***Relevamiento de requerimientos***

***–***

***Evaluación técnica PWC***

## ***Detalle de la tecnología:***

Base de datos: SQL server

Back-end : .net core con entity framewok

Front\_end: React

## ***Requerimientos:***

a.- El estado actual del servicio para cada línea (ver API método /subtes/serviceAlerts)

b.- Dada una Línea, Estación y Destino de subte (ver API método /subtes/forecastGTFS) indiqué el tiempo estimado de arribo del próximo tren.

c.- Mostrar un histórico de Incidentes (proviene de las consultas del punto a) sucedidos para una línea en un rango de fechas.

## ***Resolución Técnica de requerimientos:***

Detalles de la resolución técnica:

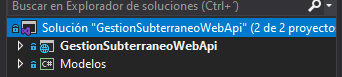
El sistema se divide en 3 partes:

La base de datos, el backend y el front end.

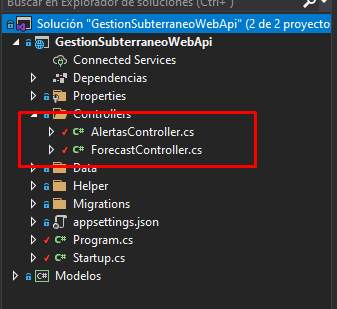
La base de datos realizada en sql server tiene la información de registros de usuarios que utilizan el sistema al momento de consultar incidentes.

El backend está dividido en dos soluciones:

1. Una librería de clases, donde tiene todos los modelos utilizados en el backend
2. Una api web que satisface a los tres requerimientos solicitados



La api web contiene dos controladores



Primer controlador: Controlador Dedicado para realizar consultas y peticiones a subtes/Alertas y utiliza entity framework para obtener los registros de los usuarios y inserta nuevos registros cuando se realiza la peticion

Segundo controlador:Controlador dedicado para realizar peticiones subtes/forecastGTFS, no utiliza entity framework porque no requiere de realizar peticiones o inserciones a la base de datos.

Tiene un método adicional que permite retornar un listado de todas las estaciones un línea pasada por el body de la api llamada GetEstacion.

El front-end es una aplicación web desarrollada con react y creado con npm, por lo cual antes de utilizarse debe ser instalada sus dependencias con: npm i

1. ***Primer Requerimiento.***

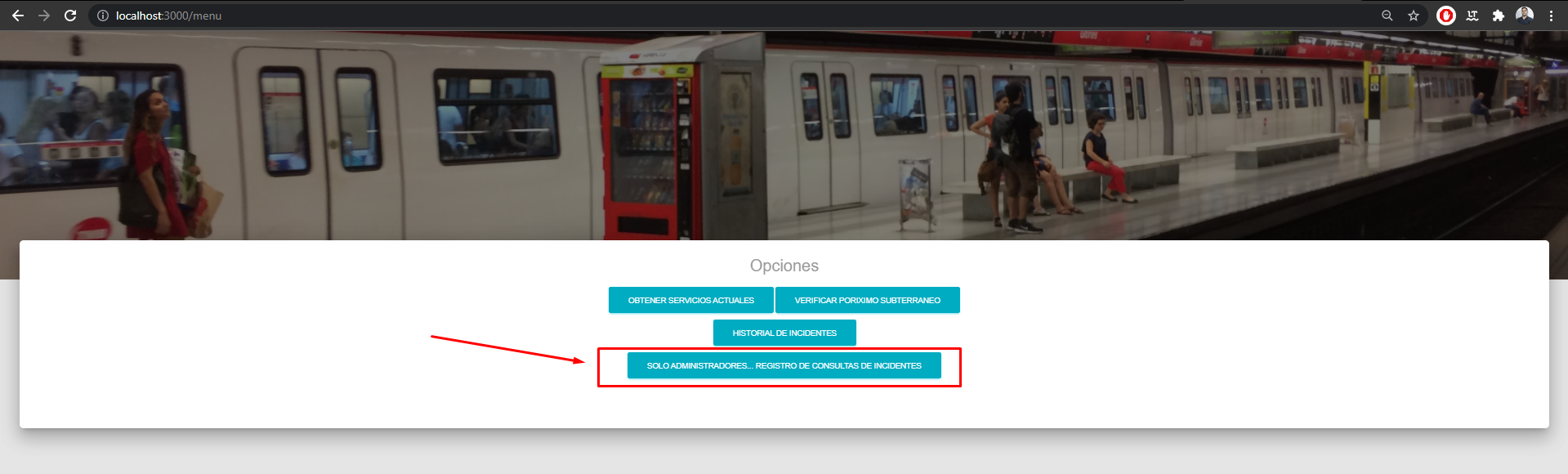
***C. Tercer Requerimiento.***

El primer requerimiento se encuentra en el controlador AlertasController, la spa realiza una petición GetEstadoRamales pasándole el nombre de usuario a cifrar y pasar por header como client\_id.

