Este documento lo hago con motivo de aligerar la carga de la persona que tenga interés en ver los documentos que he ido dejando de mi proceso de trabajo

Ante todo quiero remarcar algo. Mis máximas son la sinceridad y la honestidad. Lo que verás dentro de estas carpetas es el aprendizaje y trabajo que he desarrollado por mí mismo. Verás que hay referencias a páginas externas, conversaciones enteras de chatGPT, apuntes que yo mismo me he hecho y, en definitiva, todo el proceso de creación del proyecto ha sido documentado casi paso por paso si sigues los problemas que he ido afrontando y los diferentes archivos.

Esto no significa que yo en un futuro documentaré siempre así las cosas por el consumo de tiempo que lleva.

¿Y yo qué saco?

En primer lugar demuestro que el trabajo lo he hecho yo y no otra persona.

Puede ser que las extensivas conversaciones con la IA puedan dar, de entrada, una mala imagen, pero nada más lejos de realidad. Gracias a ese proceso he aprendido, he resuelto problemas de manera más eficaz y he materializado el qué en el cómo sin pasar horas mirando líneas y líneas de código o múltiples hojas de documentación, además de poder hacer una página con unos requisitos y características que de otra manera, con mi experiencia actual, ni en sueños la hago en 3 semanas.

Para facilitar la lectura de esos documentos, he marcado en azul mis consultas.

De todas formas, como leer todo es sencillamente mucho, en esta hoja voy a poner un resumen de los problemas que he tenido por el camino y como más o menos los he podido resolver, así como del proceso que he seguido.

BBDD:
Decidí montar mi BBDD en oracle, quizás no fue la mejor idea, pero quería aprender esta tecnología. Tuve algunos problemas como la creación de un usuario nuevo para tener mis tablas separadas y también tuve algunos problemas con las restricciones de las tablas.

En cuanto al diseño, hice un boceto yo mismo y le pedí a chatGPT revisión basando mi concepto en todo-ist. Este me sugirió añadir muchos campos que yo no había contemplado pero los que ya había sugerido eran buena idea. Como de todas formas hay que empezar por la base, decidí implementar algunos al principios e incorporar más adelante otros.

Backend:

El backend al principio fue duro. Pese a que javascript y typescript son parecidos, nunca había trabajado typescript y tuve que ver un video para hacerme más cómodo al lenguaje.

Una vez aprendido eso tuve que hacerme a la forma de organizar las cosas en nestjs. Una forma en la que tampoco estaba muy acostumbrado que digamos.

Una vez entendí la base, empecé a instalar dependencias y a hacer un poco el esqueleto para poder tener un crud de USERS. Sé de sobra que aquí el camino crítico no son los users, son las tareas, pero confío plenamente en que será fácil hacer una cosa teniendo la otra, ¿y si no? No sé para qué me he tirado estudiando dos años, la verdad.

Reconozco que el esqueleto ha sido un poco tirar de que chatGPT lo haga, sabiendo yo e indicándole como quiero estructurar las cosas y sabiendo lo que hay que hacer, eso sí y por supuesto revisando después todo lo que haya que no entienda. Pero ahorro más tiempo así que si me tengo que meter a buscar documentación en inglés de 3 o 4 cosas diferentes. Tampoco tendría problema en hacerlo si no me quedara ninguna otra opción pero eso no quita que es tedioso. En ese sentido mi lema es: Búscalo ya hecho, entiéndelo y replícalo. Es mucho más rápido que si tienes que indagar en la documentación y averiguar cómo se hace y al final vas a llegar al mismo camino, aprenderlo bien. (Para mi suerte tendré que replicar lo que hice en users en tasks, así que me vendrá genial)

Durante la fase del montado de la API, también he tenido problemas con la conexión de la base de datos. Primero elegí prisma y me llevó un buen rato, múltiples errores y cuando finalmente parecía tener todo atado, el error era que oracle no se detectaba como gestor de bbdd. Así que tuve que pasarme a TYPEORM.

La configuración ya me era más conocida y fácil de emplear y la conexión a la bbdd más fácil. Rápidamente pude hacer un get usando postman y luego para el post tuve algunos problemas como que faltaba el header content-type y luego de eso mandar el JSON mal ya que faltaban las comillas en las claves. Por culpa de eso no se detectaba bien el JSON y me saltaban múltiples errores que nada tenían que ver con lo que tenía que resolver.

Con esto ya podía crear usuarios y obtenerlos (por ahora borrarlos y actualizarlos es irrelevante)

Después de eso estuve refactorizando código para no tener mucha configuración de la base de datos en el archivo principal del proyecto.

