ELVI MIHAI SABAU SABAU - 51254875L

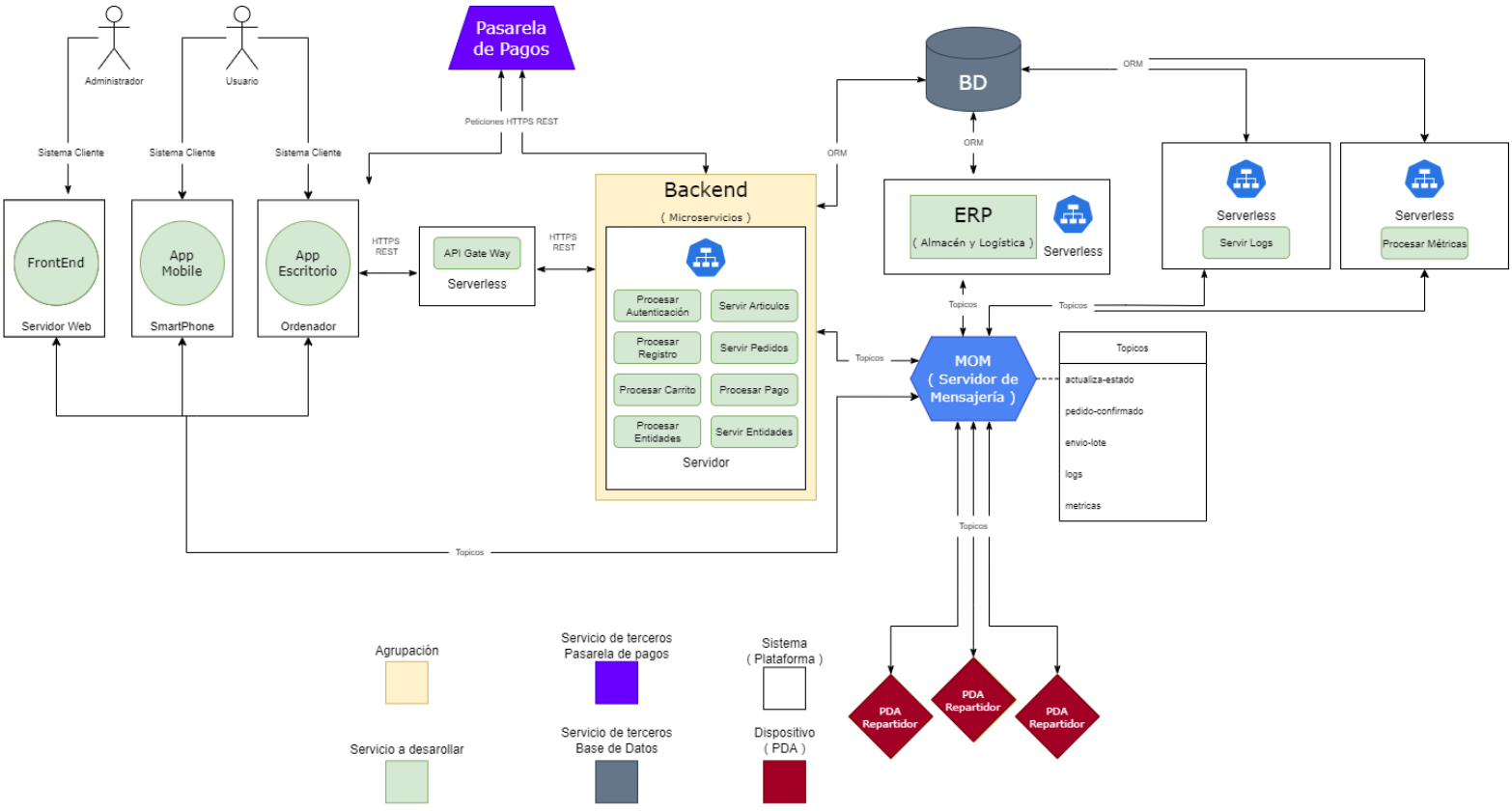
Distributed Systems S.L.