Este método realiza la petición a subtes/alertas y obtiene su response y lo guarda en un objeto de modelo ResponseAlerta, este objeto se mapea a un modelo abstraído, que es más resumido y proporciona información más simple para abastecer a la spa de react, al crear el modelo de abstracción llamado ResponseAlertaAbstraccion se guarda el registro del usuario en el modelo RegistroIncidenteDbContext que contiene del dbset de la tabla registro de la base de datos.

Al registrar la consulta de incídete del usuario tenemos un método que se llama GetRegistrosIncidentes que obtiene un listado de todas las consultas de usuarios y envía a la aplicación de react un listado de estos registros que se mapean y se muestran en una tabla de material ui.

Para entrar a este requerimiento en la ui es en esta opción:

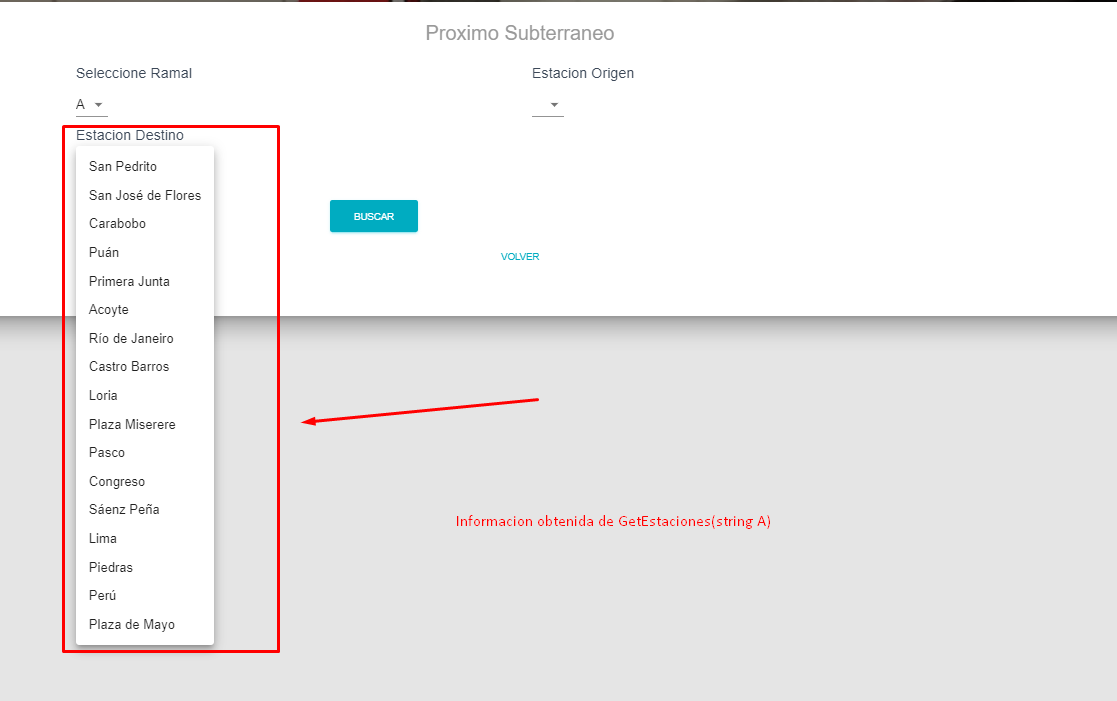


1. ***Segundo Requerimiento.***

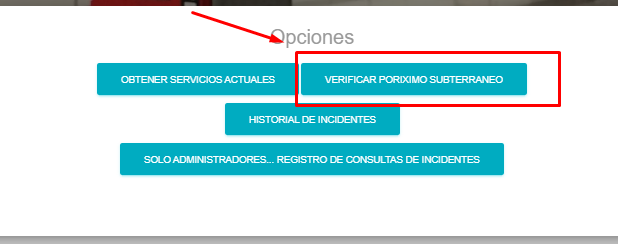
El Segundo requerimiento se encuentra en el controlador ForecastController. La spa Realiza la petición al método GetForecast que recibe como parámetro el nombre de usuario, una estación,una linea y una estacion de destino.