Luego empecé a hacer las migraciones que me dieron bastantes problemas:

Realmente no sé como lo resolví porque fui dando muchos palos de ciego. Busqué usando diversas ias y diferentes fuentes (he dejado al margen esas consultas de ia porque fue ir dando vueltas en círculos hasta encontrar la salida), entre ellas un vídeo de alguien que realizaba las migraciones y la documentación oficial. Me llevó bastante rato por los errores que daba y aún así solo conseguí que la migración fuera parcial, es decir, el comando que la generaba automáticamente solo generaba los cambios que había en la bbdd, no generaba una migración desde 0, con lo que tuve que hacer la migración yo a mano. Sin embargo, la experiencia de aprender a configurar comandos para hacer las migraciones en el script del json (que también me dio diversos problemas) también estuvo interesante.

Al terminar las migraciones me puse con los bearer tokens y el JWT. Tuve que instalar algunas extensiones, configurar el módulo de autenticación y el de swagger (este último sobre todo por limpieza de código). Tuve varios problemas por el camino como no darme cuenta de donde se introducía el código en swagger y hacer algunas configuraciones para que el token no fuera estático.

Como fueron muchos problemas, si te interesa profundizar puedes mirar la conversación con chatGPT desarollobackend6.

Después de eso me pongo a preguntar dudas de los cabos sueltos que se hayan quedado y subo el crud de usuarios a un repositorio.

Sigo con el CRUD de categorías.

Descubro gracias a la IA que con el comando generate resource ya me viene todo el crud básico hecho ¡Y ADEMÁS SE IMPORTA SOLO AL app.module!. Ahora que entiendo como se hace y el flujo básico es mucho mejor hacerlo así que ir poco a poco, fichero a fichero como iba antes, para acelerar más las cosas.

Durante este proceso observe que había inconsistencias en la bbdd y el código. Esto pude observarlo porque las migraciones se generan a partir de los cambios que hay entre una cosa y otra y la idea es que ambas cosas fuesen iguales.

Estuve haciendo los cambios en el código para que este se ajustara a las especificaciones que yo tenía en la cabeza.

Después, al pasar a implementar el POST de Categorías tuve algunos problemas con el envío de datos y la gestión de TYPEorm ya que al trabajar con relaciones en la entidad es recomendable colocar el objeto entero por si fuese necesario acceder a sus propiedades (yo no necesitaba hacer esto pero quería practicar). Y esto causaba que a la hora de enviar la información se debe hacer de una manera muy concreta, poniendo el nombre del objeto seguido de llaves con los atributos que se quieren mandar. Aunque esto parece muy fácil la realidad es que me tuvo unas cuantas horas dando vueltas, sobre todo porque la aplicación tiraba errores de SQL relacionados con que la información no estaba llegando bien.

Me puse con el resto del CRUD y pude hacerlo bastante fácil, a excepción de un error donde la API se colgaba debido a que no estaba manejando el error adecuadamente. Esto ocurría porque estaba usando líneas que devolvían promesas con más código a continuación, sin embargo, no estaba usando async-await, lo que causaba que, en caso de fallo, la API se colgaba por completo por la falta de manejo de ese error y porque no se estaba esperando a la resolución de la promesa adecuadamente antes de seguir.

Después de mejorar la documentación y resolver algunos bugs / mejorar el código, me puse con el POST de task, y tuve que afrontar varios problemas e incognitas.
Al poderse crear tareas sin categoría ¿Qué valor introduzco en la base de datos? ¿Cómo lo manejo en el front? En qué sentido modifico las migración y entidad?
Hasta ahora el userId se obtenía del token JWT y no venía con el DTO. Ahora necesito transmitir la información del categoryId que, no está en el JWT, pero por otro lado no la elige como tal el usuario. ¿La pongo en el DTO? ¿Cómo transmito la id si en el código estoy jugando con el objeto relación en su totalidad? Estas preguntas y otras cuestiones así como problemas que han ido surgiendo durante el desarrollo (así como olvidar incluir el estado en la información de la tarea a introducir, que siempre es 0 y el usuario no la decide) ha sido lo que he tenido que hacer frente y que se encuentra recogido en la hoja desarrollobackend10.

Por hacer un resumen, he dejado la categoryId en task como nullable, y para que los endpoints no den problemas, siempre que quiero apuntar a buscar null o a introducir o a modificar a null en la BBDD juego

con el -1 y en el service, que se encarga de la lógica transformo y convierto este dato. Por otro lado he preferido dejar null en la BBDD como dato por defecto porque según vi es lo convencional y además da menos problemas en algunos aspectos que ponerlo personalizado. Sobre todo y más que nada porque la categoría -1 no existe, pero con null es diferente. Oracle entiende automáticamente que no debe de vincularlo estrictamente cuando el campo vale null. Así se evita crear un registro fantasma y la implicación de tener que pensar a qué usuario vincular esta categoría fantasma.

