PROPUESTA TÉCNICA | GESTIÓN DE RECLAMOS CIUDADES DEL FUTURO

EQUIPO D DE YATAY A DE ORT ARGENTINA

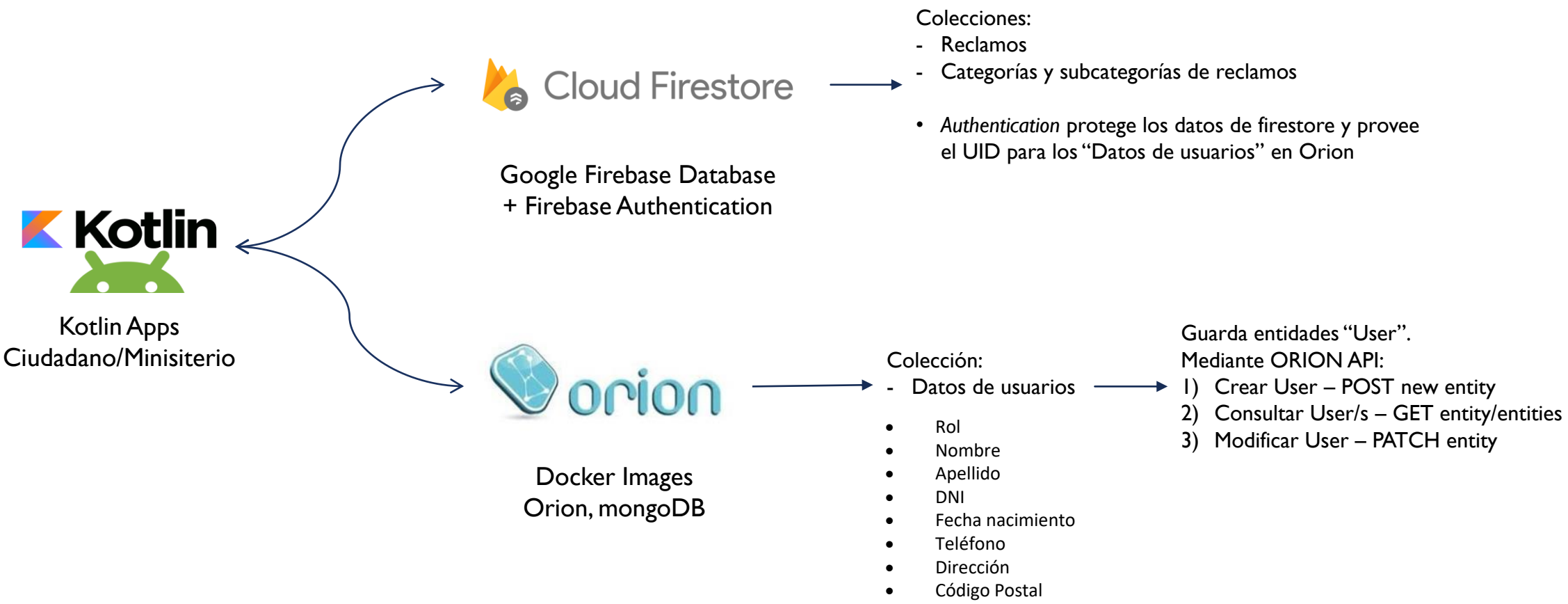
Autores:

- Arbio, Nicolás Gabriel
- Gao, Alan
- Martinez, Damián
- Perchuk, Federico
- Pisterman, Ariel
- Steinberg, Federico

TECNOLOGÍAS UTILIZADAS

- Kotlin en Android Studio para desarrollo Mobile de las 2 aplicaciones para Sistema Operativo Android
- Google Firebase
  - Firestore Database: base de datos no-SQL basada en documentos.
  - Authentication: proveerá seguridad a la aplicación y el UID de cada usuario.
- Fiware
  - Docker y DockerCompose: Para la configuración y ejecución de los componentes Fiware.
  - Orion Context Broker: Componente principal de Fiware para la administración de los datos del contexto (usuarios en este caso)

ARQUITECTURA



FIWARE – ORION CONTEXT BROKER

- Fiware es un ecosistema de **componentes interconectados**. impulsado por la Unión Europea, para el desarrollo y despliegue global de aplicaciones de Internet del Futuro. Para esto se provee una arquitectura abierta, pública y libre.
- Para que una solución tecnológica resulte "Powered by Fiware" o "Fiware Compliant" debe tener en su arquitectura el componente core de Fiware llamado **Orion Context Broker**.
- Este componente es el que utilizaremos para este proyecto para la administración de las entidades de tipo Usuario.
- OCB administra los datos del contexto en una base de datos MongoDB y los expone mediante una API desarrollada bajo el estándar **NGSI** (Next Generation Service Interfaces).

DOCKER Y DOCKER-COMPOSE

- Docker es un software que asiste el despliegue de aplicaciones dentro de contenedores, abstrayendo los recursos necesitados gracias a la virtualización en los diferentes sistemas operativos. Permite crear, ejecutar y manipular imágenes de diferentes productos, los cuales en conjunto proveerán diferentes servicios.
- Docker-Compose permite crear un contenedor que alberga varios servicios. Mediante un archivo YAML se describen las imágenes requeridas y las configuraciones pertinentes a cada una. Se puede configurar la red interna, crear volúmenes, configurar comandos por cada imagen y crear dependencias entre imágenes, entre otros.