Proyecto:
Super Sneakers

Esquema Estructural

A continuación mostraremos el esquema estructural del sistema informático.

Distributed Systems S.L. Super Sneakers



ELVI MIHAI SABAU SABAU - 51254875L

Definición Estructural

Especificación de tecnologías.

Se usará GitHub con Circle y DockerHub con la opción de autobild para un desarrollo ágil y despliegue multiplataforma.

El tipo de bases de datos que se usará será PostgreSQL, ya que nos proporciona ciertas funcionalidades extra que otras bases de datos relacionales, como la inserción / actualización en masa, o soportes para enumeradores.

La tecnología que se usará estará basada en el lenguaje JavaScript, se ha seleccionado este lenguaje debido al dinamismo que proporciona ciertos frameworks basados en este lenguaje.

Aún así, no usaremos directamente JavaScript, usaremos TypeScript, un supertipo de JavaScript, que nos permitirá escribir código de manera segura, y fuertemente tipado, que transpila a JavaScript.

Ya que, como veremos más adelante, podremos reutilizar hasta un 90% del código escrito en un tipo de frontend como en otro.

El entorno de desarrollo y ejecución será Node.

El framework que se usará para el Backend será Expressjs, este framework permite servir conexiones de tipo peticiones http usando el protocolo REST. Este framework procesa las peticiones entrantes de manera asíncrona en su totalidad.

El código en nuestro Backend estará conectado con la base de datos usando un ORM llamado Sequelize, que nos permitirá crear identidades llamadas modelos, que se pueden reutilizar en cualquier otra aplicación que use Sequelize, además de poder modificar o crear nuevos modelos de manera fácil.

El framework que se usará para el Frontend será Angular. Este framework nos permite programar aplicaciones web progresivas complejas de manera sencilla y muy escalable en caso de que en un futuro haya cambios en la aplicación.

Aún así, este framework es usado para crear aplicaciones web, pero usando tecnologías como el framework de Ionic o Electron, nos permite usar el 80% - 90% el mismo código en Angular para desarollar las aplicaciones moviles, o la aplicación de escritorio.

Esto se debe a que Ionic y Electron permiten usar el mismo codebase basado en JavaScript creado en cualquier framework para crear aplicaciones moviles, o la aplicación de escritorio.

Además, Ionic nos permite buildear la app para cualquier smartphone SO del mercado actualmente (para iPhones se requiere obligatoriamente un MacOS con CodeX para su compilamiento).

Usaremos NGINX como servidor web para servir la página web.

La manera de desplegar los servicios se hará mediante contenedores, en Kubernetes, usando imágenes de Docker. De esta manera, tendremos cada servicio encapsulado en contenedores, y desplegado usando Kubernetes, esto nos proporcionará una funcionalidad de balanceo de contenedores y nodos de tal manera que será automáticamente escalable.

Para el desarollo se usará docker-compose, con volúmenes mapeados a la carpeta del código a desarollar, de esta manera, la creación de las imágenes de Docker será mucho más automática. Una vez acabado un módulo, solo se tendrá que generar la imágen, subirla al Cloud Host o VPS, y desplegar un contenedor usando dicha imágen.

El uso de Docker y docker-compose nos permitirá desplegar y desarrollar los servicios y apps en la mayoría de SO de uso común actualmente, tales como Windows, GNU/Linux & derivados y MacOS.

El tipo de Host que se usará para tener hosteados los servicios de la aplicación en internet será el Cloud Host de AWS (Amazon Web Services), usando el servicio de EKS (Amazon Elastic Kubernetes Service) para el servidor con microservicios, y para servicios individuales usaremos SAM (Serverless Application Model).

