NOMBRES:

DANIEL ENRIQUE

APELLIDOS:

FIGUEROA BRITO

SEMANA 2

EXAMEN SEMANA 2

ACADEMIA JAVA.

Índice

1. Scrum

Cómo implementar la metodología SCRUM.

Es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

1. El primer paso para la implementación de nuestra metodología scrum será definir a la **persona responsable del producto** esta persona será la encargada de darnos la visión de lo que realizaremos hasta de lo que podremos lograr siempre teniendo en cuenta los pros y contras del producto.
2. El siguiente en la metodología SCRUM será **definir el equipo de trabajo** este equipo es el que se encargará de hacer posible el desarrollo del producto este equipo debe contar con todas las habilidades necesarias para poder llevar a cabo con éxito la visión del responsable del producto
3. Lo siguiente será **escoger un scrum master** el cual tendrá la tarea de ayudar al equipo con el entendimiento de scrum y les dará soporte para eliminar todo lo que retrase al equipo.
4. **Se llevará a cabo una bitácora del producto** la cual consistirá en plasmar todas las actividades que se deberán realizar para hacer realidad la visión.
5. En este paso **se afinan detalles de la bitácora del producto** todo el equipo participará para contemplar el esfuerzo que se debe realizar y si es viable llevar a cabo la visión del producto.
6. Planeación del sprint en este paso todo el equipo se reúne con el scrum master para definir la duración del sprint, así como las actividades que se realizarán durante ese sprint.
7. El siguiente paso es hacer **visible el trabajo** que se realizará y para eso hay que crear una tabla con tres columnas pendiente, proceso. Terminada esto nos servirá para ver cómo se va avanzando en las actividades.
8. **Parada diaria o scrum diario** en esta etapa se lleva a cabo reuniones las cuales son a la misma hora todos los días debe contar con un tiempo estimado de 15 min y los puntos a ver son: qué hiciste ayer para ayudar al equipo a terminar el sprint, que harás mañana para ayudar al equipo a completar el sprint , y algún obstáculo no te permite completar el Sprint estas reuniones no deben durar más de 15 minutos.
9. En la siguiente se realiza la **revisión del sprint o demostración del sprint** la cual nos sirve para visualizar todas las actividades finalizadas durante el sprint en estas reuniones pueden asistir externos al team para ver los avances que se van teniendo por cada sprint.
10. **Retrospectiva del sprint** en esta etapa se lo ya concluido se le envía al cliente para tener una retroalimentación del sprint y el equipo analiza la realización de las actividades para poder hacer más eficiente el próximo sprint.

11. **Comienza de inmediato el ciclo del siguiente sprint**, tomando en cuenta la experiencia del equipo con los impedimentos y mejoras del proceso.

2. GIT

¿Qué es el comando brach?

El comando *git branch* te permite crear, enumerar y eliminar ramas, así como cambiar su nombre. No te permite cambiar entre ramas o volver a unir un historial bifurcado. Por este motivo, git branch está estrechamente integrado con los checkout y merge.

¿Qué es el comando merge?

El comando *git merge* permite tomar las líneas independientes de desarrollo creadas por *git branch* e integrarlas en una sola rama.

Cómo solucionar conflictos.

Existen 2 Tipos de conflictos de fusión

El primero es cuando se realizan cambios sobre la misma línea de desarrollo

que dos personas realizando cambios borre el archivo que está siendo modificado por la otra persona.

Formas de resolver estos conflictos.

La forma más directa de resolver un conflicto de fusión es editar el archivo conflictivo.

Cuando hayas editado el archivo, utiliza *git add* <*name*>para preparar el nuevo contenido fusionado. Para finalizar la fusión, crea una nueva confirmación ejecutando lo siguiente:

```
git commit -m "merged and resolved the conflict in Archive”
```

Git verá que se ha resuelto el conflicto y crea una nueva confirmación de fusión para finalizar la fusión

Herramientas que nos pueden servir para la solución de conflictos de fusión.

el comando *git status* el cual nos ayudará a identificar los archivos que están causando conflicto.

el comando *git log –merge* este comando pasándole el argumento merge creará un registro con una lista de las confirmaciones que tuvieron conflicto entre las ramas que se quieren fusionar.

