Grupo ASD

Prueba técnica desarrollador fullstack

“Prueba Practica JavaScript Interna.docx”

Miguel Angel Mendigaño Arismendy.

Java developer.

Duitama-Boyacá

20-11-2022

RESUMEN

Grupo ASD fue contratado para realizar el registro de todos los seres que tienen poderes de algún tipo del universo Marvel. Por tal motivo se debe tener el un registro de cada ser, un número de indicación única, conocer si pertenece a algún grupo de súper héroes o villanos, su lugar de operación principal (el nombre de la ciudad), su condición: si está en libertad, detenido o desconocido y el tipo de súper poder: si es Mental, Poderes Mágicos, Súper Fuerza… etc, algunos seres súper poderosos como Thanos tienen más de un tipo.

Además, se debe tener registro si tienen algún vehículo y el tipo de vehículo que utilizan.

Mas información ver: https://es.wikipedia.org/wiki/Actos_de_Registro_(c%C3%B3mic)

Su misión es modelar un API RESTFul1 que permita listar los mutantes, regístralos y además los vehículos, esta debería permitir buscar por todos los mutantes.

- Buscar los mutantes por: nombre y lugar de operación.
- Crear nuevos mutantes.
- Actualizar mutantes.
- Permite cambiar el nombre y el lugar de operación.
- Listar los vehículos.

Tecnologías Debe utilizar las siguientes tecnologías:

- NodeJS.
- Angular 8 o superior o ReactJS.

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 o Subversion, siguiendo algún modelo de entrega. (se sugiere GIT, basado en un flujo de rama por característica).

No debe haber excepciones sin capturar. • Las respuestas HTTP de la API deben estar estandarizadas a^4 :

- 200 Por consultas exitosas o 400 Para reportar datos faltantes al crear o actualizar un activo.
- 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.

1. Introducción

Se realizo el desarrollo de la aplicación ASD_APP_Mutant con el uso de NODE JS para el backend, soportando su estructura con el frameworks propio de express, se realiza la base de datos con My SQL.

Para el desarrollo del frontend se utilizó angular 8. Con la ayuda de estilos del framework de boostraps, también se utilizaron estilo de fontowesone, para la letra y algunos iconos.

Todo el proceso se realizó en línea trabajando en la plataforma de githug para el control de versiones con el uso de git, se manejó el concepto de Branch Flow.

1.2 Objetivos

El objetivo de este desarrollo es mostrar las habilidades que poseo en el desarrollo de la aplicación, cumpliendo los estándares de clean code, basado en principios SOLID, de acuerdo a los requerimientos de la solución abordada.

La documentación de la aplicación web basada en Node js y angular, se complementando y adjuntando en la carpeta de recursos del api, a la par que se fue desarrollando dicha aplicación, se adjuntaron los querys de construcción de la base de datos y los insert para tener información en ellas y realizar los test.

La idea es mostrar a pequeña escala todas las fases de un proyecto de desarrollo desde mi punto de vista como desarrollador.

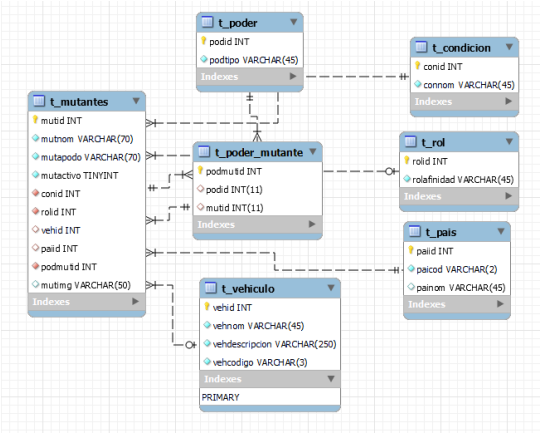
1.2.1 Objetivos específicos

Demostrar habilidades propias el desarrollo de aplicaciones web con el uso de lenguajes y herramientas Propias.

