Introducción

En el marco del desafío técnico propuesto por MercadoLibre, se desarrolló utilizando como tecnologías base **Python** y **Flask**, una **API** que actúa como *Middleware* entre el usuario y la ya existente PokéAPI. Esta aplicación contiene la lógica para, a través de **distintas rutas** o **paths**, obtener información de los distintos Pokemon utilizando como base las funcionalidades descritas en el challenge.
Durante el desarrollo de este documento se describe el comportamiento y las posibles respuestas de cada uno de los endpoints propuestos. A su vez, la aplicación cuenta con un módulo de autenticación que mantiene seguros los endpoints que devuelven información.

Repositorio Git

Durante el desarrollo de la aplicación se utilizó un repositorio Git para el control de versiones. Este repositorio contiene dos ramas o branches:

1. **Test:** utilizada para desarrollo local sobre versiones estables.
2. **Main:** versiones estables a partir de test.

El repositorio cuenta con un archivo requirements para instalar dependencias, sin embargo, es también necesaria la creación de un archivo **.env** dentro del directorio **/app/config** para que el módulo de autenticación integrado con Okta tenga acceso a las variables de entorno necesarias y funcione correctamente.

Utilización de Docker y repositorio DockerHub

Una vez alcanzada la versión 1.0 del desarrollo se utilizó **Docker** para crear una imágen que contenga tanto el código como las dependencias de la aplicación. La misma puede ser obtenida desde el siguiente repositorio DockerHub.
Al igual que con su versión en Git, es necesaria la creación de un archivo **.env** dentro del directorio **/app/config** para que el módulo de autenticación integrado con Okta tenga acceso a las variables de entorno necesarias y funcione correctamente.

Hosting de la aplicación con Google Cloud

Adicionalmente, con el objetivo de proporcionar una **versión hosteada en un dominio público** que pueda ser utilizada desde cualquier equipo, se optó por utilizar el contenedor desarrollado en Docker con la imágen de la aplicación para **crear un servicio** en **Google Cloud.**
Además de mantener a la aplicación prendida como un servicio, se utilizó el módulo de **Secret Manager** de la solución, permitiendo así, **alojar de manera segura** las **variables** de entorno **sensibles** que en el proyecto original se encuentran en el archivo **.env** dentro del directorio **/app/config**.

Integración con Okta - Módulo de autenticación

Dentro del código de la aplicación se encuentra un servicio que funciona como módulo de autenticación con Okta a través del protocolo OIDC con grant type **'Resource Owner'**.

¿Por qué Resource Owner?

En Okta, las integraciones OIDC son de varios tipos dependiendo del tipo de app que se busque integrar. El flujo de authorization code es típicamente utilizado por los primeros dos:

1. **Web Application:** aplicaciones Web que muestran un front-end y que realizan la autenticación y manejo de tokens del lado del backend.
2. **Single Page Application:** aplicaciones que muestran un front-end y que realizan la autenticación y manejo de tokens desde el navegador.
3. **Native Application:** aplicaciones desktop o mobile que manejan las solicitudes de autenticación y tokens desde un backend y, que además, redirigen a los usuarios a un callback no-HTTP.

Al **no contar con un front-end** y tampoco ser solicitado en el challenge, la solicitud de autenticación por parte de un flujo de Authorization Code no iba a ser posible y se dirigieron los esfuerzos en utilizar la opción de Native Application, que además permitía el flujo de Resource Owner para que la app pueda solicitar la autenticación de un usuario existente en el directorio de Okta de manera remota desde un backend.

Existe un **cuarto tipo de aplicación**: **API Services Application**, pero la misma está diseñada para comunicación **Machine-to-Machine** y al finalizar expide un token de acceso con permisos sobre la misma instancia de Okta. Se utiliza para gestionar el ambiente de Okta desde su conjunto de APIs y no para autenticar usuarios en aplicaciones externas.

