Respuestas

1.

Se necesita una aplicación web principal que maneja cuatro tipos de usuarios: clientes, tiendas, repartidores y administradores del sistema. Las tiendas podrán crear sus comercios y subir sus productos, los clientes podrán buscar y comprar productos, los repartidores recibirán pedidos que ir a recoger y entregar al cliente y los administradores que podrán realizar acciones que permitan solucionar problemáticas del día a día. La aplicación web va a contar con una API de backend y una aplicación de frontend usando React u otro. También se necesita un servicio batch que corra periódicamente para rescatar los pedidos que no se están procesando (que no fueron aceptados por el comercio o no fueron aceptados por un repartidor). Además se necesita contar con varios modelos corriendo en servicios de API (con ayuda del DS y MLE): predicción de capacidad de reparto en un zona, tiempo de preparación de tienda y entrega de pedidos.

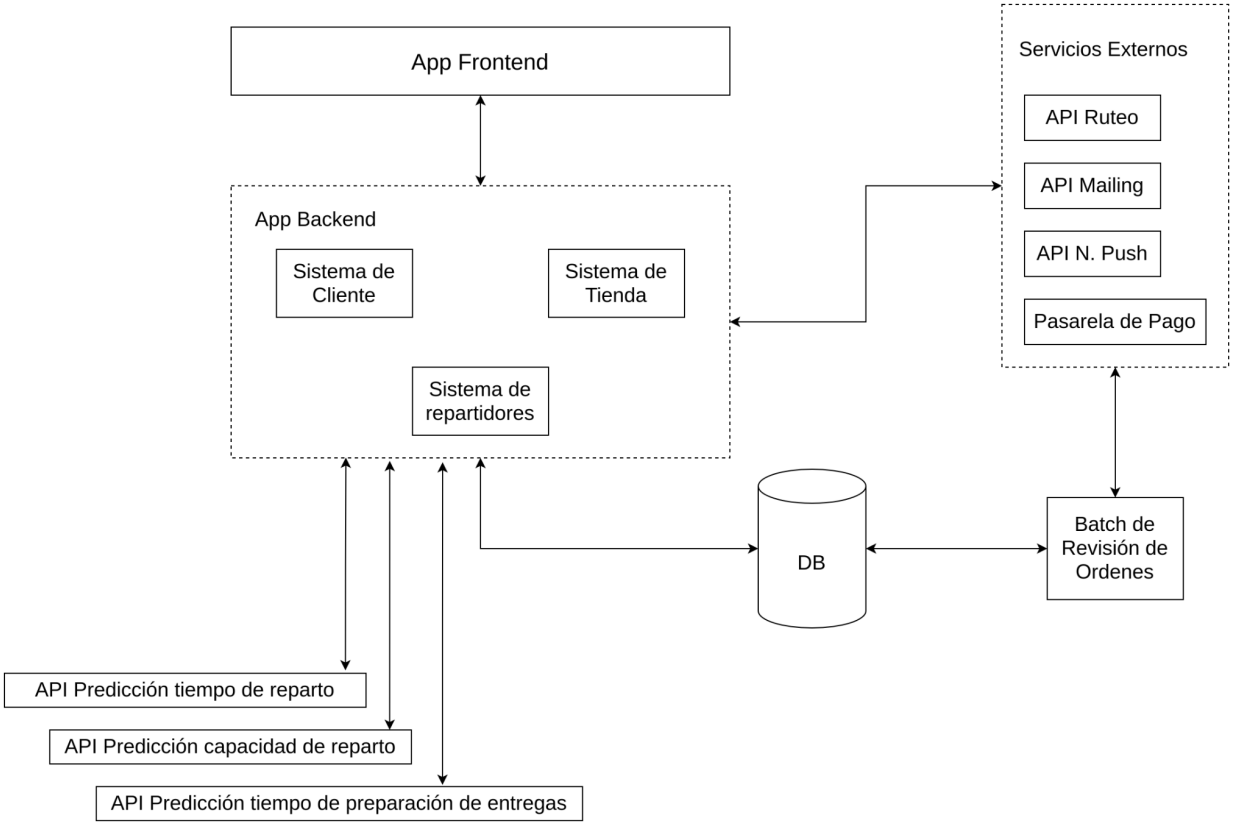

Figura 1. Diagrama de aplicaciones sistema de delivery.