El resto del CRUD fue bastante fácil teniendo en cuenta el trabajo que ya tenía detrás y solo tuve algunos problemas con el campo category_id y sus peculiaridades.

FrontEnd:

Al terminar de configurar mi proyecto vi que tenía bastantes problemas. No recordaba muy bien como usar SASS, tampoco había trabajado con Angular, no conocía la estructura de los proyectos y además iban a ir surgiendo varios problemas durante el desarrollo. Así que me tocó hacer un curso acelerado de cómo funciona.

Después de aprender lo básico de angular y configurar mi proyecto con bootstrap y Sass para tener todas las opciones abiertas, estuve un rato decidiendo con qué estructura de carpetas montar mi proyecto.

A continuación hice una barra simple de navegación y un footer como componentes que se encontraban dentro de un componente mayor llamado layout. Estos componentes los hice siguiendo una serie de pasos y más o menos empecé a entender como funcionaban las cosas gracias a saber como hacer esto.

Lo siguiente era hacer las páginas a las que iban a llevar y darle funcionalidad a la navbar.

Generé una plantilla fácil y agradable a la vista y le puse la funcionalidad de navegación con el routerLink. También había un botón en el home que hacía la redirección.

Después de eso hice un servicio y varias pruebas para conectar con la API y finalmente conseguí hacer un register y login (que mostraba por alert el Access token).

Hasta aquí no tuve mayores problemas, salvo que me encontraba entendiendo el entorno y es por ello que me demoré bastante rato en hacerlo.

Lo siguiente es la gestión del token para controlar la sesión y ya de paso hacer diferentes páginas para cuando se está logueado o no.

La verdad es que tampoco tuve demasiados problemas. Fue ciertamente difícil entender la estructura que tenía que montar porque nunca me había adentrado tanto en angular, pero siguiendo los pasos todo ha ido saliendo más o menos a la primera.

Reconozco que no he tenido tiempo para hilar tan fino ni controlar las cosas de manera tan clara como hice con el backend, ni tampoco tuve el tiempo de dedicar a pensar las cosas y hacer el trabajo de conciencia de tratar de toparme con los errores típicos de ir haciendo lo que ‘yo creo que es’ para luego darme cuenta de cómo es en realidad, pero si hubiera querido hacer ese mismo trabajo no creo que hubiese terminado la aplicación a tiempo.

Además de hacer funcionar el token, hice algunos ajustes en la manera de enrutar y las redirecciones y también implementé validaciones en el front para las contraseñas y el email.

Lo siguiente es hacer que funcione el cierre de sesión. Lo hice fácilmente también.

Después me centré en poner unos datos que no fueran estáticos y que se pudieran modificar así como la obtención de los mismos de la API. Previamente tuve que modificar el esquema que tenía para adaptarlo a mis campos de la BBDD.

Una vez tenía la recolección de datos para ver el perfil según el usuario que está logueado, me puse manos a la obra con el botón de modificar perfil y el de borrado de cuenta.

Estos me dieron bastantes problemas ya que me vine arriba creando componentes para una ventana modal y esto me dio problemas varios, por ejemplo con ciertas dependencias o con el componente en sí. Al final muy a duras penas conseguí sacarlo adelante, pero me llevó un buen rato.

En desarrollofrontend5 y 6 están algunos de los problemas que tuve.

Para finalizar hice el CRUD de categorías y tareas, que me dieron diversos problemas, pero la verdad es que, sin dolor ni gloria, hice un trabajo de fondo para tratar de entender demasiado, en ese punto ya estaba un tanto desesperado por terminar el trabajo y me limité a copiar y pegar de chat gpt, ver los problemas y solucionarlos. Algunos de ellos chat gpt no pudo solucionármelos y en un momento de más desesperación aún tuve que ponerme yo a entender el código para arreglarlos yo mismo. Al final si que me acabó por servir para algo y tuve que dedicarle un buen rato a subsanar todos estos problemas

También aproveché para poner las estadísticas en la página de perfil..

De todas formas si quieres ver el proceso, puedes consultar las conversaciones desarrollofrontend7 hasta la 10.

Una vez hecho todo eso simplemente faltaba pulir las páginas con estilos, usar el TOASTR y alguna otra cosa más de cambios menores, así como resolver algún bug que se vaya presentando, pero el trabajo en sí ya estaba hecho.

El ToastR también me ha dado bastantes problemas. En primer lugar quise poner iconos de bootstrap, pero ToastR también ponía sus iconos por defecto en un color blanco poco distinguible. Para solucionarlo finalmente establecí el tamaño del background (no el background image o el background color) a 0. Sin embargo, dar con esta solución me costó horrores de tiempo.

Otro problema que tuve fue que la configuración que le estaba metiendo al toastR no se estaba aplicando bien. También conseguí solucionarlo pero me costó otro buen rato. (Se puede ver en desarrollofrontend 11 y 12)