- Estructura de código limpio
- Manejo de control de versiones
- Entendimiento de programación orientada a objetos
- Conceptos de SOLID

- Uso de javascript
- Manejo de HTTP Verbs

1.3 Modelo de base de datos



Se creo un total de siete tablas con el uso de prefijos comunes adecuados al tipo de datos a manejar, esto en grandes volúmenes de datos facilita las operaciones entre ellos y con una distribución final de la siguiente manera:

T_pais: Contiene todos los atributos para la entidad país, en el cual nació, se encuentra u opera la entidad mutante.

Paiid: identificación de cada objeto

Painom: nombre del país

Paicod: indicativo de cada país

T_rol: Contiene todos los atributos para la entidad rol que ejerce la entidad mutante.

rolid: identificación de cada objeto

rolafinidad: define la tendencia del mutante a que bando tomar.

T_condicion: Contiene todos los atributos para la entidad condición, donde se define la situación presidiaria del mutante.

conid: identificación de cada objeto

connom: nombre de la condición en la que se encuentra

T_vehiculo: Contiene todos los atributos para la entidad vehículo, donde se definen los tipos de vehículos que usan los mutantes.

vehid: identificación de cada objeto.

vehnom: nombre del vehículo.

vehcod: código de identificación propio del manejo de siglas, comúnmente visto en el universo Marvel.

vehactivo:el estado en el que se encuentra

vehdescripcion: donde se especifican las características del vehículo en mención.

vehimg: se adjuntan los links con las imágenes correspondientes de la entidad.

T_vehiculo: Contiene todos los atributos para la entidad mutante, es la tabla donde se manejan el grueso de datos y relaciones ya que contiene cinco llaves foráneas, donde se definen los atributos de la entidad.

mutid: identificación de cada objeto.

mutnom: nombre del vehículo.

mutapodo: código de identificación propio del manejo de siglas, comúnmente visto en el universo Marvel.

mutactivo: el estado en el que se encuentra, por trazabilidad de datos no se deben borrar de la base de datos, solo se cambia su estado para definir si son visibles o no.

conid: llave foránea.

rolid: llave foránea.

vehid: llave foránea.

paiid: llave foránea.

podmutid: llave foránea.

mutimg: se adjuntan los links con las imágenes correspondientes de la entidad.

T_poder_mutante: Contiene todos los atributos muchos a uno de la entidad poder a mutante, donde un mutante

Puede tener varios poderes.

podmutid: identificación de cada objeto

podid: llave foránea

mutid: llave foránea

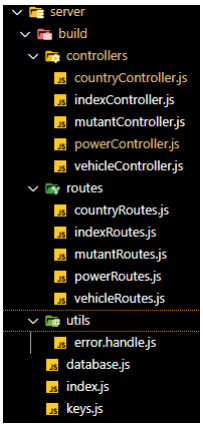
Todos los DLL para la creación de las tablas e insert de la base de datos, se encuentran en :

ASD_APP_MUTANT_Miguel_Angel_Mendiga-o\ASD_APP_api\MySql

1.4 Backend con Node JS

Basado en los conceptos de SOLID y la responsabilidad única, se manejó la estructura del

backend de acuerdo a entidades, donde ejecutan servicios para el llamado de datos a la base de datos por medio de un controller, de esta manera se encuentra organizado el backend.



La ejecución del backend se da por el comando npm run dev, que ejecuta el archivo de carga index, alojado en la carpeta build.

Para la creación del backend fue necesario la instalación de librerías de express, promises-mysql y nodemon, estos facilita la creación del api desde typescript, para posteriormente ser convertido a javascript para ser ejecutado desde este, también simplifica la creación del pool de conexiones de comunicación con la base de datos, la cual para pruebas se realizó en un repositorio local, por intermedio del localhost en el puerto 3306, se usan claves genéricas para root: admin.

