|  |  |
| --- | --- |
| **Fecha:** | 18/02/2022 |
| **Nombre:** | Alberto Ovalle Balaguera |
| **Cédula:** | 3136555278 |
| **Calificación:** |  |

**Prueba Técnica: API Activos Fijos**

Los activos fijos de una empresa son los bienes materiales, son necesarios para el funcionamiento de la empresa y no se destinan para la venta, ejemplos de estos son: bienes inmuebles, maquinaria, material de oficina, etc.

Estos activos fijos tienen características comunes entre todas como son: nombre, descripción, tipo, serial, numero interno de inventario, peso, alto, ancho, largo, valor compra, fecha de compra, fecha de baja, estado actual (activo, dado de baja, en reparación, disponible, asignado), color.

Los activos se pueden asignar a un área o persona, si es a un área esta se encuentra en una ciudad (diferentes áreas se encuentran en diferentes ciudades).

Diseñar una API RESTFul[[1]](#footnote-1)que permite administrar los activos, esta debería permitir:

* Buscar por todos los activos.
* Buscar los activos por: tipo, fecha de compra, serial.
* Crear nuevos activos.
* Actualizar activos.

o Permite cambiar el serial interno y fecha de baja.

* Listar las áreas o personas.

# Tecnologías

Debe utilizar las siguientes tecnologías:

* Plataforma Java 11 o superior.
* Framework Spring Boot.

# Tecnologías extras

Puede utilizar alguna de las siguientes tecnologías:

* Servidor de aplicaciones Tomcat, Wildfly o Glassfish.
* Maven, Gradle u otro.

# Motor de base de datos

Es libre de utilizar cualquier motor de base de datos relacional, pero se recomienda MYSQL o Postgres. Si decide utilizar tecnologías NoSQL se requiere justificar la selección de motor y ventajas.

# Reglas

* El código debe estar versionado en GIT, siguiendo algún modelo de entrega. (se sugiere GIT, basado en un flujo de rama por característica).
* Se deben validar las fechas (nunca una fecha de baja debe ser superior a una fecha de compra).
* No debe haber excepciones sin capturar.
* Las respuestas HTTP de la API deben estar estandarizadas a:
  + 200 Por consultas exitosas o 400 Para reportar datos faltantes al crear o actualizar un activo. o 404 para búsquedas sin resultados.
  + 500 Para errores que pasen en la capa de backend. o Todas las respuestas deben contener una descripción o resumen de lo ocurrido.

# Entregables

* Repositorio con el código donde se puedan ver los commit realizados.
* Archivo de despliegue o ejecutable (WAR, EAR o JAR).
* Documento en formato PDF con los siguientes items:
  + Modelo de datos: puede ser un modelo de clases con las entidades o un modelo relacional.

o Arquitectura: diagrama de componentes e indicar las tecnologías utilizadas. Justificar por qué la aplicación fue desarrollada utilizando esas tecnologías.

o Arquitectura de despliegue en una nube.

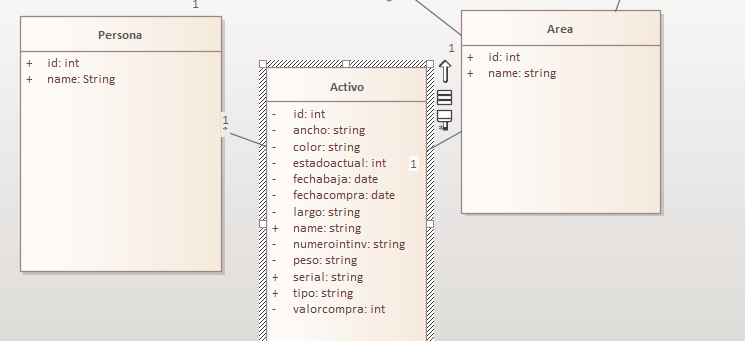
* + Instrucciones para la construcción y generación del archivo de despliegue o ejecutable.
  + Instrucciones para el despliegue y ejecución de la aplicación.
* Archivo de pruebas en POSTMAN.

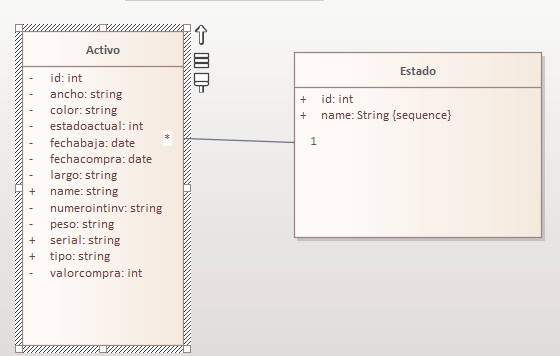
# Puntos adicionales

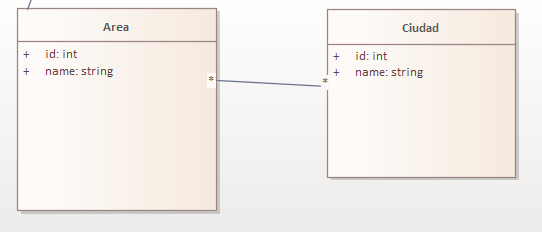
Los siguientes puntos no son requeridos en su solución, pero si los decide incluir serán tenidos en cuenta para mejorar el escalafón (Junior, Senior o Máster).