Al dicha petición el método se encarga de obtener la respuesta de la api subtes/forecastGTFS y obtiene su respuesta en el modelo ForecastResponse después de ya tener este objeto cargado se realiza la abstracción del modelo en el modelo ForcastResponseAbstraccion este objeto ya va a tener la información lista del próximo subterráneo a arribar a la estación origen.

Adicional del requerimiento: Para mayor comodidad del usuario Al seleccionar un ramal en el select de ramales de subterráneo, hacemos una consulta a GetEstaciones en el backend y le pasamos por parámetro la estación que eligió el usuario, y en este método se obtiene un listado de todas las estaciones de esa línea de subterráneo, entonces desde la spa de react mostramos el listado en un select, de esa manera no tendríamos problema de realizar una filtración de estaciones y no tendríamos que hacer validaciones de si la estación está correctamente escrita igual que en la api subtes/forecastGTFS



Para entrar a este requerimiento en la ui es en esta opción:



## ***Aspectos entregados:***

### · Fecha de compromiso

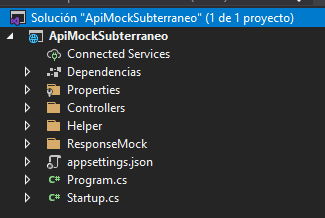
Fecha de compromiso de entrega día 14 de octubre.

### · ***Documentación de código***

* Documentación en el código por medio de Summarys
* Documentación en naming de variables
* Documentación carpeta
  + Colecciones de Postman
  + Diseños de baja fidelidad de la interfaz de usuario
  + Documentación de requerimientos(este archivo)
  + Request y response de apis subtes utilizado en la evaluación
  + Diseños en Uml
    - Diagrama de Casos de uso del usuario con el sistema
    - Diagrama de Casos de uso de la aplicación spa de react con las apis de Diagrama de Casos de uso
    - Diagrama de secuencia de la inserción a la base de datos por realizar una consulta de incidentes

### · ***Buenas prácticas***

* Aplicación de Principios Solid
* Buen naming en las variables, propiedades y métodos
* Llamados simples a apis
* La aplicación de react fue construida con definiciones de Atomic web design
* Uso de Hooks y customHooks(useEfect,useState,useContext)
* Pruebas unitarias con llamadas a apis mock, nombre de la solución ApiMockSubterraneo
* Manejo de llamadas asíncronas en .net core con Httpclien y con Axios en React.



### · ***Tecnologías Aplicada***

* Migraciones, se utilizó migraciones para interactuar con la base de datos de una manera más extensible y escalable para el futuro
* Se utilizó la librería HttpClient para realizar llamadas a apis (BackEnd)
* Se utilizó decorador para interactuar con diferentes métodos gets en apis
* Se utilizó entity framewok core
* la api Diseño de Casos de uso es un proyecto realizado con .net core 3.1
* se utilizó Axios en la spa para simplificar los llamados a las apis
* se utilizó la libreria material ui para crear componentes
* Seguridad de llamados a api, en la api GestionSubterraneoWebApi hacemos una validación de seguridad para que el proveedor siendo la aplicación web pueda comunicarse con la misma.

### · ***Extensibilidad y mantenibilidad del código***

* Migraciones: las migraciones hacen que el diagrama de la base de datos y la funcionalidad de la misma api sea mucho mas extensiva y escalable.
* En mantenibilidad del código, se defino que por api a consultar se delegue la responsabilidad en un controlador único de esa forma, respetando esa presmia el código va a tener mayor mantenibilidad y legibilidad.
* React router dom, en cuanto la aplicación web al utilizar react router dom para la navegación y esto lo hace mucho más escalable, ya que obliga que todas las rutas de las diferentes “páginas” se declaren en componente index de la aplicación web.

### · ***Premisas y assumptions***

Como premisas para realizar la evaluación técnica, fui desglosándolo en tareas más chicas, cree un kanban en notion y dejo en constancia la imagen del mismo.