En el controller encontraremos la interface de la petición con la base de datos, se usó query native, para cargar el parte del proceso de operación a la base de datos.

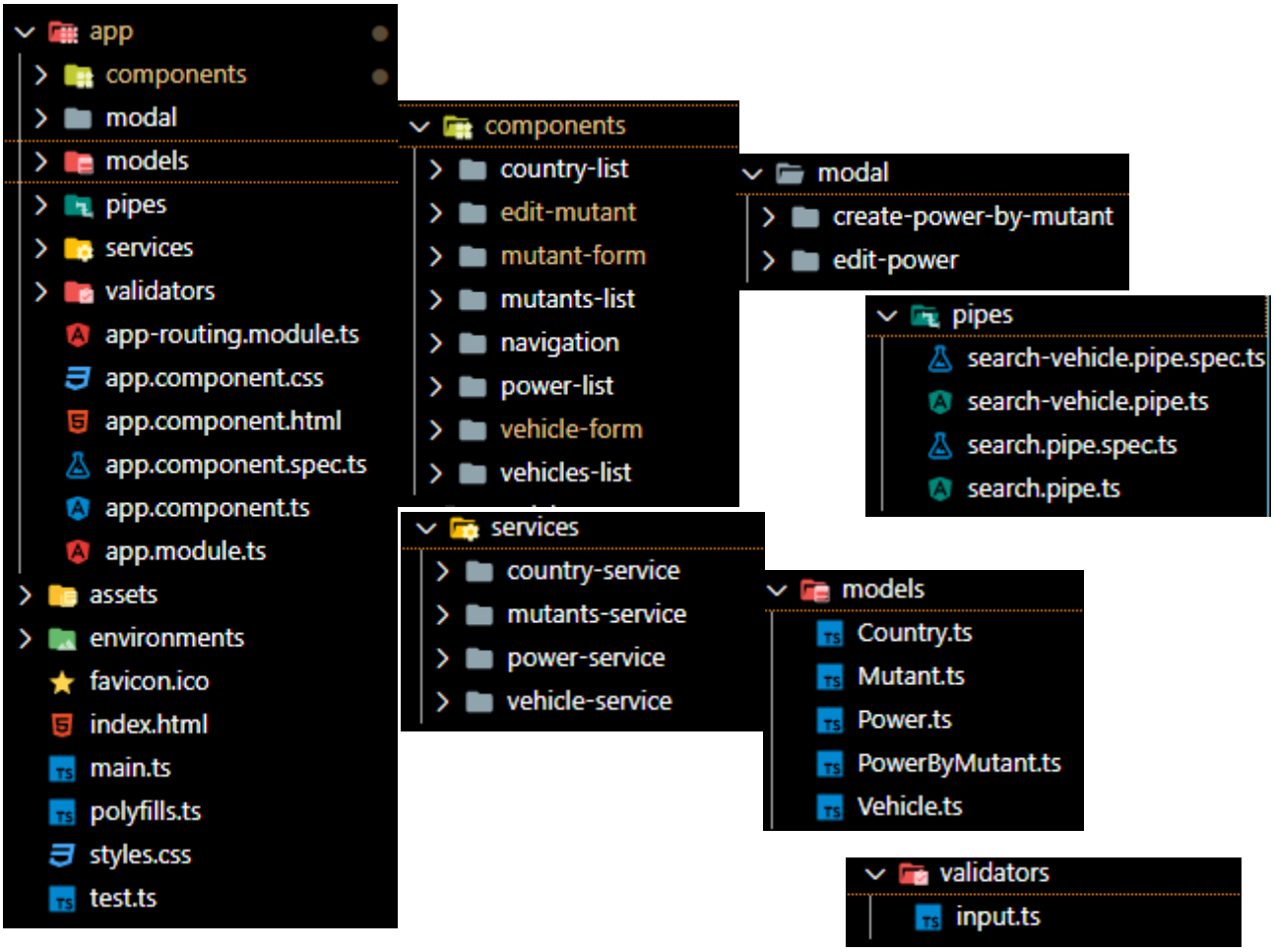
En la carpeta de routes se encontrarán los servicios que interactuaran con el front y en la carpeta útil se encuentra los archivos de control para la listas y error, desde donde se implementaran los try catch, los cuales en el momento solo están enviando mensajes, pero con la ayuda de este archivo se pueden realizar múltiples configuraciones de respuesta.