Se usará Google Analytics y Yanderex Metrica como inyectores en la página web y las aplicaciones, para compilar datos de uso y cookies, de esta manera se podrá servir sugerencias de productos o incluso publicidad adaptada al usuario.

Se usará la librería Morgan para las peticiones http entre servicios y Winston para los eventos internos de procesamiento de cada servicio para logear en la base de datos todos los eventos sucedidos entre los servicios.

Se usará Swagger para realizar la documentación de la API.

Se usará Axios para los servicios que ataquen al Backend, Axios es un cliente http que permite mandar solicitudes REST, y Expressjs es un framework para crear servidores que reciben y procesan peticiones REST.

Las peticiones / envios de información entre front y back serán encriptadas, para ello se realizarán usando HTTPS sobre la capa TLS, para ello compraremos un dominio supersneakers.com desde namecheap.com y le configuraremos un certificado.

Para hacer el mantenimiento del certificado más automático, usaremos CertBot con Teaefik.

Para la autenticación se usará Auth0, una plataforma de autenticación segura y dinámica que soporta JWT.

La pasarela de pagos se hará via Stripe, una plataforma de pagos electrónicos que automatiza los pagos, ofreciendo varios métodos de pago.

Y para implementar el servidor de mensajería, se podría usar Kafka.

Frontend y Backend:

El frontend son 3 partes, dependiendo desde que dispositivo se acceda.

La página web que funciona con nginx usando Angular, La aplicación móvil usando ionic y la aplicación de escritorio usando Electron.

Todas las conexiones con el BackEnd o la pasarela serán asíncronas, usando el protocolo REST mandando peticiones HTTP usando el cliente Axios.

El FrontEnd enviará los datos al BackEnd, a un microservicio en concreto, esta transferencia se realiza mediante una petición HTTP mandando los datos en formato JSON en el cuerpo de dicha petición, a un endpoint del BackEnd especificado con una URI y un método en concreto (GET, POST, PUT, DELETE).

Una vez el BackEnd procese los datos, le responderá a dicha petición con una respuesta tambien en formato JSON, en el cuerpo de la petición y un código de estado.

Por cada funcionalidad se realizarán una serie de acciones:

Register.

La pantalla de registro recibirá las credenciales del usuario, y las enviará al BackEnd para que este las guarde en la base de datos.

Datos sensitivos tales como contraseñas serán hasheadas previamente al guardado.

- Endpoint en el BackEnd:
  - Service: `Procesar Registro`.
    - URI: `POST - supersneakers.com/api/v1/auth/register`

Login.

La pantalla de lógin recibirá las credenciales del usuario, y las enviará al BackEnd para que este ataque a la base de datos y verifique si las credenciales son correctas, se usará de

intermediario Auth0 para permitir el inicio de sesión usando cuentas de Google, Facebook, y otros.

Las sesiones se mantendrán usando JWT, cada vez que el usuario solicite un dato al backend (da igual si es ver pedidos, como si es filtrar artículos en el frontend), el backend le mandará un JWT refrescado con un tiempo limitado de 5min, cosa que significa que si el usuario no cambia de página en 5 minutos, su sesión habrá caducado.

Al iniciar sesión se comparará la contraseña recibida hasheada con la contraseña previamente hasheada en la base de datos.

- Endpoint en el BackEnd:
    - Service: `Procesar Autenticación.`
        - URI: `POST - supersneakers.com/api/v1/auth/login`

Cesta de la Compra.

El usuario podrá añadir, editar y eliminar productos a la cesta. Los datos de la cesta serán analizados por el inyector de analíticas preferido (Google analitycs y/o Yandex Metrica). Los datos de la cesta estarán localmente guardados:

- En caso del navegador o ordenadores, se guardará en LocalStorage.
- En caso de smartphones, se guardará en Storage.

Al realizar el pedido, antes de pagar, se mostrará un estimado del precio total de los productos, y los productos a comprar, para ello se confirmará el stock y validará los precios mandando los artículos del carrito al BackEnd.

