Planificación de Proyectos:

Considero que para tener una buena planeación de cualquier lenguaje de programación, es entender muy bien que impacto y hasta donde se debe llegar con el proyecto, esto implica, saber cómo se modularizara, que componentes y diferentes parte del proyecto tendrán más prioridad, para tener encuentra en cada entrega, si de metodologías agile estamos hablando.

Que parte del proyecto tendrá más impacto en el cliente y si en realidad esto hace que el proyecto cumpla con un producto mínimo viable.

Por lo general el concepto del negocio es el que se prioriza y se pone en marcha y se revisa su avance en cada ceremonia celebrado con todos los stakeholders.

Angular en estos casos puede ser un gran aliado para proyectos que pretenden escalar rápidamente, ya que promueve la arquitectura modular, entonces dicho esto, Angular parece perfecto para cuando solo se requiere comenzar ciertos módulos por sprint, si bien algunos módulos pueden ser grandes, estos se pueden desglosar en partes muchas más pequeñas como compontes. Esto es bueno, ya que cada componente se debe diseñar de tal manera que se pueda reutilizar, bien sea en su mismo modulo o uno externo.

En la asignación de recursos, primero debemos revisar el alcance y la duración del proyecto para determinar cuánto tiempo tomará completarlo. También es esencial definir la frecuencia de las entregas continuas al cliente. Con esta información, podremos asignar épicas o historias de usuario a grupos específicos de desarrolladores que tengan las habilidades adecuadas para cada tarea.

Por ejemplo, si el proyecto es a gran escala y se planea completar en 12 sprints (cada sprint dura 15 días o dos semanas), es necesario terminar seis módulos en su totalidad. Para lograrlo, podríamos formar un equipo compuesto por dos desarrolladores experimentados en la herramienta a utilizar, un líder técnico que actúe como enlace entre los aspectos técnicos y comerciales, un product owner que supervise el avance y evalúe la viabilidad de cumplir con los objetivos, proponiendo soluciones en caso de ser necesario, y un tester encargado de probar las nuevas funcionalidades que se vayan implementando en cada sprint.

También utilizar herramientas de gestión de proyectos para facilitar la comunicación, el seguimiento del progreso y la asignación de tareas, ejemplo: Azure devops, Jira u otros.

Gestión de Dependencias:

Por lo general, siempre analizamos que el framework con que estemos trabajando, tiene lo necesario para cumplir con el objectivo planteado y ver si realmente necesitamos de una dependencia, ya que esta nos puede ayudar a avanzar mas rapido en vez de generarnos reprocesos o reinventos de la rueda.

Considero que una de las cosas más importantes al gestionar una biblioteca o dependencias es asegurarse de que el paquete sea totalmente confiable. Es fundamental revisar que se estén realizando actualizaciones regularmente y que su puntuación inspire confianza entre los usuarios que lo descargan. Si es de código abierto, es importante verificar que haya colaboradores activos que constantemente suban sus pull requests (PR).

En ambientes de javascript como node y Angular, contamos con npm audit, es importante monitorear estas dependencias en vez en cuando ya que podrfiamos detectar vulnerabilidades.

Tambien podria decir que algo que nos puede ayudar a gestionar muy bien las dependencias, son las actualizacions de sus versiones, ya que en ocaciones en la version que esta mas actualizada se hacen algunos parches de seguridad y asi.

Para mi caso, he contado con herramientas de GitHub como Dependabot, que me notifica cuando una libraria, biblioteca o bien una dependencia, tiene una vulnerabilidad o esta deprecada.

Por ultimo concidero que si ya no se usa mas las depencnecias, deberiamos eliminarlo de nuetro package o cualquier otro gestor de dependecias.

Colaboración en Equipo:

Mi respuesta siempre, va ser que gracias a las metodologías agile, hoy por hoy, vemos más a fin el trabajo colaborativo en equipo, ya que todo el tiempo sabemos de los avances de mi compañero y si bien, pueda que haya algo en que yo ya haya trabajo podría colaborarle para que podamos cumplir con el objetivo que se planteó en el Sprint.

Ceremonias como la planning, ayuda a saber cómo nos repartiremos las obligaciones y por medio de la estimación, votaremos según nuestra perspectiva, en incertidumbre, esfuerzo y complejidad, y tal vez lo que es muy complejo, para mi compañero, para mí no lo es tanto, entonces ya sabemos que el tendrá un apoyo de mi parte, para cumplir los objetivos de todo el equipo.

Priorización de Tareas:

Como dice la frase cliché, divide y vencerás.

En los últimos sprints, hemos enfrentado procesos complejos que no pueden completarse en un solo sprint. En estos casos, priorizamos junto con todos los implicados qué funciones pueden ser útiles para el equipo de testing y aporten un valor real. De esta manera, evitamos intentar hacer todo de “tacazo”, lo cual podría resultar en funcionalidades incompletas y con bugs que necesiten ser solucionados posteriormente.