DOCKER COMPOSE – ORION & MONGODB

docker-compose.yml

```yaml
version: "3.5"
services:
  mongo:
    image: mongo:4.4
    hostname: mongo
    container_name: mongo
    command: --nojournal
    expose:
      - "27017"
    ports:
      - "27017:27017"
    networks:
      - default
    volumes:
      - mongo-db:/data

  orion:
    image: fiware/orion
    hostname: orion
    container_name: fiware-orion
    depends_on:
      - mongo
    networks:
      - default
    ports:
      - "1026:1026"
    command: -dbhost mongo

networks:
  default:
    ipam:
      config:
        - subnet: 172.18.1.0/24

volumes:
  mongo-db: ~
```

CONTEXT BROKER – EJEMPLO ENTIDAD USUARIO


```
{
    "id": "Vyw3MLQs1Ue3b9wiFBkQxd9X8BD2",
    "type": "Usuario",
    "apellido": {
        "type": "Text",
        "value": "Administrador",
        "metadata": {}
    },
    "codigoPostal": {
        "type": "Text",
        "value": "B1900ASW",
        "metadata": {}
    },
    "direccion": {
        "type": "Text",
        "value": "Calle 50 Nº575 1º Piso of 107 - La Plata, Buenos Aires, Argentina",
        "metadata": {}
    },
    "dni": {
        "type": "Text",
        "value": "12.345.789",
        "metadata": {}
    },
    "email": {
        "type": "Text",
        "value": "admin@cdf.com",
        "metadata": {}
    },
    "fechaDeNacimiento": {
        "type": "Text",
        "value": "",
        "metadata": {}
    },
    "fotoPerfil": {
        "type": "Text",
        "value": "",
        "metadata": {}
    },
    "isEnabled": {
        "type": "Text",
        "value": "enabled",
        "metadata": {}
    },
    "nombre": {
        "type": "Text",
        "value": "Usuario",
        "metadata": {}
    },
    "rol": {
        "type": "Text",
        "value": "Admin",
        "metadata": {}
    },
    "telefono": {
        "type": "Text",
        "value": "+54 9 11 9876 5432",
        "metadata": {}
    }
}
```



```
{
    "id": "Vyw3MLQs1Ue3b9wiFBkQxd9X8BD2",
    "type": "Usuario",
    "apellido": "Administrador",
    "codigoPostal": "B1900ASW",
    "direccion": "Calle 50 Nº575 1º Piso of 107 - La Plata, Buenos Aires, Argentina",
    "dni": "12.345.789",
    "email": "admin@cdf.com",
    "fechaDeNacimiento": "",
    "fotoPerfil": "",
    "isEnabled": "enabled",
    "nombre": "Usuario",
    "rol": "Admin",
    "telefono": "+54 9 11 9876 5432"
}
```

APLICACIÓN AL PROYECTO DE GESTIÓN DE RECLAMOS

Instalación y configuración de Docker

En Windows:

1. Descargar e instalar "Docker Desktop for Windows": https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe
2. Durante la instalación dar los permisos e instalar o configurar las dependencias solicitadas.

En Ubuntu:

1. Instalar Docker Engine: https://docs.docker.com/engine/install/ubuntu/
2. Instalar Docker Compose: https://docs.docker.com/compose/install/

Luego:

3. Crear un directorio fiware y dentro descargar el archivo docker-compose.yml (Ejemplo: "C:\fiware\docker-compose.yml)
4. Abrir la consola y desde ese directorio creado ejecutar "docker-compose up -d"

POSTMAN

- Postman es una aplicación que nos permite realizar pruebas API. Es un cliente HTTP que nos da la posibilidad de testear 'HTTP requests' a través de una interfaz gráfica de usuario, por medio de la cual obtendremos diferentes tipos de respuesta que posteriormente deberán ser validados.
- Este producto se utilizó para la confección de los endpoints necesitados por las aplicaciones
- También se utiliza para crear usuarios de Administrador

KOTLIN: RETROFIT Y MOSHI

- Para integrar a Kotlin el servicio de Orion provisto por Docker, se requiere un cliente HTTP como Retrofit el cual recibirá en texto la respuesta del servicio. Para entender esta respusta y poder convertirla en un objeto de Kotlin se utilizó Moshi, que es un conversor de JSON a objeto Kotlin.

KOTLIN: RETROFIT Y MOSHI

- Para su uso, se definió un objeto OrionApi el cual construye un atributo retrofitService a partir de un objeto Retrofit (que a su vez precisa un objeto Moshi y la URL base del servicio a consumir). Este objeto OrionApi implementa una interfaz de Kotlin donde se declaran los metodos que estarán estrechamente conectados (mediante una notación) con los endpoints necesitados.

```
private const val BASE_URL = "http://${IP}:1026/v2/"
private val moshi = Moshi.Builder()
    .add(KotlinJsonAdapterFactory())
    .build()
private val retrofit = Retrofit.Builder()
    .addConverterFactory(MoshiConverterFactory.create(moshi))
    .baseUrl(BASE_URL)
    .build()

interface OrionApiService {
    @GET("/version")
    suspend fun verificarConexion(): Response<Unit>

    @GET("entities/{id}?options=keyValues&q=isEnabled:"+OrionApi.USER_ENABLED)
    suspend fun getUsuarioByUID(@Path("id") UID: String): Usuario

    @GET("entities?options=keyValues&type=Usuario")
    suspend fun getUsuarioByQuery(@Query("q") query: String): List<Usuario>

    @POST("entities?options=keyValues")
    suspend fun registrarUsuario(@Body usuario: Usuario)
}

object OrionApi {
    val retrofitService : OrionApiService by lazy {
        retrofit.create(OrionApiService::class.java)
    }
}
```