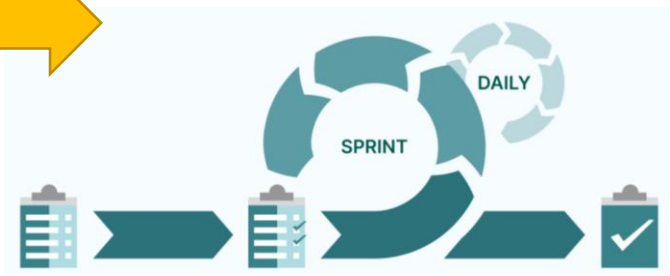
Explica cómo se implementa la metodología SCRUM

Es una metodología de trabajo que, a diferencia de los sistemas de trabajo que le precedieron -basados en el método en cascada-, se asemeja a los sistemas evolutivos, adaptativos y capaces de autocorregirse.

Método en cascada

1993

SCRUM



Implementación

1

Elige el responsable del producto. Este individuo tiene la visión de lo que se logrará por que es el encargado de decidir qué hacer, los pendientes el orden de los mismos y su contenido. Debe saber balancear la visión del producto, los recursos, los riesgos y las recompensas.

2

Selecciona tu equipo. Las personas con capacidades para convertir la visión en una realidad, es necesario buscar que un equipo sea trascendente, autónomo e interfuncional. Las personas en el mismo deben buscar siempre la mejora, ser capaz de autogestionar su trabajo y de usar sus ventajas mientras trabaja con las ventajas de los compañeros. Los equipos ideales son pequeños.

3

Elige un SCRUM Master. Es el especialista en la metodología y se encargará de capacitar al resto en la misma así como eliminar obstáculos. Al ser el experto en estrategia sabe que es importante ser razonable con las metas que sean desafiantes sin ser imposibles y deprimentes, que es importante enfocarse en una actividad y hacer las cosas bien a la primera.

4

Crea una bitácora del producto. La bitácora es una lista de alto nivel que contiene lo que debe hacerse para volver realidad la visión. La bitácora evolucionará de acuerdo a la avance del producto. Se debe considerar a todos los involucrados para asegurarse de que el producto es lo que se necesita y que se tiene lo necesario para llevarse acabo.

5

Afina y estima la bitácora del producto. Hacer estimaciones del esfuerzo que implicará la realización del producto. Cada elemento del equipo debe ver si hay recursos para llevar a cabo un elemento y acordar los parámetros para considerar un elemento terminado (entregable). Parte de los principios de SCRUM es que se adapta con el tiempo y las necesidades del proyecto. Es importante tener conciencia del avanzce del equipo para también motivar a mejorar.

Planeación del sprint. Es la primera de las reuniones y los que se reúnen son el SCRUM Master, el equipo y el responsable del producto para planear el sprint. Los *sprints* son periodos de tiempo de trabajo, tienen una extensión fija que generalmente es inferior al mes. En esta reunión se examina la bitácora y se pronostica cuánto se puede hacer en el *sprint* así como evaluar la velocidad de los *sprints* pasados, así mismo se consideran mejoras y metas para el siguiente *sprint*.

Vuelve visible el trabajo. Es importante llevar un registro visible del trqabajo realizado, un ejemplo simple es una tabla con tres columnas Pendiente , En proceso y terminado, así se puede observar cuánto se lleva y cuánto falta. Otra técnica es el diagrama de finalización que considera puntos por los avance y es el SCRUM master el que grafica estos avances. Idealmente también se puede observar una regresión de días o tareas que faltan para que termine el *sprint*. Esto además de hacer eficiente el trabajo ayuda a promover un ambiente en el que los integrantes del equipo se sientan motivados porque se saben importantes en el proyecto.

Parada diaria o scrum diario: Cada día a la misma hora y por no mas de 15 minutos el equipo y scrum master se reúnen y contestan las siguientes preguntas: ¿Qué hiciste ayer para ayudar al equipo a completar el sprint?, ¿Qué haras hoy para ayudar al equipo a completar el sprint? ¿Hay algún obstáculo que te impide a ti o al equipo cumplir la meta del sprint? Esta autoevaluación permite plantear metas para mejorar en el día.