Control de Versiones:

Al tratarse de proyectos en su mayoría grandes y que se tienes que hacer entregas continuas del mismo, siempre contamos con un equipo capacitado para realizar dichas entregas, y para que los equipos se puedan entender a la hora de realizar sus cambios o subir una nueva funcionalidad, tenemos el control de versiones como GIT.

Personalmente he trabajado con equipos grandes, trabajando para un mismo proyecto, y casi siempre hay un problema en general y es el tema de los conflictos.

Para esto, siempre en los equipo que he estado, usamos Git Flow, que es una manera de priorizar ramas de Git, y por ejemplo, casi siempre en un proyecto nuevo, el líder técnico, comparte un repositorio, casi siempre con una plantilla preparada y esta esta ubicada en la rama master.

De la manera en que he trabajado, siempre sacamos nuestra rama de master, nombrándolas según el caso por ejemplo si es un nueva características, podría ser algo: feature/sprint-1/123, el cual indicamos que trabajaremos una feature, en el sprint 1 y ponemos el número del ticket o work item asociado, para tener un mejor seguimiento a que rama pertenece una historia de usuario.

Luego de haber trabajado en la rama individualizada, cuando ya se hizo el trabajo completo, se procede a crear un pull request (PR) a unas de las ramas priorizadas, en mi caso he trabajo con uat y de. Uat es donde los tester probaran las nuevas funcionalidad y verificar que todo haya quedado bien y dev, es complemente para que e desarrollador, revise que lo que hizo si se cumpla según los lineamientos y criterios de aceptación.

Luego de saber que los cambios hechos fueron aprobados por LT y hay visto bueno por el lado del equipo de testing, quiere decir que la nueva funcionalidad ya puede unirse con la rama master, pero antes los cambios tienes que pasar por una rama llamada reléase, quien es la que pretende desplegar la nueva funcionalidad a producción.

En los casos de los conflictos, siempre y cuando sea que un compañero hizo algún cambio, la idea es hablar con el y revisar sus cambios, ver que va o no.

Cuando hay conflictos porque en una rama (por ejemplo, master) hay cambios que no están en UAT, o viceversa, generalmente procedo de la siguiente manera: si mi rama se creó desde master y debo fusionarla en UAT, creo una nueva rama a partir de UAT (temp/sprint-1/123) y realizo un merge con mis cambios de la rama originada en master (feature/sprint-1/123). Me gusta usar Visual Studio Code para resolver conflictos. Así, mantengo los cambios de master en master y los de UAT en UAT, resolviendo los conflictos en el proceso. Finalmente, hago un pull request de la rama creada a partir de UAT en UAT.

Manejo de Errores:

Donde más se nos puede presentar temas de errores no controlados, es en la parte de servicios, ya que no podemos saber si el back nuestro o de un tercero, deje de funcionar o ya no tenga el recurso solicitado.

Para estos casos gracias a que angular esta sobre TypeScript, podríamos saber de que manera nos llegan las respuestas y así mapearlas, a nuestra conveniencia y taparlas para generar menos errores dentro del código.

También en los servicios contamos gracias a librería rxjs, el cual viene con catchError, el cual ns dara manjo si algo fuera de lo normal ocurre.

Tampoco estamo esscento de los errores en los compoenentes, en este caso podemos hacer uso, del ya conocido de toda la vida try catch.

Tambien como mejoras, podriamos implentar un manejador de errores centyralizado, esto haria que el codigo se mantuviera limpio y facil de mantener.

Pruebas y Aseguramiento de la Calidad:

Por el momento, solo he realizado una que otra prueba unitarias, básicamente siempre garantizando el funcionamiento del código y la calidad del mismo, temas de que si una funciona ha sido llamada con x o y parámetros y que realmente cumpla con el type asignado o a devolver.

También he trabajado en temas del coverage, para saber que tanta afectación puede tener en caso que el día de mana el código sea modificado.

Estrategias de Contenerización:

He trabajado con Dockerfile, para el cliente puntos Colombia, la ventaja que le veo en dockerizar aplicaciones y más una de angular, es que nos ayuda y nos da una especie de garantiza que nuestro código se ejecute en cualquier entorno.

Para la gestión de variables de entonces por lo general usamos el siguiente código:
docker run –rm -ti –env.file .env -p 80:80 –name <nombre-aplicacion>

Monitorización y Optimización del Rendimiento:

Por lo general, las aplicaciones web, con el tiempo mientras más features implementamos más pesadas se vuelven, es qui donde considero, que es necesario, el monitoreo, ya que una web lenta, hace que el usuario considere irme a otra, además monitorear la web también nos sirve para saber en qué están más interesado los usuarios y si tenemos un mapa de calor podremos saber a qué darle más importancia u oportunidades de mejoras.