Internamente la aplicación realiza **dos llamados**:

1. Al endpoint **/token** del ambiente de Okta. Este llamado se realiza al momento de solicitar un token de acceso para utilizar los endpoints de la aplicación. La solicitud contiene los siguientes parámetros:

URL relativa	**Método**
/oauth2/default/v1/token	POST
Parámetro	**Valor**
username	Atributo 'login' del usuario en Okta.
password	Contraseña del usuario en Okta.
client_id	Client ID de la definición OIDC de la app configurada en Okta.
client_secret	Client Secret de la definición OIDC de la app configurada en Okta.
scope	'openid'
grant_type	'password'

2. Al endpoint **/introspect** del ambiente de Okta. Este llamado se realiza en cada una de las solicitudes a los endpoints con el fin de verificar la integridad y la validez del token proporcionado. La solicitud contiene los siguientes parámetros:

URL relativa	**Método**
/oauth2/default/v1/introspect	POST
Parámetro	**Valor**
token	Token obtenido en el endpoint /obtener-ficha
token_type_hint	'access_token'
client_id	Client ID de la definición OIDC de la app configurada en Okta.
client_secret	Client Secret de la definición OIDC de la app configurada en Okta.

En la **versión local de la aplicación**, los valores de client_id y client_secret, así como el dominio de Okta se encuentran en el archivo **.env** del directorio **/app/config**. En su **versión como servicio en Google Cloud**, estas variables se alojan en el módulo de **Secret Manager** de la solución para ser invocadas durante estas dos solicitudes.

Colecciones de Postman - Pokedex

Para poder comenzar con la utilización de la API se disponibilizan dos colecciones Postman, una para sus versiones locales (obtenibles desde Git y Docker) y otra para su versión como servicio (hosteada en Google Cloud). Las mismas pueden ser encontradas abajo:

Versión Local	**Versión Cloud**
Descarga	Descarga

Construcción local de la aplicación - Pokedex

A continuación se detallarán los pasos a seguir para obtener una versión ejecutable del proyecto de manera local, dependiendo del repositorio elegido.

Repositorio Git

Desde una consola con acceso a comandos git (ej: GitBash), introducir los siguientes comandos:

```
#Obtiene los archivos del repositorio
git clone https://github.com/FranBillan/Pokedex-ML-Challenge.git

#Instala dependencias especificadas en el archivo requirements
pip install -r requirements.txt
```

Luego, dentro del directorio **/app/config** se deberá crear un archivo **.env** con el siguiente contenido:

```
OKTA_DOMAIN=dev.xxxxxxxxxxxx.okta.com
OKTA_CLIENT_ID=0oaxxxxxxxo5d7
OKTA_CLIENT_SECRET=6llbxxxxxxxxxxxxxxxxxxxxobRtbfi
```

Estos valores dependen del ambiente de Okta en el que se configure la integración. En caso de querer utilizar el ambiente y definición OIDC bajo la que se desarrolló la aplicación, se otorgarán dichos valores de manera sincrónica al personal de MercadoLibre encargado de evaluar el challenge.

```
#Luego de la correcta creación del archivo, desde el directorio del repositorio se ejecuta
python run.py
```

Repositorio DockerHub

Desde una consola con acceso a comandos Docker, introducir los siguientes comandos:

```
#Obtiene la imágen del repositorio Docker-Hub
docker pull frbillan/pokedex-api
```

Luego, dentro del directorio donde se encuentre el usuario, se deberá crear un archivo **.env** con el siguiente contenido:

```
OKTA_DOMAIN=dev.xxxxxxxxxxxx.okta.com
OKTA_CLIENT_ID=0oaxxxxxxxo5d7
OKTA_CLIENT_SECRET=6llbxxxxxxxxxxxxxxxxxxxxobRtbfi
```

Estos valores dependen del ambiente de Okta en el que se configure la integración. En caso de querer utilizar el ambiente y definición OIDC bajo la que se desarrolló la aplicación, se otorgarán dichos valores de manera sincrónica al personal de MercadoLibre encargado de evaluar el challenge.

```
#Luego de la correcta creación del archivo, desde el mismo directorio se ejecuta
docker run --env-file ./.env -p 5000:5000 pokedex-api
```