El comando *git diff* este nos ayuda a encontrar diferencias entre repositorio/archivos este es muy útil para predecir y evitar conflictos de fusión.

Herramientas que no nos permiten iniciar una función

El comando *git checkout* puede servir para deshacer cambios y también puede ser utilizado para cambiar ramas.

El comando *git reset –mixed* puede usarse para deshacer cambios en el directorio de trabajo y en el entorno de ensayo.

Herramientas para cuando surgen conflictos de git durante una fusión

El comando *git merge –abort* si al comando merge le pasamos el argumento –abort se saldrá del proceso de fusión y devolverá la rama a su estado en el cual estaba antes de iniciar la fusión.

El comando *git reset* durante un conflicto de fusión para restablecer los archivos conflictivos a un estado que se sabe que es adecuado.

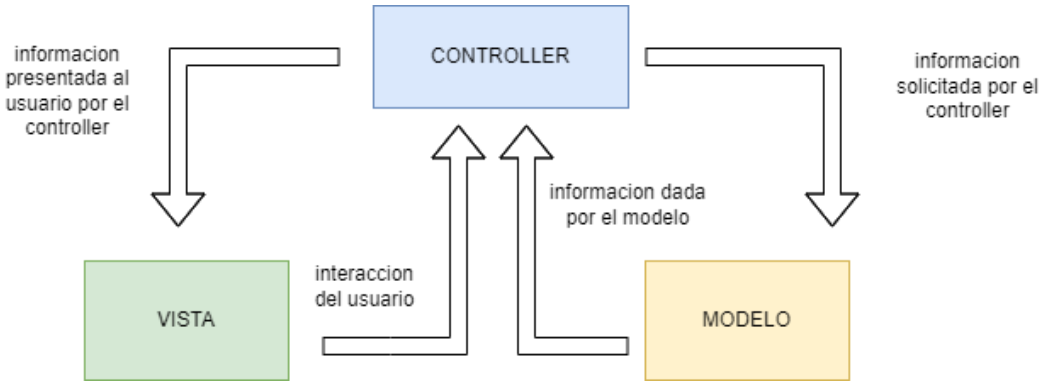
3. MVC

Explica y diagrama EL MVC (Modelo Vista Controlador).

El MVC es un patrón de diseño de software que se basa en que la aplicación esta seccionada en 3 partes la vista se encarga de presentarle la información al usuario, la vista es la parte de la aplicación con la que interactúa más el usuario.

El controlador es el que delega las responsabilidades al modelo, funge su función de orquestador, pero al mismo tiempo también hace de intermediario entre modelo y vista.

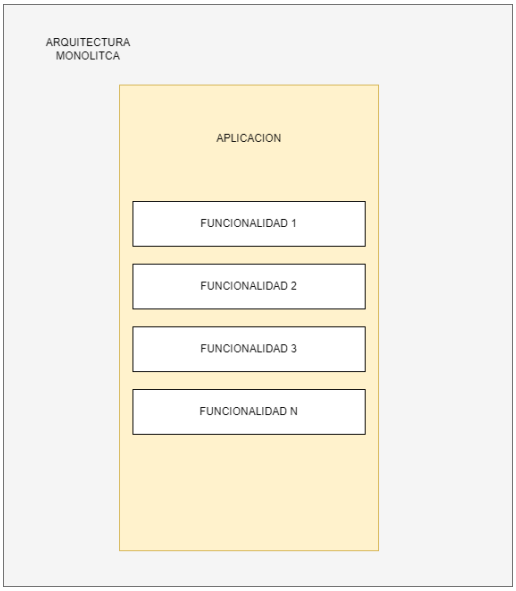
El modelo se encarga de realizar las funciones dadas por el controlador, llevando a cabo la lógica y representación de los datos que serán presentados por la vista.