* Documentación de la API: esta puede ser realizada manualmente o utilizando una herramienta como RAML, Swagger u otra. • Documentación Java (JAVADOC)
* Geolocalización.
* Soporte de autenticación por token de alguno de los servicios REST.
* Demostrar el manejo de internacionalización (i18n).
* Demostrar el manejo de librería de Log (log4j, logback u otra).
* Pruebas Unitarias (JUnit, TestNG u otra).
* Frontend que consuma uno de los servicios utilizando alguna tecnología web como por ejemplo Angular, React u otro.
* Crear archivos Dockerfile para el proyecto.

Modelo de Datos







Arquitectura

Angular

Por que es la mejor manera de escribir código organizado y seguro sin bloque controlado de eventos se puede trabajar en un entorno ligero y eficiente.

Spring Boot

Facilita el empaquetado se puede trabajar con una arquitectura Microservicios

Que facilita

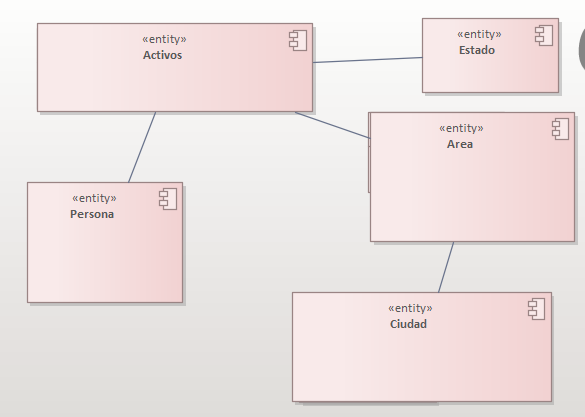
Modularidad: al tratarse de servicios autónomos, se pueden desarrollar y desplegar de forma independiente. también un error en un servicio no debería afectar la capacidad de otros servicios para seguir trabajando según lo previsto.

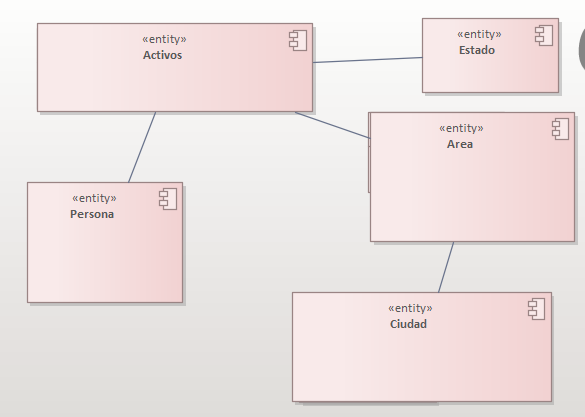
Mantenimiento sencillo y económico al poder hacerse mejoras de un solo módulo y no tener que intervenir en toda la estructura

Escalabilidad: como es una aplicación modular, se puede escalar horizontalmente cada parte según sea necesario, aumentando el escalado de los módulos que tengan un procesamiento más intensivo.

Mysql

Eficiencia de memoria, es rápida, es compatible con Linux, es gratuita

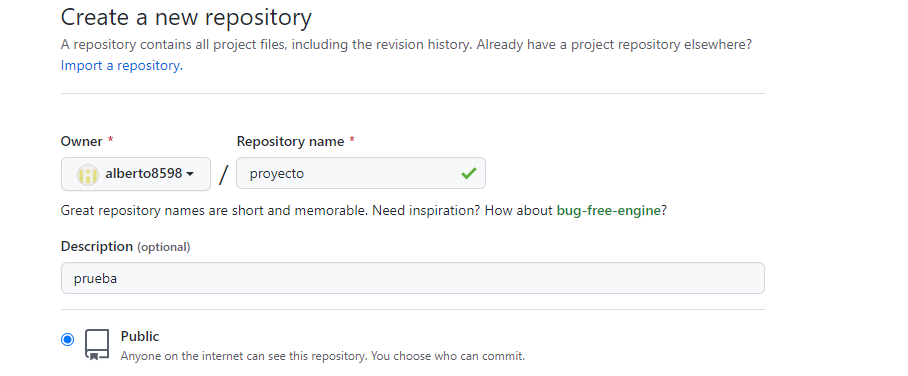


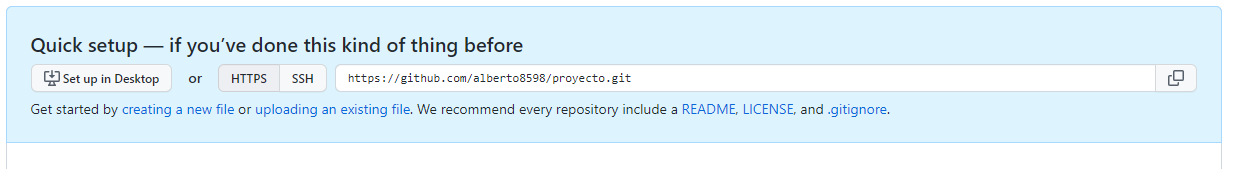


INSTRUCCIONES

1. Instalar Node js
2. Compilar el proyecto front directorio ejemplo1
3. Instalar Spring Boot
4. Intalar Eclipse
5. Ejecutar el proyecto back directorio ejemplo
6. Instalar Mysql
7. Copiar el directorio “ejemplo” base de datos en la carpeta xampp\mysql\data\
8. Ejecuta el front asi: http://localhost:4200/

GIT





https://github.com/alberto8598/proyecto.git

1. Servicios WEB REST

   [↑](#footnote-ref-1)