Fuera de las carpetas encontraremos el archivo base de datos que contiene el código de interacción del pool de conexiones con la base de datos, el archivo index que contiene los datos de arranque y expedición de rutas de servicios y el archivo keys que contiene las variables correspondientes a nuestra base de datos.

1.5 Frontend en Angular.

Para el desarrollo del frontend se utilizo angular con ayuda de Bootstrap para los estilos, basado en un sistema de módulos con componentes independientes por entidad y función, se implemento un aplicativo web de multipaginado.

La estructura del proyecto es definida en el siguiente árbol:



Components: Se encuentra el bloque de archivos js, html, Css/scss y espec para cada función de cada entidad.

Modals: contiene el bloque de archivos js, html, Css/scss y espec para lo modales utilizados.

Pipes: contiene las herramientas que nos permiten modificar o transformar la información presentada en pantalla, para este caso la búsqueda.

Services: contiene los servicios que interactúan con el backend.

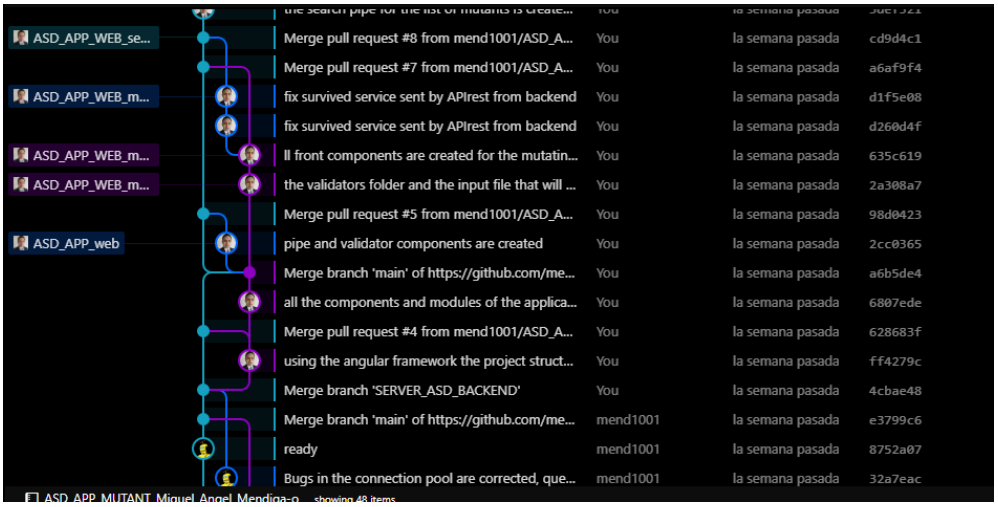
Validators: contiene el bloque de funciones para el manejo de datos en la interacción del cliente.

2. Control de versiones

El control de versiones se realizó desde un repositorio en github:

https://github.com/mend1001/ASD_APP_MUTANT_Miguel_Angel_Mendiga-o.git

Mediante el flujo de ramas, se realizo el control de versiones, para cada funcionalidad de la aplicación se realizó una rama nueva, las cuales se borraban una vez terminado el desarrollo de la misma, todos los commit se realizaron de acuerdo a las normas work flows, donde fueron específicos , detallados y en inglés, para facilitar el entendimiento a diferentes desarrolladores.



3. Despliegue de pruebas con postman.

Se crea una colección de pruebas postman con los diferentes servicios creados, la colección completa y los resultados se guardan en la carpeta de recursos en el API.

