tarjeta Kanban” representa una tarea, y esta tarjeta de tarea avanza a través de las etapas del trabajo a medida que se finaliza. Los equipos que usan un sistema Kanban tienden a colaborar en un único tablero Kanban, aunque las tareas generalmente se asignan a miembros individuales del equipo.

**Los principios Kanban**

Existen cuatro principios básicos que te ayudarán a guiar a tu equipo al momento de implementar la metodología Kanban:

**1. Empieza con lo que haces ahora**

Puedes implementar el marco Kanban a cualquier proceso o flujo de trabajo actual. A diferencia de algunos procesos de gestión ágil más definidos como Scrum, el marco Kanban es lo suficientemente flexible como para adaptarse a las prácticas centrales de tu equipo de trabajo.

**2. Comprométete a buscar e implementar cambios progresivos y evolutivos**

Los grandes cambios pueden ser perjudiciales para tu equipo y, si intentas cambiar todo a la vez, es posible que el nuevo sistema no funcione como esperabas. Ese es el motivo por el cual el marco Kanban está diseñado para fomentar la mejora continua y el cambio progresivo. En lugar de cambiar todo de una vez, empieza por buscar cambios progresivos para lograr que los procesos de tu equipo realmente evolucionen con el tiempo.

**3. Respeta los procesos, los roles y las responsabilidades actuales**

A diferencia de otras metodologías Lean, Kanban no tiene roles integrados y puede funcionar con la estructura y los procesos actuales de tu equipo. Además, tu proceso actual podría tener elementos excelentes, que se perderían si intentaras modificar completamente tu sistema de trabajo de un momento a otro.

**4. Impulsa el liderazgo en todos los niveles**

Con el espíritu de mejora continua, el método Kanban reconoce que el cambio puede provenir de cualquier dirección y no solo “de arriba abajo”. Con Kanban, se alienta a los miembros del equipo a participar, proponer nuevas formas para lograr que los procesos evolucionen y emprender nuevas iniciativas de trabajo.

**Las 6 prácticas de la metodología Kanban**

Los principios básicos de Kanban te ayudan a guiar la mentalidad de tu equipo al momento de abordar un flujo de trabajo Kanban. Para implementar un proceso Kanban, sigue estas seis prácticas que usan las grandes empresas y que te permitirán ayudar a tu equipo a implementar una mentalidad de mejora continua y a lograr un crecimiento progresivo: los principios esenciales del marco Kanban.

**1. Visualizar el trabajo**

Una de las principales ventajas de Kanban es que puedes visualizar cómo el trabajo “avanza” a través de las etapas. Una tarjeta Kanban de tarea comenzará su viaje en el lado izquierdo de tu tablero y, a medida que tu equipo trabaja en ella, recorrerá lentamente las siguientes etapas hasta que aterrice en la columna Finalizadas. Esta práctica no solo te brinda una idea general de cómo el trabajo avanza a través de las etapas, sino que también te permite obtener información en tiempo real y apreciar de un vistazo el estado de los proyectos.

**2. Limitar el trabajo en curso**

Como metodología ágil, Kanban se centra en un principio de entrega temprana, lo que implica que las tareas deben moverse rápidamente de una columna a otra en lugar de estancarse en un estado ambiguo de “trabajo en progreso” (wip). Aunque no existe un requisito establecido sobre cuántas tareas deben estar “en progreso” en un momento dado, es importante establecer los límites del trabajo; por lo que anima a tu equipo a centrarse en finalizar tareas individuales y a evitar realizar varias tareas a la vez.

**3. Gestionar el flujo de trabajo**

La práctica n.° 2 recomienda limitar la cantidad de trabajo en curso, y la mejor manera de hacerlo es con la optimización del flujo de tareas dentro del tablero Kanban. Gestionar y mejorar el flujo de trabajo te permitirá controlar el tiempo predestinado para el trabajo y así poder reducir el tiempo de entrega (el tiempo que pasa entre el inicio de una tarea hasta que llega a la columna Finalizadas de tu tablero Kanban) y garantizar que estás entregando tareas o enviando nuevos productos mientras siguen siendo relevantes.

**4. Implementar políticas de procesos explícitas**

Debido a la rapidez con la que se mueven las tareas en Kanban, asegúrate de que tu equipo haya establecido y comunicado claramente las convenciones. Las políticas de tu proceso deben guiar a tu equipo en la implementación de la metodología Kanban. Además, se debe alentar a todos en el equipo a participar e innovar las políticas Kanban, tal como se establece en el cuarto principio básico de Kanban: Impulsar el liderazgo en todos los niveles.

**5. Implementar ciclos de comentarios**

En Kanban, necesitas recopilar comentarios de dos grupos distintos: tus clientes y tu equipo.