- Endpoint en el BackEnd:
    - Service: `Procesar Métricas.`
        - Tópico del MOM: `métricas`
    - Service: `Procesar Carrito.`
        - URI: `POST - supersneakers.com/api/v1/order`

Listar articulos.

Los usuarios, una vez hayan iniciado sesión, podrán ver una lista de artículos, adaptada a sus preferencias, esto lo hacemos gracias a las métricas que iremos recogiendo de los usuarios mientras van usando la tienda.

Además, en la misma pantalla, se podrá filtrar los artículos mostrados por tipos o marcas, de dichos artículos, y ver cuántos artículos hay por cada tipo.

- Endpoint en el BackEnd:
  - Service: `Procesar Métricas`.
    - Tópico del MOM: `métricas`
  - Service: `Procesar Articulos`.
    - URI: `GET - supersneakers.com/api/v1/item?type=”&brand=”`
    - Note: Dependiendo del valor del parámetro de la URI, se filtrará los artículos a mostrar por tipo (`type`) o marca (`brand`).

Realizar compra.

A la hora de realizar la transacción, el FrontEnd tramitará el pago conectando con Stripe y una vez realizado mandará al servidor de mensajería los productos que se han comprado, y la ID del pago. El backend recibirá dichos datos desde el servidor de mensajería, y guardará los datos en la base de datos. Por otro lado, el ERP recibirá dichos datos también, y procederán a realizar el trámite del pedido.

- Endpoint en el BackEnd:
  - Service: `Procesar Métricas`.
    - Tópico del MOM: `métricas`
  - Service: `Procesar Pago`.
    - URI: `POST - supersneakers.com/api/v1/transact/order`
- Endpoint en Stripe:
  - Service (Stripe): `Charges`.
    - URI: `POST - stripe.com/api/v1/charges`
    - Note: Se usa esta API contra Stripe para emitir el cargo.
  - Service (Stripe): `Charges`.
    - URI: `GET - stripe.com/api/v1/charges`
    - Note: Se usa esta API contra Stripe para confirmar el cargo.
- Tópicos en el servidor de mensajeria:
  - `pedido-confirmado`: Contiene un array de productos con sus IDs, y la ID de confirmación del pago.

Panel de administración.

Los usuarios con Rol administrador podrán acceder a una pantalla de administración, que les permitirá procesar todas las entidades, incluidas las que no están disponibles para el usuario final, para ello hay unos microservicios que sirven de gestores para todos los modelos del ORM de Sequelize, un microservicio capaz de gestionar cualquier modelo usando programación reflexiva.

- Endpoint en el BackEnd:
  - Service: `Procesar Entidades & Servir Entidades.`
    - URI: `GET  - supersneakers.com/api/v1/metrics/overseer`
    - URI: `GET  - supersneakers.com/api/v1/metrics/overseer/:id`
    - URI: `POST - supersneakers.com/api/v1/metrics/overseer/`
    - URI: `PUT  - supersneakers.com/api/v1/metrics/overseer/:id`
    - URI: `DEL  - supersneakers.com/api/v1/metrics/overseer/:id`

    - URI: `GET  - supersneakers.com/api/v1/transact/order/`
    - URI: `GET  - supersneakers.com/api/v1/transact/order/:id`
    - URI: `POST - supersneakers.com/api/v1/transact/order/:id`
    - URI: `PUT  - supersneakers.com/api/v1/transact/order/:id`
    - URI: `DEL  - supersneakers.com/api/v1/transact/order/:id`

    - URI: `GET  - supersneakers.com/api/v1/item?type=”&brand=”`
    - URI: `GET  - supersneakers.com/api/v1/item/:id`
    - URI: `DEL  - supersneakers.com/api/v1/item/:id`
    - URI: `PUT  - supersneakers.com/api/v1/item/:id`
    - URI: `POST - supersneakers.com/api/v1/item/`