Se realiza un breve video demostrando la operación de la prueba.

https://youtu.be/Cp85nHMm0Xk

4. Despliegue.

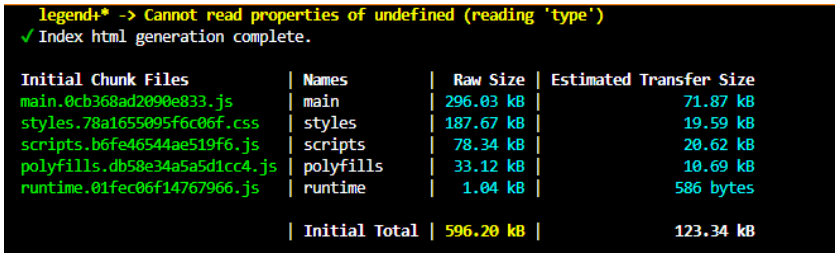
Una vez finaliza la etapa de desarrollo, se requieren ciertos paso y procedimientos para poner en producción la app, iniciando por tener operando la app, para este caso se subirá la base de datos a un repositorio en la nube, siguiendo los siguientes pasos.

- Para la etapa de desarrollo se manejo los datos de conexión con el archivo keys, que contiene los datos para el enlace con la base de datos del localhost, sin embargo en producción esto cambia, se debe instalar la dependencia dotenv, lo cual se realiza con “npm i dotenv” desde la consola o el ide puesto en la ubicación del proyecto, esto con el fin que la app pueda leer archivos .env.
- Se creo un repositorio en github independiente para el api: https://github.com/mend1001/API_APP_MUTANT_MIGUEL_MENDIGANO.git
- Se agrega en el archivo package.json la directiva "start":"node ./build" ya que en producción se usa start en lugar de dev.
- Para desplegar la aplicación utilizara la plataforma de “Railway” que se puede asociar directamente a github.

- Sobre la plataforma de “Railway” se crea una base de datos de my sql con los datos de las tablas generadas para la aplicación.
- Se puede acceder con el host de repositorio al backend con el siguiente enlace:

https://apiappmutantmiguelmendigano-production.up.railway.app/

- Se pueden realizar consultas agregando mutant, vehicle, power.
- Una vez desplegado el backend-end se puede ejecutar el front, cambiando el punto de enlace por el link.
- Una de las ventajas de usar angular es que nos proporciona un comando para compilar de manera sencilla, el “ng build --prod” o en caso de tener problemas por la versión se puede usar “ng build --configuration production”, esto tomara algunos minutos.
-



- La carpeta que genera, para este caso con el nombre de dist, se debe comprimir en zip y así se puede subir a la nube para ser compartida.

5. Mejoras.

Una vez entregada una solución funcional, se pasa a la etapa de refinamiento, donde se crean las siguientes tareas para mejoramiento del proyecto:

Base de datos

- usar secuencias propias para los id.
- Crear stored procedures para las consultas.

Backend

- Separar del controller la consulta query, creando un repository.
- Cambiar el uso de query native por el equivalente jpa en node js, puede ser con el uso node DBI o node jdbc, si la aplicación crece se puede hacer uso de stored procedures directamente en la base de datos.
- Enviar datos desde el backend para contenido pageable.
- Realizar try catch para consultas no creadas, evitando sabotajes con el uso de algún framework propio de node como express-promise-router.
- Usar una clase abstracta para todos los controller y router, ya que contienen métodos en común.

- Adicionar los test unitarios.

Frontend

- Uso de formularios reactivos.
- Implementación de validadores.
- Implementación de debounce para un tiempo de espera en el pipe de búsqueda después del evento onclick.
- Crear clase abstracta para cada entidad.
- Mejoras del aspecto visual en general, tales como scroll horizontal(carrusel), menú de navegación tipo hamburguesa, interfaces de visualización.