Nota: *tal como se referencia en el código de docker run, el archivo .env debe crearse en el directorio donde se crea conveniente para poder ser obtenido en el build. En el comando de ejemplo se usa el directorio donde está parado el usuario en la consola, pero en caso de utilizarse otro, se especificará luego de "–env-file".*

Servicio en Google Cloud

Bajo esta opción, no es necesario obtener ni ejecutar la aplicación de manera local. Sólamente es necesario importar la colección de Postman correspondiente para su utilización.

Descripción de funcionalidades y Endpoints

Para la realización del challenge, el equipo de MercadoLibre solicitó que la aplicación cuente con las siguientes funcionalidades:

1. Obtener el tipo de un Pokémon (fuego, agua, tierra, aire, etc) según su nombre.
2. Obtener un Pokémon al azar de un tipo en específico.
3. Obtener el Pokémon con nombre más largo de cierto tipo.

La aplicación desarrollada cuenta con un total de ocho endpoints que buscan cumplir con las tres funcionalidades descritas por el desafío y que son documentadas en el desarrollo de esta sección.

PokeAPI - Status

Descripción: endpoint base que devuelve un mensaje de bienvenida. A su vez, le indica al usuario continuar a /pokedex para obtener instrucciones sobre las funcionalidades disponibles.

Versión	URL	Método
Local	http://localhost:5000	GET
Cloud	https://pokedex-api-656201522543.us-central1.run.app	GET
Parámetros	**Ejemplo**	
Header	N/A	
Body	N/A	

Respuesta esperada:

Status: ***200***

```
{
   "mensaje": "¡Pokedex en línea, bienvenido!",
   "consejo": "GET a /pokedex para ver todas las funciones disponibles."
}
```

PokeAPI - Pokedex

Descripción: endpoint que devuelve instrucciones sobre cómo consumir los distintos endpoints. A su vez, le indica al usuario que para consumir dichos endpoints es necesario que se autentique.

Versión	URL	Método
Local	http://localhost:5000/pokedex	GET
Cloud	https://pokedex-api-656201522543.us-central1.run.app/pokedex	GET
Parámetros	**Ejemplo**	
Header	N/A	
Body	N/A	

Respuesta esperada:

Status: ***200***


```
{
   "mensaje": "¡Pokedex en línea, bienvenido!",
   "consejo": "GET a /pokedex para ver todas las funciones disponibles."
}"mensaje": "¡Hola! A continuación, te detallo todas las funciones disponibles:",
   "funciones_disponibles": [
      {
         "endpoint": "/pokedex/<nombre>",
         "descripción": "¿Querés saber el tipo de un Pokemon? Dame su nombre.",
         "ejemplo": "/pokedex/serperior",
         "método": "GET"
      },
      {
         "endpoint": "/pokedex/types",
         "descripción": "¿No recordás todos los tipos? Te muestro una lista completa.",
         "ejemplo": "/pokedex/types",
         "método": "GET"
      } …. resto de los métodos
```


PokeAPI - Get Auth Token

Descripción: endpoint que envía las credenciales del usuario en Okta para obtener un token de acceso que permite la utilización de los endpoints.

Versión	URL	Método
Local	http://localhost:5000/obtener-ficha	POST
Cloud	https://pokedex-api-656201522543.us-central1.run.app/obtener-ficha	POST

Parámetros	Ejemplo
Header	Content-Type: application/json

Body	{
	"username": "username",
	"password": "password"
	}

Respuestas esperadas:

Autenticación exitosa

Status: ***200***

```
{
   "access_token": "eyJraWQiOi…..U9g"
}
```

Si una o dos de las credenciales se envían vacías

Status: ***400***

```
{
   "error": "Las credenciales no pueden estar vacías",
   "sugerencia": "Tanto el username como el password deben contener caracteres válidos.",
   "ejemplo": [
      {
         "username": "<usuario>",
         "password": "<contraseña>"
      }
   ]
}
```

Si las credenciales son incorrectas

Status: ***401***

```
{
   "error": "No hay un entrenador asociado a esas credenciales en nuestros registros. Intentá de nuevo.",
   "sugerencia": "Verificá tu username y password."
}
```

PokeAPI - Get Pokemon by Name

Descripción: devuelve el tipo y las habilidades de un Pokemon especificado en la URL. También obtiene sus atributos.