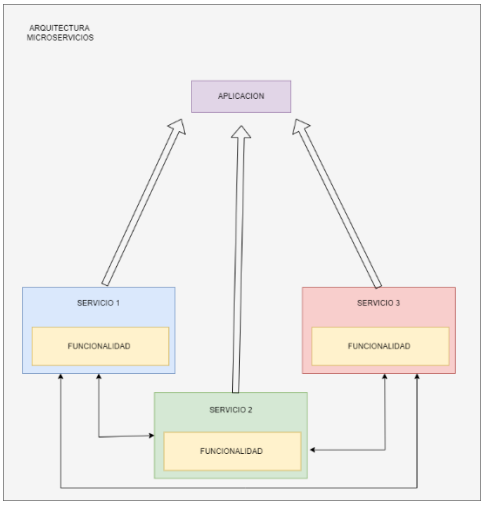
4. MICROSERVICIOS

Explica y diagrama las diferencias entre monolítica y microservicios

La principal diferencia entre una arquitectura monolítica y una arquitectura de microservicios es que en el caso de la monolítica todas las funcionalidades de una aplicación se encuentran unificadas en un solo código fuente y a su vez solo extrae información de una única fuente de datos.



La arquitectura basada en microservicios nos dice que las funciones de la aplicación están repartidas en tareas específicas esto quiere decir que cada sección procesa su propia información a pesar de que los microservicios están conectados entre sí, cada servicio se encarga de sus propias tareas.



5. EXCEPCIONES

Explica los tipos de excepciones.

Excepciones comprobadas.

Se conocen como excepciones en tiempo de compilación, ya que estas excepciones son comprobadas por el compilador durante el proceso de compilación para confirmar si el programador maneja la excepción o no. Si no es así, el sistema muestra un error de compilación.

Excepciones en tiempo de ejecución

Una excepción en tiempo de ejecución se produce simplemente porque el programador ha cometido un error. Has escrito el código, todo le parece bien al compilador y cuando vas a ejecutar el código, falla porque intentó acceder a un elemento que no existe o un error lógico hizo que se llamara a un método con un valor nulo.

Errores

Están fuera de nuestro alcance por lo tanto no podemos meterlos en tratamiento para corregirlos.

6. MULTICATCH Y TRYWITHRESOURCES

Explicar el multicatch y el throwable resource.

Multicatch

Es cuando dentro de un único catch puedes manejar varias excepciones a sumiendo que todas las excepciones que metamos en el catch tengan el mismo comportamiento.

Try-with-resource.

Esta instrucción sirve para declarar uno o más recursos, un recurso es un objeto que debe cerrarse una vez que ya halla sido utilizado en el programa el try-with-resource asegura que cada objeto que implemente la clase AutoCloseable se cierre de manera automática para no seguir ocupando espacio en memoria.

7. COLLECTIONS

Explicar de manera breve los collection.

List

Esta interfaz está dedicada a los datos de tipo lista en la que podemos almacenar toda la colección ordenada de los objetos en la cual podemos agregar o remover objetos y podemos identificar cada objeto por índice , ya que esta lista no acepta datos primitivos.

Queue

La interfaz de cola mantiene el orden FIFO (First In First Out) de forma similar a una línea de cola del mundo real. Esta interfaz se dedica a almacenar todos los elementos cuyo orden es importante.

Sets.

La interfaz set es una colección desordenada de objetos en la que no se pueden almacenar valores duplicados. Esta colección se utiliza cuando queremos evitar la duplicación de los objetos y deseamos almacenar sólo los objetos únicos.

Map

La interfaz map es una estructura de datos que soporta el par clave-valor para mapear los datos. Esta interfaz no soporta claves duplicadas porque una misma clave no puede tener múltiples mapeos, sin embargo permite valores duplicados en diferentes claves. Un mapa es útil si hay datos y deseamos realizar operaciones en base a la clave.