**Informe prueba técnica DT – Backend PHP**

**Análisis**

El proyecto Rick y Morty API está conformado de dos subproyectos, el frontend y el backend. En el front tenemos el consumo de recursos del api del back, el cuál sirve la información de personajes, lugares y episodios de la seria animada Rick And Morty. Con esta información proveniente del back se logra mostrar en varias interfaces y paginas la información de dichas entidades, así mismo se puede agregar, actualizar y eliminar nuevos o antiguos datos.

**Entendimiento**

Se logra captar la idea general de la prueba, la cual consiste en reemplazar el back de la aplicación hecha con nodejs, Express y sqlite, por un servidor hecho en laravel. Donde debemos servir la misma data recolectada del api remota de Rick & Morty, guardarla en nuestra propia base de datos y desplegarla en la misma url, junto con los recursos para poder agregar, editar y eliminar.

**Desarrollo**

Se empezó creando el proyecto de laravel en el que usaremos tres modelos Character, Location y Episode, estos modelos serás nuestras entidades a las cuales les creamos sus migraciones de estructura y seeders para alimentar la base de datos.

Los seeders los cargamos haciendo una petición GET, a los endpoints del api remota de R&M, de cada una de las entidades: characters, locations, episodes.

Teniendo la información en nuestra DB, procedemos a crear los métodos en cada controlador de cada modelo para poder obtener la información, guardar, actualizar y eliminar.

Con ayuda de estos métodos logramos crear los endpoints para poder consumir estos recursos en sus diferentes tipos como GET, POST, PUT, PATCH y DELETE.

**Complicaciones**

* La primera complicación fue encontrar todos los endpoints que el cliente o el front necesitaba para consumir toda la información y el formato de respuesta que cada endpoint debía servir. Después de una larga búsqueda y entendimiento del código en ambos proyectos base (client, server), se logra encontrar la información deseada.
* Siguiendo con el desarrollo, se tuvo que hacer una configuración de cors para poder acceder a los recursos del api hecha en laravel.
* Luego surgió el problema de guardar la información de ciertos campos de entidades como un formato json o array, esto con el objetivo de consumir la información proveniente desde el api remoto y guardarla como texto, pero servirla al cliente como un formato json. Se logra encontrar la solución mediante el ‘casting’ de campos en cada modelo.
* Por último, se tuvo que cambiar la lógica estándar del consumo de api en laravel. Normalmente cuando se quiere editar o eliminar un registro, se suele pasar en la url una variable dinámica (/characters/1 - /characters/2), para poder acceder directamente al registro del modelo en laravel, pero los endpoints se estaban pidiendo por medio de query params (/characters/?id=1), esto hizo que se tuviera que desarrollar la búsqueda del registro en el modelo manualmente y después servir la data en el formato deseado.

**Conclusiones**

Fue una prueba bastante interesante y retroalimentadora, donde me sirvió para afianzar mis conocimientos y aprender a la vez nuevas cosas. Agradezco por la oportunidad de presentarla.