Revisión del Sprint. El equipo muestra lo que hizo durante el sprint. Todos los interesados pueden asistir. Solo se demuestra lo que cumple con la definición de Terminado.

Retrospectiva del sprint: Evaluación del sprint. Se evalúa qué se puede mejorar o como pudo haber marchado mejor. Estas evaluaciones se concentran en explicar los problemas que se pudieron presentar más que culpar o juzgar. De esta manera se pueden plantar mejoras y compromisos los cuales se incluye en la bitácora y siguiente sprint.

11

Volver a iniciar con el Sprint

Explica en GitHub Branches Merge, Conflict

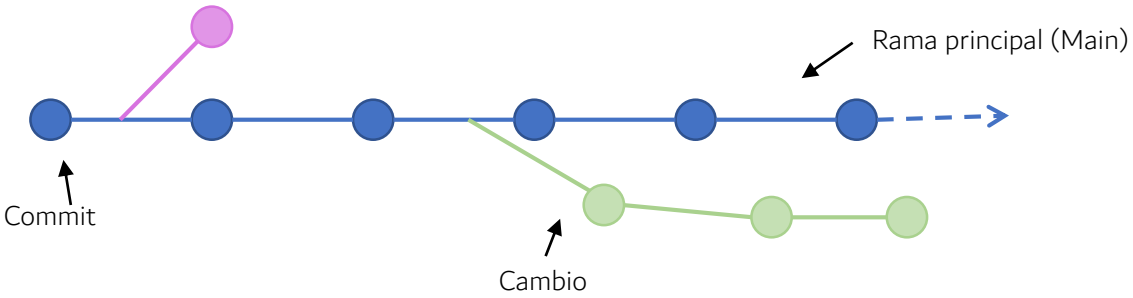
Git es un software controlador de versiones. Repositorio o base de datos que te permite tener un control de versiones.
El trabajo creativo, incluye crear archivos, modificarlos, guardarlos para después editarlos de nuevo y guardar esos cambios. Git ayuda en este último proceso. Dando información de **cuando** se hizo el cambio, **quién** lo hizo y **por qué** se hizo. ***Información disponible en el futuro***.

Graficamente así se vería el tracking:

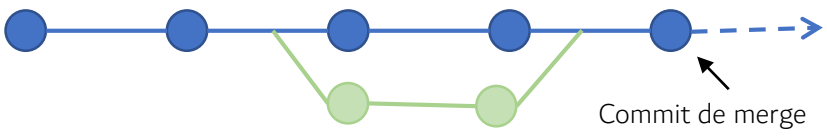
En un proyecto de una persona y de un archivo, no parece complicado, pero sí para un ***trabajo colaborativo***. Con git, puede haber registro de lo que cada uno hizo y por qué lo hizo. Para luego combinar el trabajo de todos: ***merge***.

El "Commit" es un punto en el tiempo. Una versión del trabajo.

Una implementación de alguna característica puede tener varias versiones también, para eso existen las branches/ramas, que son líneas del tiempo que agrupan un conjunto confirmaciones o "commits".



Para combinar todas las implementaciones que se le han hecho al trabajo se hace *merge* a las ramas sobe las que se haya trabajado y se combinan en 1



Resolución de conflictos

Cuando se quiere hacer *merge*, es posible que sea necesario resolver algunos conflictos que se generen. En las versiones, hablando de programación y código muchas veces el código de mi rama principal discrepará con el de la versión de la rama que pretende hacer *merge*, y por eso es necesario ir resolviendo esas discrepancias.