También es importante analizar la carga de la aplicación, por lo general he usado la herramienta lighthouse, para revisar el seo y ver que parte de mi aplicación necesita mas atención.

Estos problemas los he solucionado con, optimización de imágenes, cargas diferidas o lazy load, de módulos o compontes, y los children en las rutas, ya que si no estoy usando aun un módulo o componente este no tiene por qué cargarse al comenzar la web a ejecutarse.

Documentación del Proyecto:

Tener un proyecto bien documento, debería ser el pilar en todo desarrollo, porque uno debe pensar que no solo uno va tocar dicho proyecto, sino que hay un equipo colaborativo y si bien el día mañana, entra una nueva persona, esta debería desentenderse con la documentación que hay y debería ser lo más amigable posible para él.

Una buena documentación, nos garantiza que el código pueda ser mantenible en el tiempo, ya que sabremos que el método o el código legacy que iremos a modificar, solo permite ciertos parámetros.

Como tipo de documentación, diría yo, que es muy bueno el README de cada proyecto, ya que explica un poco como y de qué manera está construido.

También me gusta contar con diagramas secuenciales y las umls que puedan ayudar a entender el flujo de una app, también incluiría las documentación swagger, para el caso de los endpoints, de los backends.

Por ultimo he puesto en últimos meses el jsDoc y TypeDoc, para describir lo que hace una dicha función o proceso en general de algún código construido por mí.

Manejo de Datos Asíncronos:

Me gusta trabajar plenamente con lo que Angular nos proporciona. Para este caso siempre para manejar datos asincrónicos uso Observables, ya que Angular tiene su propio cliente http (HttpClient), cuando nos suscribimos a los observables, recibimos los datos y podemos manejar los errores con el `catchError`.

Migración de Versiones:

Por ejemplo cuando se haya mejorado el redimiendo y x o y modulo, librería y este afecta en el rendimiento de mi aplicación.

Primero, analizaría el impacto real, de migrar una versión a otra. Hasta donde se, las versión 6 hasta la 16 son las mismas, solo que tienen una que otra mejora en cuanto a cómo se hace esto a como se hacía antes, mejoras en temas de reactividad y así, en esos caso el impacto es menor, así que yo procedería hace la migración y haría pruebas unitarias para ver que el flujo sigue siendo el mismo en cuanto al código y regla de negocio se trata.

Segundo seria actualizar las dependencias, con las que la nueva versión requiere para trabajar

Tercero, haría prueba de regresión con el equipo de testing, ya que todo debería estar funciona como antes y es más deberían haber mejorados los tiempos de respuestas.

Y por último monitorear la app y garantizar el correcto funcionamiento.

Resiliencia y Manejo de Excepciones:

Básicamente como lo dije más arriba, podríamos optar por una centralización de manejo de errores teniendo como protagonista el servicio de ErrorHandler, ya que lo podemos moldear o extender, para customizarlo.

Si el usuario solicita un recurso y este no se encuentra, mostrarle la página correspondiente con un 404 descriptivo, y si es otro código de error como 500 o 502, trabajarlo por consola o con un sistema, de monitorio o de loggin, tipo splunk.

Autoaprendizaje y enseñanza:

¿Cómo te mantienes motivado y enfocado durante el autoaprendizaje?

Las tendencias en el mundo TI, además soy de los que piensa que entre más aprendo más me doy cuenta de lo poco que se, así que siempre busco la manera de ir un pasito así sea leyendo un capítulo de https://medium.com/ para saber cómo estamos y para donde vamos y que está en tendencia en el mundo TI.

Además en el caso del autoaprendizaje, siempre me pone feliz aprender algo nuevo, resulta que a veces trabajamos en una tecnología y hacemos las cosas tan rutinarias que cuando ver un post o un artículo de alguien que lo hace de manera más eficiente, te asombras, entonces esta carrera es de eso de emociones y de siempre estar enfocado “engomado”.

¿Cómo integrarías el autoaprendizaje en tu rutina diaria como desarrollador de software?

Como en toda la vida, con tiempo, siempre tiene que haber un tiempo para todo, como cuando sacamos tiempo para ir al gimnasio, así sea de media a una hora diaria.

Librería Angular:

Últimamente me he interesado por Angular material, por que trae muchos componentes que me ahorran demasiados trabajo, a parte no soy muy bueno en el diseño, entonces para temas que tienen que ver con crear aplicaciones rápidas, confío en angular material y solo es usar sus módulos y si quiero usar un componente de autocomplete, solo, seria ir a la doc, revisar su ejemplo y su implementación y ya está, es muy fácil de entender y útil para entregar soluciones rápidas con diseño atractivo.