Versión	URL	Método
Local	http://localhost:5000/pokedex/$pokemon_name	GET
Cloud	https://pokedex-api-656201522543.us-central1.run.app/pokedex/$pokemon_name	GET
Parámetros	**Ejemplo**	
Header	Authorization: Bearer *eyJraWQiOi…..U9g*	
Body	N/A	

Respuesta esperada:

Si el usuario se autentica y encuentra al Pokemon solicitado

Status: ***200***


```
{
   "mensaje": "¡Atrapaste a Serperior! A continuación, te presento su información:",
   "pokemon": {
      "nombre": "serperior",
      "tipos": [
         "grass"
      ],
      "altura": "3.3 mt",
      "peso": "63.0 kg",
      "número_pokedex": 497,
      "habilidades": [
         "overgrow",
         "contrary"
      ],
      "stats": {
         "hp": 75,
         "ataque": 75… resto de respuesta
}
```

Si el usuario se autentica y no encuentra al Pokemon solicitado

Status: ***404***

```
{
   "error": "¡Ups! No conozco ese Pokemon... ¿es uno de los nuevos?",
   "sugerencia": "Revisá que el nombre esté bien escrito."
}
```

PokeAPI - Get all Types

Descripción: esta función no fue solicitada en el desafío, sin embargo, me pareció útil incluirla para que un usuario que no conoce sobre Pokemon pueda identificar los tipos a utilizar en el resto de solicitudes.

Versión	URL	Método
Local	http://localhost:5000/pokedex/types	GET
Cloud	https://pokedex-api-656201522543.us-central1.run.app/pokedex/types	GET
Parámetros	**Ejemplo**	
Header	Authorization: Bearer *eyJraWQiOi…..U9g*	
Body	N/A	

Respuesta esperada:

Si el usuario se autentica y encuentra al Pokemon solicitado

Status: ***200***

```
{
   "mensaje": "¡Estos son todos los tipos de Pokemon disponibles!",
   "tipos": [
      "normal",
      "fighting",
      "flying",
      "poison",
      "ground",
      "rock",
      "bug",
      "ghost",
      "steel",
      "fire"… resto de respuesta
}
```

PokeAPI - Get Random Pokemon

Descripción: obtiene un Pokemon aleatorio entre todos los tipos disponibles.

Versión	URL	Método
Local	http://localhost:5000/pokedex/whos-that-pokemon/	GET
Cloud	https://pokedex-api-656201522543.us-central1.run.app/pokedex/whos-that-pokemon/	GET
Parámetros	**Ejemplo**	
Header	Authorization: Bearer *eyJraWQiOi…..U9g*	
Body	N/A	

Respuesta esperada:

Si el usuario se autentica y obtiene un Pokemon aleatorio

Status: ***200***


```
{
    "mensaje": "¡Un Pokemon salvaje apareció!",
    "pokemon": {
        "nombre": "arcanine",
        "tipos": [
            "fire"
        ],
        "altura": "1.9 mt",
        "peso": "155.0 kg",
        "número_pokedex": 59
    }
}
```

PokeAPI - Get Random Pokemon by Type

Descripción: obtiene un Pokemon aleatorio del tipo especificado.