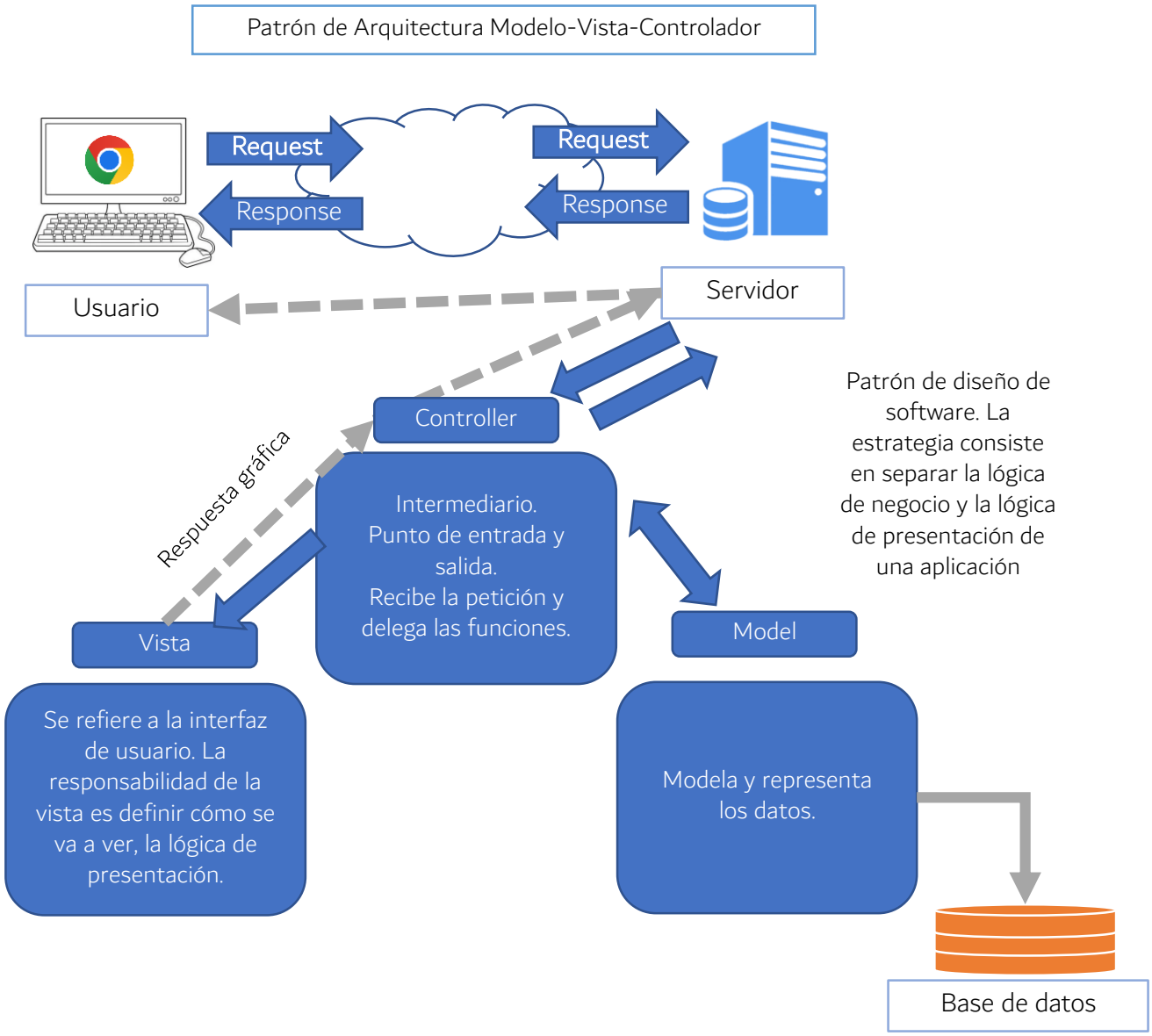
Para lo cual es necesario abrir los archivos e indicar los cambios que se desean dejar. Después de eso se debe hacer un commit.