Recopila comentarios de tus clientes sobre la calidad y eficacia de la solución que produjo el equipo. ¿Fue el producto adecuado? ¿Hubo algún problema? En el caso de que haya surgido algún problema, como errores en un código o cualquier otro defecto del producto, revisa tu flujo Kanban y agrega más tiempo para la revisión, los ajustes y la evaluación.

Realiza consultas frecuentes con el equipo sobre el proceso de ejecución de un marco Kanban. ¿Cómo se sienten con los resultados? Aquí tienes otra oportunidad para fomentar el liderazgo en todos los niveles y mejorar las políticas de procesos del equipo.

**6. Mejorar colaborando y evolucionar experimentando**

En esencia, Kanban se trata de una mejora continua. Sin embargo, también significa que otros sistemas podrían funcionar bien junto con Kanban. Ya sea Scrum o alguna otra metodología, debes estar siempre dispuesto a colaborar, experimentar y desarrollar tus procesos si es necesario.

**Cómo implementar la metodología Kanban**

De acuerdo con el primer principio básico de Kanban (Empieza con lo que haces ahora), puedes aplicar Kanban a cualquier flujo de trabajo. Puedes crear un tablero Kanban con una pizarra y algunos post-it o con una hoja de cálculo y un formato dado, pero la mejor manera de visualizar un tablero Kanban es con una herramienta de gestión del trabajo como Asana. En Asana, cada proyecto se puede visualizar de cuatro formas diferentes, incluida una vista de tableros tipo Kanban.

Independientemente de cómo estés creando tu tablero Kanban, sigue estos sencillos pasos para aplicar Kanban a cualquiera de tus procesos existentes:

**1. Empieza con un tablero en blanco**

El tablero Kanban eventualmente contendrá todos los elementos de trabajo, incluido el trabajo finalizado. Sin embargo, por el momento se recomienda empezar con un tablero en blanco. Si usas una herramienta de gestión del trabajo como Asana, asegúrate de estar en la vista de Tablero.

**2. Crea columnas para representar el trabajo**

[Vista de Tablero] Tablero Kanban de un sprint de ingeniería para Scrum en Asana

Tradicionalmente, las columnas del tablero Kanban representan las distintas etapas del trabajo. Aunque las columnas que crees dependerán de tu equipo, te mostramos a continuación los tipos de columnas más comunes:

Trabajo pendiente, Bandeja de entrada o Nuevo: Esta es la columna donde se agregará el trabajo nuevo antes de ser asignado a un miembro del equipo.

Listo o Prioritario: Mueve el trabajo a esta columna cuando esté listo para empezar a trabajar.

En progreso: Se trata de las tareas en las que se está trabajando actualmente. Dependiendo de las necesidades del equipo, puedes desglosar la columna ‘En progreso’ en varias columnas más. Por ejemplo, un equipo de contenido puede crear columnas para Redacción, Revisión y Edición; en tanto un equipo de ingeniería puede crear columnas para Desarrollo, Pruebas e Implementación.

En espera: Mueve la tarea a esta columna si el trabajo presenta algunos cuellos de botella por algún motivo.

Finalizadas o Trabajo terminado: Las tareas deberían aparecer en esta columna una vez que hayan sido finalizadas.

**3. Agrega tareas para representar el trabajo**

[Tarea] Tarea en un tablero Kanban de Asana

En un tablero Kanban, cada tarea está representada por una tarjeta. Asegúrate de que los títulos de las tareas sean concretos. Recomendamos empezar con un verbo para que tu equipo sepa exactamente qué debe hacer.

Si usas una solución virtual de gestión del trabajo, también puedes agregar información adicional, contexto y archivos a las tarjetas Kanban. Luego, usa las etiquetas para dar seguimiento a los metadatos, como la duración o la priorización de la tarea.

**4. Haz que el trabajo avance por las etapas**

Un elemento central de la gestión del flujo de trabajo con los tableros Kanban es hacer avanzar el trabajo por las diferentes etapas. Puedes hacerlo manualmente arrastrando y soltando tareas, o puedes buscar una solución virtual de gestión del trabajo que te permita automatizar este proceso. Por ejemplo, con Asana puedes establecer reglas para orientar automáticamente los elementos de trabajo a las diferentes columnas según la información relevante de la tarea. Por ejemplo, puedes establecer una regla para mover una tarea a la columna ‘En progreso’ una vez que se haya asignado a un miembro del equipo.

Automatiza el trabajo con Asana

**5. Crea nuevos tableros Kanban si es necesario**

En teoría, podrías usar el mismo tablero Kanban virtual de forma indefinida. Debido a que los tableros Kanban dan seguimiento al trabajo a través de un proceso continuo, no hay motivo alguno para dejar de usar tu tablero actual. Sin embargo, en algunos sistemas como Scrum, con frecuencia crearás tableros Kanban para cada sprint nuevo. Simplemente crea el tablero nuevo (o borra tu pizarra), agrega la lista de tareas pendientes y empieza una vez más a trabajar.

**https://asana.com/es/resources/what-is-kanban**