2.

Se podría considerar que el tipo de arquitectura es de micro servicios: una API de backend (la aplicación más grande), una aplicación js de frontend, un servicio batch de rescate de pedidos, tres APIS con modelos de predicción. Una arquitectura de este tipo tiene la complejidad de gestionar el entorno en donde viven todos estos servicios, lo que sin duda puede no ser deseable en una prueba de concepto. Sin embargo, el beneficio de separar los entornos de trabajo de los diferentes equipos y la velocidad de entrega y despliegue que se gana con esto, favorecerá el desarrollo. Además, el mantener esta separación de servicios nos permite tener una arquitectura más específica para los requerimientos particulares de cada uno de los sistemas: API de backend puede tener un balanceador propio al igual que las API's de predicción, mientras que el frontend se puede desplegar en un CDN.

3.

La metodología de trabajo que aplicaría sería Scrum. Usar esta metodología implica gastar tiempo en realizar sus ceremonias y mantener su estructura a lo largo del tiempo. Sin embargo, más allá de sus ceremonias creo que hay dos áreas en dónde radica principalmente el valor que veo en su uso: Existe un punto de encuentro en las historias entre el equipo técnico y el área de negocio y permite evaluar empíricamente la velocidad de desarrollo del equipo. Es muy importante que exista una sinergia entre el negocio y el equipo técnico, al punto de entender las decisiones y cambios y que estas no sean vista como un problema para el equipo técnico sino que desde el beneficio que entregan al negocio. Tener una medición empírica del desempeño del equipo ayudará a entender si se puede lograr la velocidad de despliegue de las características de la plataforma.

4.

Aplicaría gitflow pero sin usar pull request en los merge. La gracia es que permite mantener ordenado el desarrollo y mantener cierta independencia de las características nuevas vs la plataforma que ya está desplegada. Es importante notar, que dependiendo del esquema de despliegue (el proceso para pasar a staging y luego a producción) o por decisión de separar el desarrollo de los diferentes servicios, se necesitara tener repos separados para cada equipo. La idea de no usar pull request, es que dado el tamaño del equipo y que se espera que tengan cierta independencia y experiencia suficiente para manejar los despliegues, es que mantener los pull request solo agregaria mas trabajo a un despliegue sin entregar beneficios para separar el equipo de despliegue del de desarrollo (separación que tiende a no existir en un equipo pequeño).

5.
A priori el equipo está bien balanceado: contraparte de negocio, equipo para abordar separadamente el frontend del backend y equipo para las API's de predicción. De todas formas, para poder tener claridad sobre la necesidad de expansión de un equipo, es necesario estimar el esfuerzo inicial de las características que se quieren desarrollar. Ahora solo conociendo la visión sin el modelo de negocio que se quiere probar es muy difícil asumir la necesidad de más equipo.
Tomando en cuenta lo anterior, puedo decir que la necesidad de más equipo depende del esfuerzo estimado y el tiempo en que se quiere lograr el despliegue, junto a los recursos que se disponen para lograr la tarea.
Durante el desarrollo, el mantener claridad de la capacidad para desplegar historias del equipo va a permitir conocer la necesidad futura de expandir alguno de los equipos, dada la velocidad de desarrollo que se quiere mantener.

6.
Es necesario tener en consideración la comunicación entre el área de negocio y el equipo de desarrollo. No es una buena práctica el mantener silos, en donde el equipo de negocio toma decisiones sin considerar el equipo técnico y sin explicarles el porqué. Este tipo de relación lleva a que el equipo técnico tenga que estar trabajando a pedido, perdiendo capacidad de análisis sobre el problema que resuelven y con alta probabilidad de construir código de pobre calidad (propenso a errores, difícil de mantener o modificar, etc). Esto se intenta solucionar con Scrum y las historias, pero es necesario considerar prácticas de DDD, al incluir en el diseño y desarrollo conceptos de negocio por sobre otros conceptos y poner énfasis al análisis del problema que se resuelve.
También es necesario mantener buenas prácticas de desarrollo (clean code, entre otras) y despliegue (testing en diferentes niveles según sea necesario).