Principales comandos

Iniciar el versionado `$git init`	Este comando agrega una carpeta e inicia un repositorio en la carpeta en la que el comando se ejecuta.
Ver el estado de los Archivos `$git branch <Nombre>`	Con este comando se puede indicar si ha habido algún cambio en los archivos o si están agregados al *stage área*
Agregar un archivo al repositorio `$git add <Nombre>`	Una vez que existen cambio es los archivos versionados deben agregarse al *stage área* listos para hacer commit.
Hacer un commit `$git commit -m "<Descripcion del cambio>"`	Cuando un cambio ha resultado exitoso es momento de marcar un punto en el tiempo con el commit. Se agrega un mensaje para que se sepa de qué se trató el cambiio-.
Eliminar archivos del repositorio `$git rm <Nombre archivo>`	En caso de que se quiera eliminar un archivo del repositorio se utiliza el comando y el nombre del archivo.
Historial de cambios `$git log`	Con este comando se puede ver el historial de los commits realizados.
Crear una Rama `$git branch <Nombre archivo>`	Un branch es una linea de desarrollo distinta de la principal.
Cambiar de Rama `$git checkout <Nueva Rama>`	Se cambia a la rama de la que se indique el nombre.
Mezclar `$git merge <Rama con los cambios>`	Se mezclan las ramas. La rama en la que se está es en la que se mezclara la rama .del nombre que se indique en el comando

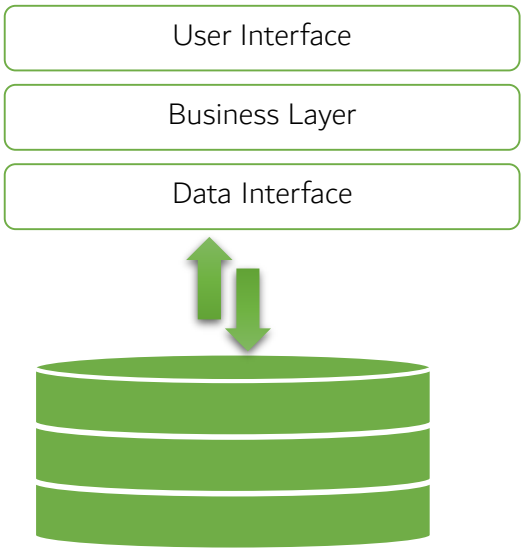
Explica y diagrama el modelo MVC

Patron de Arquitectura Modelo-Vista-Controlador



Explica la diferencia entre Microservicios y Arquitectura Monolitica

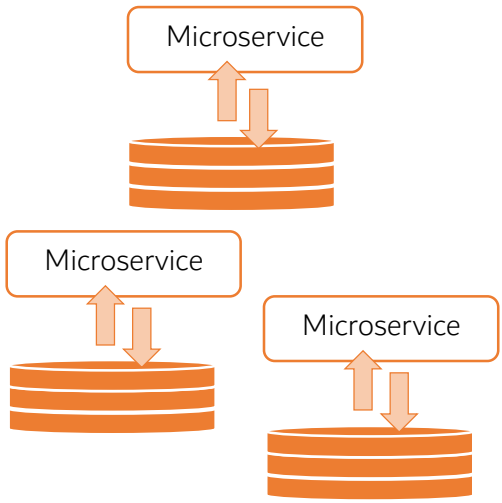
Arquitectura Monolítica



Es una aplicación que se desarrolla como una sola unidad; tiene una interfaz de usuario del lado del cliente (User Interface), una aplicación del lado del servidor (Business Layer) y una base de datos.

Todas las funciones se realizan en un mismo lugar por lo que es fácil de testear desplegar y mantener.

Arquitectura de Microservices

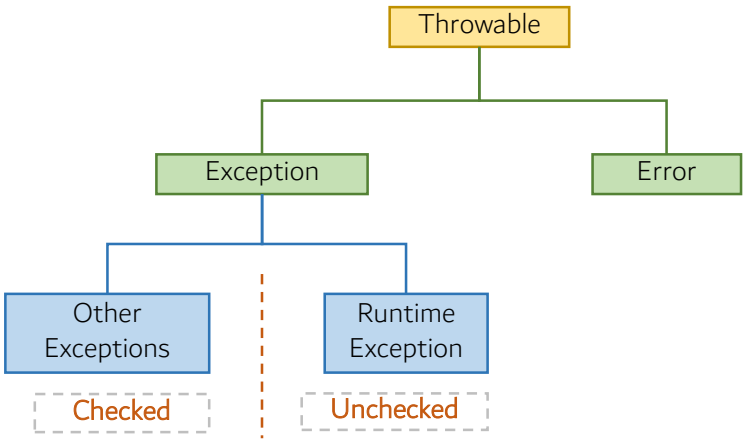


En la arquitectura de microservicios divide la aplicación en unidades independientes más pequeñas cada una de estas partes tiene su lógica independiente y su base de datos; se comunican entre sí a través de un API.

