Desafío SpikeLab – App de delivery

Ari Romero G.

Piezas de software para el prototipo funcional

Las piezas de software principales que este servicio debería tener, considerando que debe ser funcional para clientes y repartidores, son al menos:

- App Móvil Repartidores: Permite tomar pedidos, ver ruta de entrega, actualizar estado de pedidos y registrar nuevos repartidores.
- App Móvil Clientes: Permite crear pedidos, obtener información en tiempo real del pedido y registrar nuevos clientes.
- Sitio web:
  - Versión web de app clientes, para uso en escritorio.
  - App web para repartidores.
  - App web para comercios.
- App Web Soporte: Permite ver pedidos de los clientes y sus estados para realizar gestión de soporte sobre ellos.
- APIS: Dado que el servicio es multi aplicación, es necesario contar al menos con las siguientes APIS.
  - API Pedidos: crear, ver y listar pedidos, cambiar estados.
  - API Seguimiento: para realizar seguimiento de pedidos a través de la App web y App móvil.
  - API de ruteo: calcular ruta y distancia, entregar un precio por el servicio.
  - API de pagos: recibe pagos de los clientes.
  - API de registro y autenticación de usuarios.
  - API de registro de comercios.
- Procesos: Se incluyen procesos que deben operar
  - Proceso de pago a comercios.
  - Proceso de pago a repartidores.

Arquitectura de la solución

Todas las piezas de software descritas anteriormente pueden convivir en un escenario Serverless, utilizando servicios de un proveedor IaaS como Amazon Web Services, ya que se está en un

escenario de prueba de concepto, pero que podría eventualmente crecer en clientes. Esta selección se sustenta en varios factores:

1. **Escalabilidad:** al externalizar la infraestructura, se puede crecer en capacidad del servicio a medida que los clientes y las transacciones así lo requieren.
2. **Baja inversión inicial:** los costos de infraestructura se pueden considerar modestos mientras se está comenzando a desarrollar el proyecto, por lo que elimina la barrera de entrada de inversión en infraestructura y un equipo de administración.
3. **Seguridad:** dado que que el proyecto se considera una prueba de concepto, es también conveniente mantenerlo aislado de los demás proyectos de la empresa.
4. **Rendimiento:** Los front-end se pueden mantener en CDN's y las API's en funciones como AWS Lambda, servicios que están optimizados para entregar rápidas respuestas y pensados en servir amplias regiones geográficas.

Metodología de trabajo

En este caso, creo que depende más del perfil del equipo de trabajo que del proyecto, creo también que la metodología de trabajo se debe ir adaptando en la medida que el equipo madura. Es por eso que considero que la mejor respuesta para este caso es "sumarme a la metodología ya existente" y realizar un esfuerzo individual por mantener al resto del equipo debidamente comunicado.

Dentro de lo que me ha tocado trabajar, la intención de SCRUM se adecúa a lo que este proyecto requiere, por lo que tomaría este esquema como base.

Flujo de trabajo en GIT

Creo que la rama Master siempre debería ser una versión lista para poner en producción, por lo que tomaría la idea de Gitflow, trabajando sobre una rama para desarrollo y generando ramas para características individuales. Veo que lo más probable es que los repositorios no sean manipulados por más de una persona en un instante, pero aún así, es una buena práctica pensando en que más adelante la base de código puede crecer.

Revisión del equipo de trabajo

Me parece un equipo suficiente para abordar las tareas de desarrollo para el prototipo inicial, aunque claramente faltaría un desarrollador móvil dedicado, ya que éste debería trabajar a la par con el devsigner realizando su contraparte móvil.

Al momento de pasada la fase prototipo, sería necesario agregar algunos integrantes importantes:

- Involucraría a más desarrolladores (Movil, backend, frontend), no sólo para acelerar las tareas de desarrollo y su creciente demanda, sino también para capacitarse en el producto y así no depender de una sola persona para mantener cada área del producto.

- Un perfil DevOps, que se encargue de poner en producción el software debidamente probado y se haga cargo de administrar los servicios cloud asociados al producto.

Ejercicio práctico

El ejercicio práctico, lógicamente muestra el uso de la arquitectura anteriormente expuesta. En particular, utiliza la capa gratuita de Amazon Web Services, incluyendo los siguientes servicios:

- S3 para almacenar una aplicación web escrita en ReactJS.
- DynamoDB para almacenar datos.
- Lambda y API Gateway para exponer una API, también escrita en Javascript.
- Cognito para registro y autenticación de usuarios.

La aplicación es de hecho muy sencilla, aunque tiene el mérito de haber sido desarrollada a medida que aprendía el stack serverless, principalmente porque no podía recomendar una arquitectura en la pregunta 2 y realizar el ejercicio práctico en otra. También fue un auto desafío, ya que así compruebo mi propia capacidad de adaptarme a escenarios nuevos. También lo hizo más divertido.