Diseño/Desarrollo

Para el desarrollo de las API's utilicé NODE js porque, a pesar de tener experiencia en C# y nunca haber usado node, lo vi más viable por el hecho de que en caso de hacerlo en C# hubiera utilzado .NET Core y al no tener experiencia en docker no sabría cómo correrlo correctamente. En cambio, NODE, estaba seguro que con una línea de comando era suficiente. Además, no pensé que fuera un problema ya que suelo manejar js.

Al empezar el challange me encargué de investigar, principalmente sobre docker y como poder correr el compose. Para posteriormente enteder el funcionamiento del escaner y la manera de obtener los resultados.

A nivel base datos, hice 4 tablas. Una que guarde el estado del escaneo y la url, otra con la descripción de las vulnerabilidades encontradas; cada una de ellas con sus respetivos ids. La tercer tabla es una intermedia entre la de escaneos y vulnerabilidades, que guarda la relación entre ellos mediante el id. La última tabla guarda el id de la tabla de escaneos y el id del escaneo realizado por el servicio ZAP.
La función de esta última tabla, es guardar un registro de los escaneos ya que, al realizarlos, es muy probable que la web no haya terminado de escanearse por completa. Esta tabla cumple la función de relación entre la tabla de la base con los datos de los servicios, permitiendo revisar el estado nuevamente para ser actualizado. En el momento que el escaneo pasa a "FINISHED" el registro es borrado de la tabla.

La lógica de la API se trata que constantemente, cada 10 segundos, se busque en la tabla de la relación entre los escaneos de la BD y el servicio algún registro. En el caso de haberlo, consultaría un servicio de estados para actualizarlo, también ingresa a servicios de reporte para encontrar nuevas vulnerabilidades mediante carga el escaneo web.
Agregando también, cada uno de los métodos, siguiendo lógicas similares: Consumición de servicios, verificación de que estén en la base de datos, y en caso de no estar, cargarlos en la misma.

Separé el desarrollo en módulos para una mejor comprensión:
Services: Contiene la lógica necesaria para consumir los servicios
db: Contiene la lógica y conexiones con la base de datos.
routes: Actuan como el controlador, toman los vistas llamadas y según cuál se elija se ejecuta el servicio correspondiente.
También están los archivos json environment y environment _prod, contienen los datos necesario para la conexión de la bd y del ZAP, tanto para manejo local y en docker respectivamente. Estos archivos son importados en la capa de servicios y la de datos. Con comentar y descomentar alguna de las líneas de importe, se puede cambiar de ambiente.