Escenario creación de ramas

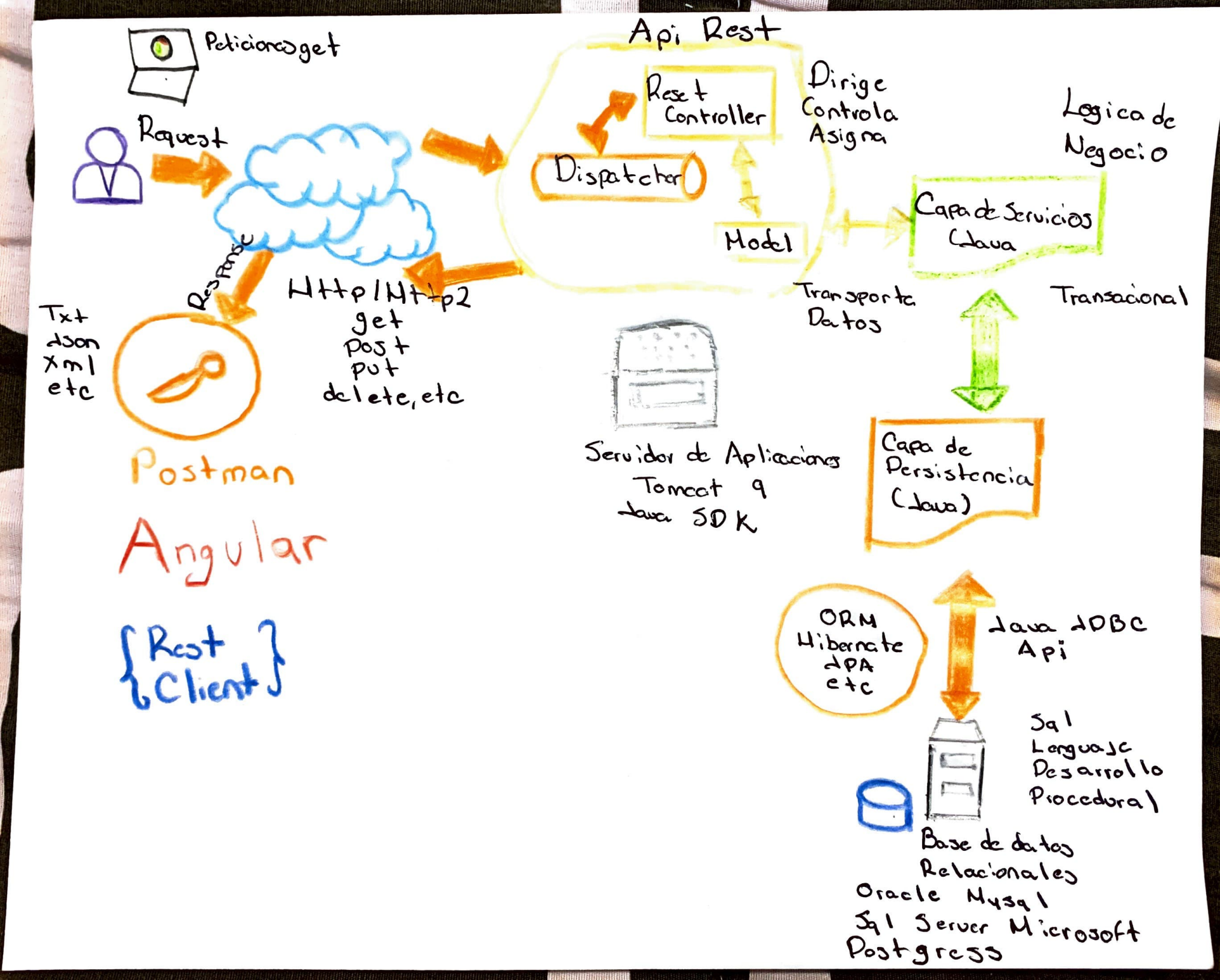
Se crea una carpeta llamada Pregunta Examen con el comando mkdir PreguntaExamen
Se ingresa a ella mediante el comando cd PreguntaExamen
Se iniciativa con el comando git init
Al inicializarse se crea la rama master que es donde nos encontramos
Se crea un archivo pregunta.txt con el comando touch pregunta.txt
Se agrega el texto "Pregunta examen en la rama master"
Se utiliza el comando git status para que nos muestre que tenemos un archivo modificado
Después se utiliza el comando git add .
Para poder agregar todas las modificaciones realizadas
Ahora mediante el comando git commit -m "Se agrega información en la rama master a pregunta.txt"
Git status para ver que todo se haya realizado de manera correcta
Ahora creamos otra rama con el comando git switch -c rama-a
Este comando creara la rama y nos moverá automáticamente a ella, ahora nos encontramos en rama-a que contiene el archivo txt Pregunta, ya que esta rama es una copia de la rama master
Se cambia el archivo txt con "Pregunta examen en la rama-a"
Se procede a utilizar un git add .
Y git commit -m "Se creo la rama-a y se modificó pregunta.txt"
git status, para asegurarnos que todo fue agregado con exito
Ahora con el comando git switch master
Nos movemos nuevamente a la rama master donde procederemos a crear una 2da rama, ahora llamada rama-b
Esta se creará con el comando git branch rama-b
Después nos moveremos a el con git switch rama-b
Al igual que la rama-a esta es una copia de rama master por lo que tiene un archivo pregunta.txt con "Pregunta examen en la rama master" como contenido, lo modificamos para que ahora sea "Pregunta examen en la rama-b"
y ahora procedemos a realizar un git add .
Y un git commit -m "Se creo la rama-b y se modificó pregunta.txt"
git status para asegurar que todo está correcto
Ahora tenemos 3 ramas:
Master
Rama-a
Rama-b
Con un archivo pregunta.txt que se modificó en cada caso en la misma línea referente a la rama donde se encuentra
Procedemos a desplazarnos a la rama master
Con git switch master
Ahora hacemos un git merge rama-a
Se crea un conflicto ya que se modificó el mismo archivo en la misma línea
El archivo txt contiene lo siguiente:

<<<<<<< Head
Ejercicio de examen rama master
=======
Ejercicio de examen en rama-a
>>>>>>> rama-a
Se modifica el archivo quedando
Ejercicio de examen en rama master y en rama-a
Se ejecuta un git add .
Y un git commit -m "Se unieron las ramas master y rama-a, y se corrigió el archivo pregunta.txt"
Git status para asegurarnos que se realizó de manera correcta
Ahora se realiza lo mismo con la rama-b
Realizamos un git merge rama-b
Se crea un conflicto ya que se modificó el mismo archivo en la misma línea
El archivo txt contiene lo siguiente:
<<<<<<< Head
Ejercicio de examen en rama master y en rama-a
=======
Ejercicio de examen en rama-b
>>>>>>> rama-b
Se modifica el archivo quedando
Ejercicio de examen en rama master, en rama-a y en rama-b
Se ejecuta un git add .
Y un git commit -m "Se unio la rama b, a la rama master que previamente se habia unido a la rama-a y se modificó el archivo pregunta.txt"
Git status para asegurarnos que se realizó de manera correcta
Y para finalizar realizamos un git log que nos permite ver todos los commit que se hicieron durante el ejercicio

Inyección de dependencias

La inyección de dependencias se refiere a evitar que un objeto se administre así mismo las referencias con las que trabaja, dando lugar a que otra administre esto por ella, esto lo podemos referir como que una clase da los objetos con los que trabajara otra clase.
Una forma de verlo es que todo debe estar realizado de forma abstracta, para que un objeto no dependa de el mismo, si no de una abstracción de el.
Existen 3 tipos de inyección de dependencias:
Por variable o campo
Por setter
Por constructos
Sus principales ventajas son:
Extensibilidad: Nos permite añadir fácilmente nuevas funcionalidades al código
Ataduras tardías: Es la habilidad de escoger cuáles componentes usar en tiempo de ejecución en lugar de en tiempo de compilación.

Desarrollo paralelo: Si nuestro código esta débilmente acoplado, es mucho más fácil para diferentes equipos, los equipos trabajan en código fuente que no afecta directamente a los demás.
Facilidad de mantenimiento: Cuando nuestros componentes son independientes, la funcionalidad está aislada, si necesitamos buscar errores en el código o ajustar alguna funcionalidad, se realiza de una forma mas sencilla.
Facilidad para probar: Las pruebas unitarias son un tema muy importante. Su propósito es probar aisladamente pequeñas partes del código. Cuando tenemos código débilmente acoplado, fácilmente podemos usar dobles de prueba o dependencias falsas para aislar fácilmente las partes del código que deseamos probar.


Peticiones get
Request
Response
Txt
dson
Xml
etc
Http/Http2
get
Post
put
delete, etc
Postman
Angular
{Rest Client}
Api Rest
Rest Controller
Dispatcher
Model
Dirige
Controla
Asigna
Logica de Negocio
Capa de Servicios (Java)
Transacional
Transporta Datos
Servidor de Aplicaciones
Tomcat 9
Java SDK
Capa de Persistencia (Java)
ORM
Hibernate
JPA
etc
Java JDBC Api
Sql
Lenguaje
Desarrollo
Procedural
Base de datos Relacionales
Oracle Mysql
Sql Server Microsoft
Postgress