Trabajo práctico

Técnicas Avanzadas de Programación

- Primer entrega -

Docente

María Belén Alegre

Alumnos

Natalia Bellosi

Javier Escandarani

| 2° cuatrimestre 2020 |

Introducción

El objetivo del siguiente trabajo es plantear las bases para el proyecto de desarrollo del sistema de gestión de ventas y comisiones para un negocio.

El mismo tendrá la capacidad de almacenar y listar usuarios, ventas y productos. El módulo principal tendrá la responsabilidad de calcular las comisiones por vendedor en un periodo de tiempo dado.

El alcance de este proyecto incluye la planificación, implementación, testeo y puesta en producción.

Este proyecto puede ser aplicado a cualquier tipo de negocio que mantenga los flujos y entidades que en el presente documento se detallan.

Plan de trabajo

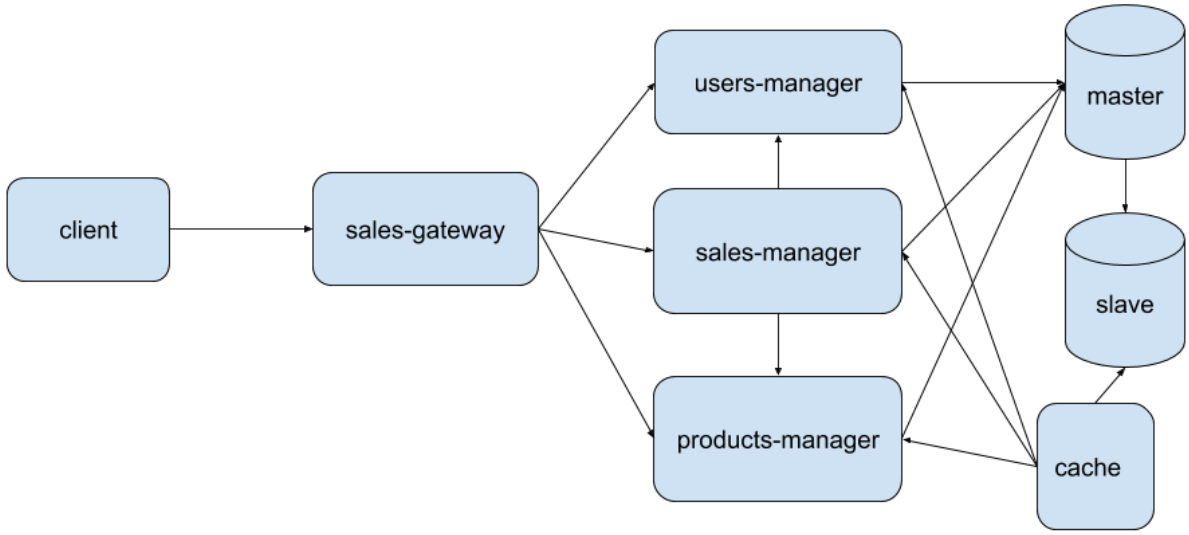
En materia de planificación la presente propuesta contempla:

Planificación de arquitectura y desarrollo: 5 horas
Desarrollo del sistema: 30 horas
Testeo (desarrollo de unit testing y testeo manual): 15 horas
Puesta en producción: 5 horas

Requerimientos

1. Administración de usuarios
2. Listado de usuarios
3. Administración de productos
4. Listado de productos
5. Administración de ventas
6. Listado de ventas
    a. Por fecha
    b. Por producto
    c. Por usuario
7. Obtención de comisiones
    a. Por usuario
        i. 7% si tuvo 10 o más ventas sobre el total de las mismas
        ii. 4% si tuvo menos de 10 ventas sobre el total de las mismas
8. Generación de reportes
    a. Usuario que más vendió en el último mes
    b. Producto más vendido en el último mes

Diagrama de arquitectura



Microservicios

sales-gateway: este microservicio será el punto de entrada para todos los requests que lleguen a nuestro sistema. Se encargará de autenticar a los usuarios y validar que quienes estén solicitando acceso a un recurso tengan el permiso correcto así como también redireccionará los requests a los microservicios correspondientes.

users-manager: contendrá la información relacionada a los usuarios así como también la administración de los mismos. Será el microservicio encargado de loguear a los usuarios en el sistema.

sales-manager: contendrá la información relacionada a las ventas incluyendo su listado, edición, agregación y eliminación. Tendrá la responsabilidad de calcular las comisiones de los vendedores.

products-manager: contendrá la información relacionada a los productos y su gestión.