La independencia entre los componentes permite la actualización y el despliegue es más sencillo. Incluso un error que ocurra en un microservicio no afecta a los demás.

Explica los tipos de excepciones

No toda aplicación va a correr perfectamente y es necesario aprender a manejar los errores que puedan ocurrir en las aplicaciones. En Java el compilador puede ayudarnos con los errores de sintaxis que cometamos, pero hay errores que sólo serán vistos en el momento de la ejecución. Hay varios tipos de excepciones aunque tienen en común que extienden de la clase throwable.



Error: Es lanzado por la JVM a nivel interno vendrán de aquellos que programen la JVM y no podemos controlar ni manejar. Ejemplo; StackOverflowError.

Runtime Exception: El compilador no verificará ni avisará si aparecerá o no por lo que no obliga a tratarla. Se le conocen como Uncheked Exceptions

Other Exception: Pueden ser excepciones personalizadas y el compilador obligará a tratar la excepción. Se le conocen también como Checked Exceptions.

Para el tratamiento de excepciones se ocupa el try/catch:

```
try{
    //Código que puede lanzar una excepción
}
Catch (NombreExcepcion e) {
   //Lo que debe hacerse si se encuentra la excepcion
}
finally {
   //Lo que debe hacer en caso  de que se encuentre o no una excepción (Opcional)
}
```

Explica el multi-catch y try with resources

Multicatch

Dentro de la estructura de try/catch es posible atrapar varios tipos de excepciones

```
try{
    //Código que puede lanzar una excepción
}
Catch (NombreExcepcion1 | NombreExcepcion2 e) {
   //Lo que debe hacerse si se encuentra alguna de las excepciones
}
finally {
   //Lo que debe hacer en caso  de que se encuentre o no una excepción (Opcional)
}
```

Esto solo aplica si se desea el mismo tratamiento en caso de que se atrape cualquiera de los tipos de excepciones dentro del catch.

Explica el multi-catch y try with resources

Try With Resources

Dentro de la estructura try/catch se encuentra un tercer parámetro opcional que es el finally, este se usa generalmente cuando se necesita que se ejecute cierto código independientemente de si se atrapa o no una excepción. En Java 8 existe un tipo de try que realiza este tipo de funciones más fácilmente, sobre todo si se necesita que se haga algo al momento que un objeto deje de existir generalmente cerrar recursos, archivos o una conexión a base de datos. Este try con resources permite automatizar esta función:

```
try (Conexión con = new Conexion()) {
    // código que utiliza el recurso
} catch (Exception e) {
    // manejar excepción
}
```

En este ejemplo la nueva "conexión" es el recurso. Y se debe declarar y abrir dentro del paréntesis junto a try.

La segunda parte de la utilización de este try es que el recurso en este caso Conexión, debe implementar la interfaz Autoclosable. Esta interfaz describe el método close() que permitirá que el recurso se cierre automáticamente.

Menciona los tipos de Collections

En Java, las *Collections* son un grupo de clases e interfaces que se utilizan para almacenar y administrar conjuntos de datos u objetos.

List: Interfaz que presenta una colección ordenada de elementos: ArrayList y LinkedList
Set: Interfaz, representa colección de elementos únicos:HashSet y TreeSet.
Map: Interfaz colección de pares llave-valor: HasgMap y TreeMap
Queue: Interfaz colección de elementos ordenados según el orden de llegada: PriorityQueue y ArrayDeque
Deque: Interfaz colección de elementos ordenados según el orden de llegada y salida: ArrayDeque y LinkedList