Versión	URL	Método
Local	http://localhost:5000/pokedex/whos-that-pokemon/$type	GET
Cloud	https://pokedex-api-656201522543.us-central1.run.app/pokedex/whos-that-pokemon/$type	GET
Parámetros	**Ejemplo**	
Header	Authorization: Bearer *eyJraWQiOi…..U9g*	
Body	N/A	

Respuesta esperada:

Si el usuario se autentica y el tipo especificado es válido

Status: ***200***

```
{
    "mensaje": "¡Un Pokemon salvaje de tipo psychic apareció!",
    "pokemon": {
        "nombre": "cosmoem",
        "tipos": [
            "psychic"
        ],
        "altura": "0.1 mt",
        "peso": "999.9 kg",
        "número_pokedex": 790
    }
}
```

Si el usuario se autentica y el tipo especificado no es válido

Status: ***404***

```
{
    "error": "¡Ups! No conozco el tipo 'test'",
    "sugerencia": "Probá con tipos como 'fire', 'water', 'electric', etc."
}
```

PokeAPI - Get Longest Name by Type

Descripción: obtiene un Pokemon aleatorio del tipo especificado.

Versión	URL	Método
Local	http://localhost:5000/pokedex/longest/$type/	GET
Cloud	https://pokedex-api-656201522543.us-central1.run.app/pokedex/longest/$type/	GET
Parámetros	**Ejemplo**	
Header	Authorization: Bearer *eyJraWQiOi…..U9g*	
Body	N/A	

Respuesta esperada:

Si el usuario se autentica y el tipo especificado es válido

Status: ***200***

```
{
    "mensaje": "¡El Pokemon de tipo fairy con el nombre más largo es...",
    "pokemon": {
        "nombre": "mimikyu-totem-disguised",
        "tipos": [
            "ghost",
            "fairy"
        ],
        "longitud_nombre": "23 caracteres",
        "número_pokedex": 10144
    }
}
```

Si el usuario se autentica y el tipo especificado no es válido

Status: ***404***

```
{
    "error": "¡Ups! No conozco el tipo 'test'",
    "sugerencia": "Probá con tipos como 'fire', 'water', 'electric', etc."
}
```

Manejo de errores - respuestas genéricas

La aplicación presenta diferentes respuestas ante solicitudes HTTP/S fallidas que son comunes a todos los endpoints API. En esta sección se especificarán cuáles son esas respuestas y ante qué errores se presentan a los usuarios.

404 - Not found

Descripción: mensaje presentado cuando un usuario intenta interactuar con un endpoint no definido en la aplicación.
Respuesta esperada:

Status: ***404***

```
{
    "error": "Por acá no hay ningún Pokemon... ¿funciona bien tu Pokeradar?",
    "sugerencia": "Revisá bien la URL o consultá /pokedex para ver todas las funciones disponibles."
}
```

401 - Unauthorized

Descripción: mensaje presentado cuando un usuario intenta interactuar con un endpoint sin haber pasado por un proceso de autenticación o cuando su access token expiró.

Respuesta esperada:

Si omite la inclusión de un access token en la request

Status: ***401***

```
{
    "error": "Esta función está disponible solo para entrenadores autorizados. Presentá tu ficha de entrenador.",
    "sugerencia": "Debés incluir en el header 'Authorization: Bearer <token>' el código obtenido en /obtener-ficha"
}
```

Si el access token de la request es inválido o está inactivo

Status: ***401***

```
{
    "error": "Ficha inactiva... ¿Te expulsaron de la liga? Tal vez solo tengas que tramitar una nueva.",
```

```
    "sugerencia": "Obtené un nuevo token en /obtener-ficha"
}
```

500 - Internal Server Error

Descripción: error presentado al usuario cuando falla el código de la aplicación.

Status: ***500***

```
{
  "error": "¡Ups! El Pokémon se escapó...",
  "sugerencia": "¡Intentalo de nuevo!"
}
```

Descripción: error presentado al usuario cuando falla un componente externo (ej: PokeApi, Okta)

Status: ***500***

```
{
  "error": "¡Ups! Parece que hay problemas técnicos.",
  "sugerencia": "Intentalo de nuevo en unos momentos."
}
```