Dependencias complementarias

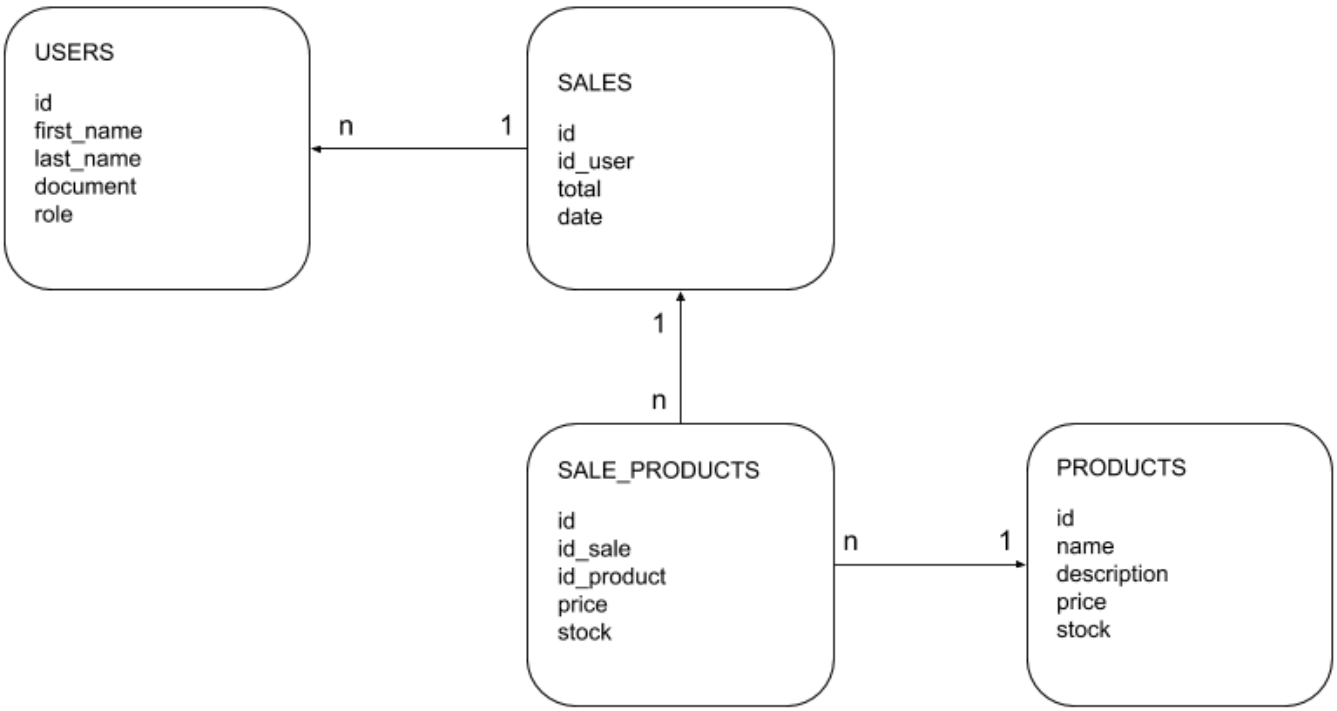
Base de datos: se utilizará el modelo master-slave en el que cada vez que se quiera agregar, modificar o eliminar una fila se replicará dicha acción en el slave (réplica). Todas las solicitudes de obtención de información se harán directo al slave para evitar sobrecargar al master.

Cache: esta capa será la encargada de reducir notablemente la cantidad de solicitudes de lecturas a la réplica de la base de datos cacheando aquellas filas que ya se hayan pedido anteriormente

Microservicios - flujos normales

El *sales-gateway* será el punto de entrada a nuestro dominio. Al redireccionar los requests a los microservicios correspondientes, deberá tener una conexión viva con cada uno de ellos. Internamente el microservicio de *sales-manager* tendrá comunicación con *users-manager* y con *products-manager* para poder obtener ambas entidades a la hora de construir el objeto de una venta. Los tres managers tendrán conexión con la base de datos y con la capa de caché.

Diagrama de Base de Datos



Infraestructura

Nuestro proveedor principal de servicios en la nube será **Amazon**.

Para evitar que todo el sistema sea afectado si una máquina virtual deja de funcionar y podamos asegurar que la aplicación se mantenga performante en todo momento utilizaremos 1 EC2 (Elastic Compute Cloud) por cada microservicio que tengamos.

