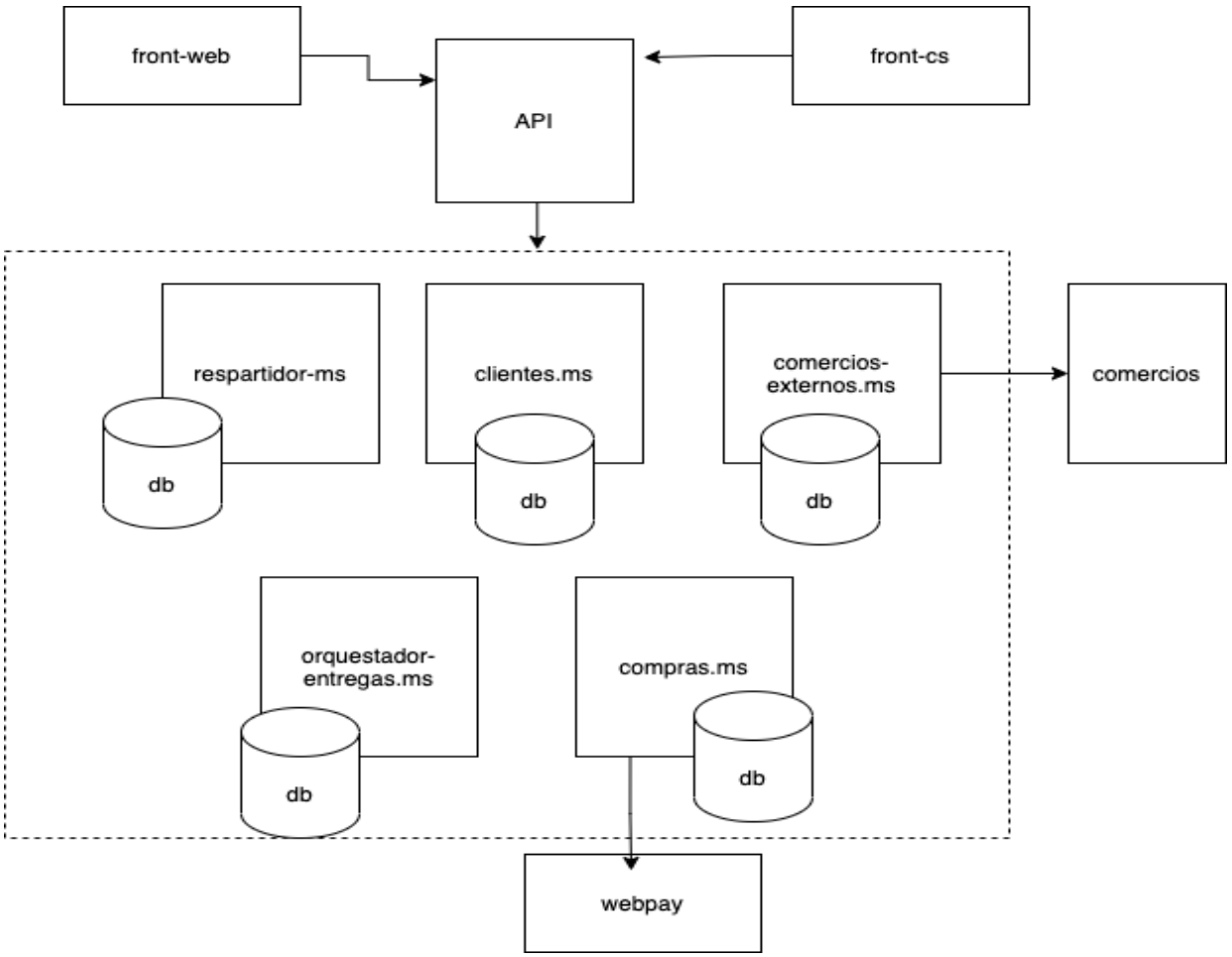
1. La solución debería contar con varias piezas de software, como un/una:

- **Aplicación móvil:** una aplicación móvil, para dos tipos de usuario: **cliente** o **repartidor**.
- **Aplicación web:** una aplicación web de **uso interno**, que sirva tanto para registrar comercios en la aplicación, como para trabajar con tickets levantados por usuarios.
- **API:** una API que exponga endpoints al web y mobile y consuma endpoints de los distintos microservicios.
- **Microservicios:** microservicios que realicen la lógica de negocios, y sean consultados por la API, u otros microservicios, para el intercambio de información.
    - **repartidor ms:** microservicio dedicado a procesar requests e información sobre repartidores.
    - **clientes ms** : microservicio dedicado a procesar requests e información sobre clientes.
    - **comercios-externos ms:** microservicio dedicado procesar requests e información sobre los comercios que se muestran en la app móvil de cara al cliente. Deberá usar **técnicas de scrapping** para actualizar la cartera de productos de los comercios registrados.
    - **orquestador-entregas ms:** microservicio dedicado a la orquestación de entregas, optimizando asignación de repartidor y tiempos de entregas.
    - **compras ms:** microservicios dedicado al flujo de **compras y pagos**, debe conectarse con la API de webpay u otros **medios de pago digitales**.



1. Se escogió una arquitectura de microservicios, debido a la complejidad de la aplicación que se quiere desarrollar. Ya que la aplicación cuenta con procesos “costosos”, como la orquestación de pedidos, y otros “sensibles” como la realización de pagos y manejo de transacciones.

   Por lo tanto es bueno **desacoplar esta lógica**, para realizarla de forma más **eficiente** y **evitar y/o reducir fallos** en procesos core (pago y transacciones).

2. La arquitectura propuesta es bastante compleja para un prototipo, por lo que inicialmente propondría un ciclo de trabajo largo, como **shapeUp**, para construir el “esqueleto” de la arquitectura, montar los microservicios y tener

todo conectado. También se podría usar este espacio para generar las "tarjetas" a tomar una vez desarrollado el esqueleto.

Luego, teniendo la arquitectura deployada, trabajaría con ciclos ágiles como SCRUM para sacar todas las features necesarias. Ya que en mi experiencia trabajando con SCRUM, esta metodología requiere tareas bien definidas y estimables. Lo cual es difícil de lograr en la etapa inicial del desarrollo.

3. Para el flujo de trabajo se utilizará Git, contando con una rama principal y ramas secundarias.
   Para el desarrollo se "sacarán" ramas desde la rama principal. Cuando se quiera hacer un merge con el cambio a la principal, se deberá realizar una **Pull Request** y contar con la aprobación de al menos otros 2 integrantes del equipo.
   La PR correrá un set de **githHub actions**, que contará con chequeos de tests, linter y vulnerabilidades.

   Al publicar la rama con la nueva feature el código pasará por un pipeline con al menos 3 ambientes.
   - dev: ambiente de desarrollo
   - stg: ambiente para realizar pruebas
   - producción: ambiente productivo

Por lo tanto, a pesar de que el código esté en main, el cambió no será productivo hasta que llegue al ambiente de producción.

5.

Durante la de prototipo, sería bueno contar con la guía de un cloud Engineer (alguien que tenga más experiencia trabajando en la nube), esta persona ayudaría al software engineer a levantar y asignar recursos a los servicios en la nube además de ayudar al equipo a mantener el costo del servicio dentro del presupuesto.

Luego de la etapa de prototipo este cloud engineer debería poder trabajar en asignar los recursos óptimos a cada servicio en la nube, y crear un plan de escalamiento.

6.

Otra consideración necesaria para un proceso de desarrollo robusto es la realización de tests unitarios del código.
De ser necesario, se pueden realizar mocks de servicios externos (puede ser que durante la fase del desarrollo del prototipo no sea posible conectarse a webpay, o con comercios) con librerías de testeo, que permiten “stubbear” estos servicios.

Tras la fase de desarrollo, podría ser útil hacer un “hacking ético” de la aplicación. Para ver posibles vulnerabilidades en el manejo de dinero, leaks de información o problemas de seguridad