    - URI: `GET  - supersneakers.com/api/v1/user/`
    - URI: `GET  - supersneakers.com/api/v1/user/:id`

        - URI: `DEL - supersneakers.com/api/v1/user/:id`
        - URI: `PUT - supersneakers.com/api/v1/user/:id`
        - URI: `POST - supersneakers.com/api/v1/user/`
        - Tópico en el MOM: `logs`

- Endpoint en Stripe:
    - Service (Stripe): `Charges`.
        - URI: `POST - stripe.com/api/v1/charges`
        - URI: `GET - stripe.com/api/v1/charges`
        - URI: `GET - stripe.com/api/v1/charges/:id`

    - Service (Stripe): `Refunds`.
        - URI: `POST - stripe.com/api/v1/refunds`
        - URI: `GET - stripe.com/api/v1/refunds`
        - URI: `GET - stripe.com/api/v1/refunds/:id`

ERP

El ERP se encarga de gestionar el almacén y el servicio de logística, revisar si hay stock para cierto artículo, y mantener un estado en vivo del estado del almacén.

Se comunica con la base de datos, actualizando el stock de los productos, y el estado de los pedidos.

El ERP es un microservicio serverless, funcionando en AWS SAM.

Recibe datos del backend sobre qué artículos se han comprado.

También se comunica con las PDAs de los repartidores, para mandarles que lotes hay que enviar, y cada artículo a que dirección va.

La comunicación que realiza el ERP con la base de datos se hace usando Sequelize, y la comunicación con el BackEnd y los Repartidores, se realiza usando un servidor de mensajería.

- Tópicos en el servidor de mensajería:
    - `pedido-confirmado`: Contiene un array de productos con sus IDs, y la ID de confirmación del pago.
    - `envio-lote`: Contiene la ID de lote, y el array de productos a repartir.
    - `actualiza-estado`: Contiene la ID del producto a actualizar su estado de envío (posición actual que se usará en el sistema para dar una estimación del tiempo restante).
    - `envio-confirmado`: Contiene la ID del producto que confirma que su envío ha sido satisfactorio, confirmando el DNI del receptor.

Las PDAs de los repartidores tendrán una aplicación, dependiendo de la empresa encargada de la logística (Seur, FedEX, UPS), que recibirá el mensaje del servidor de mensajería, siguiendo el formato adecuado especificado por la empresa repartidora.

API GateWay

La API Gateway hará de intermediario para servir los endpoints de todos los microservicios que sirve el backend.

Traza.

A continuación se explicará una traza de compra de un artículo.

1. El usuario accede a la página web, usando cualquier app del frontend (su app móvil, o la pag web, o la app instalada en su portátil).

2. Se autentica en el sistema (Registra si no tiene cuenta, y después se loguea).

3. Al autenticarse, se muestra la tienda.

4. El usuario añade a la cesta un producto.

5. El usuario procede a la compra de los artículos, el sistema verifica el stock y reserva dichos productos temporalmente.

6. El usuario procede al pago, el frontend se comunica con la pasarela de pagos y se muestra una interfaz de pago.

7. Una vez confirmado el pago, el frontend se comunica con el back end, y este confirma la ID del pago y del pedido contra la pasarela de pagos, guarda el pedido en la base de datos y emite dicho pedido al servidor de mensajería, que llega al ERP.

8. El frontend confirma dicho pedido al usuario, y guarda su estado en sus pedidos.

9. Los trabajadores del almacén reciben dicho pedido, y lo añaden al lote a enviar en el próximo envío.

10. El repartidor llega a su turno de trabajo, revisa su lote a repartir, lo recoge y procede a su repartición.

11. A medida que se van repartiendo los bultos, el tiempo restante de entrega del pedido del usuario se va actualizando.

12. El repartidor llega a la dirección del usuario, y le entrega el artículo pedido, el repartidor le pide al usuario su número de DNI, y el repartidor confirma que el pedido ha sido entregado satisfactoriamente.