A su vez, para permitir que el sistema escale ante un incremento en el tráfico que recibamos utilizaremos el EC2 Auto Scaling. Amazon ofrece con este servicio la posibilidad de escalar dinámicamente sin la necesidad de intervención del administrador de la infraestructura y lo hace tanto para arriba como para abajo. Establecimos los siguientes **mínimos de instancias** para cada microservicio:

sales-gateway -> 5 instancias. Al ser el que soportará toda la carga inicial de los requests, necesitamos asegurar que no se degrade a medida que los mismos aumenten.
users-manager -> 2 instancias. Sólo validará al usuario y tendrá la administración de los mismos, por lo que no necesitará muchas instancias.
sales-manager -> 3 instancias. Este microservicio administrará las ventas así como también calculará las comisiones. Esta última representa el requerimiento principal del sistema por lo que decidimos agregar una instancia más para soportar dicha carga.
products-manager -> 2 instancias. Al igual que el microservicio de usuarios, sólo tendrá la administración de los productos por lo que no necesitará muchas instancias.

Para las dependencias complementarias, **Amazon** provee servicios particulares. En lo que respecta a la caché utilizaremos **Amazon ElastiCache** mientras que para la base de datos trabajaremos con **Amazon RDS** (Relational Database Services).

Por último en materia de deployments utilizaremos los GitHub Actions para deployar automáticamente a producción (una vez que el Pull Request sea aprobado y mergeado). Administraremos con Spinnaker las instancias productivas deployando con Kubernetes.

Si los costos no lo permiten, se utilizará Docker para la implementación y puesta en producción del sistema pero en una máquina propia.

Tecnologías

NestJS: es un framework de NodeJS que sigue creciendo día a día y toma más presencia en el ámbito informático. Entre las funcionalidades que tiene, las que más se destacan son:

- Sistema de módulos basados en Angular
- Abstracción de dependencias externas mediante interfaz propia

A su vez permite la utilización de Typescript lo cual es bastante moderno y más prolijo que el uso de Javascript puro.

Link: https://nestjs.com/

REST: utilizaremos HTTP como protocolo y endpoints REST tanto para la comunicación del cliente con nuestro sistema, como para la comunicación interna entre los microservicios.

Typeorm: será el encargado de manejar la conexión y requests a la base de datos. Viene integrado con NestJS y provee muchas generalizaciones y optimizaciones. Aunque en este caso no sea requerido, también sirve de interfaz para comunicarse con distinto tipos de base de datos (como las no relacionales).

Link: https://typeorm.io/

MySQL: dado que las entidades que vamos a manejar van a estar fuertemente relacionadas entre ellas, utilizaremos una base de datos relacional. Se optó por MySQL debido a que consideramos que es la mejor base de datos de licencia libre, debido a su popularidad, eficacia y fácil implementación en ambientes web.

Redis: es una de las librerías más conocida y utilizada para cachear información y tiene implementaciones en varios lenguajes.

Link: https://redis.io/

Plan de testeo

Dado que trabajaremos con Typescript optamos por utilizar la librería **Jest** para la realización de unit testing. Apuntaremos a tener un **90% de coverage** en cada uno de nuestros microservicios.

Por sobre eso hemos reservado una cantidad importante de horas para el testeo manual del sistema y para la segunda versión planificamos implementar **Selenium** para realizar test automatizados integrados.

Por último en lo que respecta a ambientes de desarrollo contaremos con:

- Ambiente local
- Ambiente staging
- Ambiente producción

Endpoints

Método	Endpoint	Body
GET	/users/:id	N/A
POST	/users	{ email, firstName, lastName, document, role }
PUT	/users	{ id, email, firstName, lastName, document, role }
DELETE	/users/:id	N/A
GET	/sales/:id	N/A
GET	/sales/user/:id?from_date="2020-09-09"&to_date="2020-09-09"	N/A
GET	/sales/product/:id?from_date="2020-09-09"&to_date="2020-09-09"	N/A
GET	/sales?from_date="2020-09-09"&to_date="2020-09-09"	N/A
POST	/sales	{ products: [{ id, price }], userId, date, totalPrice }
PUT	/sales	{ id, products: [{ id, price }], userId, date, totalPrice }
DELETE	/sales/:id	N/A
GET	/products/:id	N/A
POST	/products	{ name, description, price }
PUT	/products	{ id, name, description, price }
DELETE	/products/:id	N/A
GET	/sales/user/:id/comissions?from_date="2020-09-09"&to_date="2020-09-09"	N/A