Université de Rennes 1

M2 Informatique - Ingénierie Logicielle (ILA)

Projet

Manuel technique et de reprise de code

Groupe C :

- *Abdel Karim HAMMAD*
- *Kevin MARQUER*
- *Kounadi OUATTARA*
- *Léo RAUZIER*
- *Shervin PEDARAN*
- *Soufiane DERMOUMI*

Encadrants :

- *Julien GUILLET*

Sommaire

1 - Introduction

L'objectif de ce manuel est de permettre la réutilisation du code du projet par d'autres équipes de développement en reprenant les stratégies de développement mises en place. Il comporte une explication technique du projet, l'organisation du projet et chacune des composantes logicielles.

2 - Overview

Il y a trois grandes parties pour ce projet :

- **flutter_app** pour l'application principale sur tablette
- **mavlinkapp** pour la création, l'envoie et de suivi de missions du drone
- **servicefirebaseapp** pour réception des données reçues par le drone et l'envoie sur firebase

Pour la réalisation du projet, plusieurs technologies ont été utilisées :

- **Flutter**, pour la réalisation de l'application

- **FireBase** comme backend et base de données

- **SpringBoot** pour les services REST

- **Mavsdk**, une bibliothèque Java pour la création et contrôle du drone

- **OpenStreetMap** comme map utilisée dans l'application

Le projet utilise aussi des bases de données externes :

- La base des adresses nationale
- La base des bornes incendie en France

Le projet se trouve sur GitHub où chaque dossier contient toutes les données nécessaires au lancement de l'application.

3 - Architecture globale

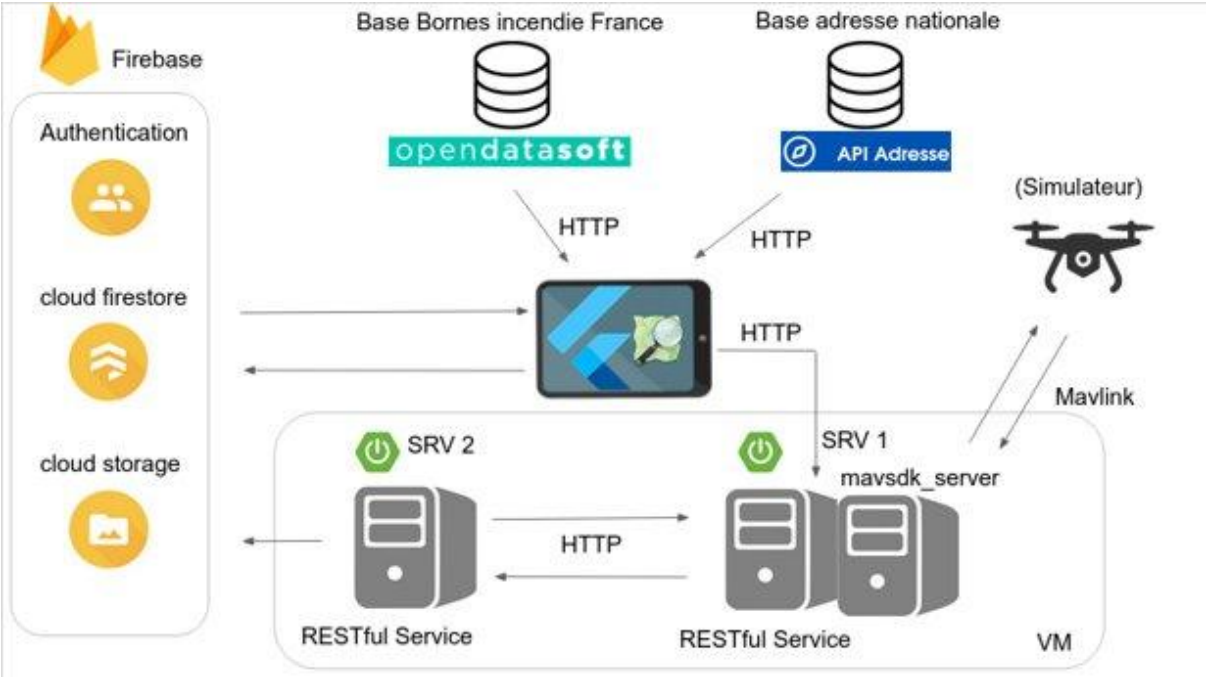


Figure 1 : Architecture technique globale

3.1 - Architecture flutter

3.1.1 - Icônes

Dans le dossier **flutter_app** se trouve un dossier **Icone_png** dans lequel est répertorié dans plusieurs sous-catégories toutes les icônes utilisées pour la carte de l'application.
Dans le dossier **Icone_png** se trouve des icônes qui n'ont pas de dossier propre soit parce qu'on à pris le temps de tout réorganisé dû au fait qu'il faut aussi prendre en compte de modifier le code pour changer le moment ou l'on appel cette icônes.
Les icônes impliquées sont :

- Icône du drone
- Icône des zone d'eau
- Poste de commandement

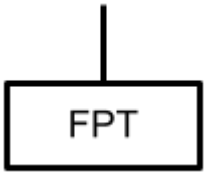
Les sous-dossiers de **Icone_png** sont :

- **Chemin** (représente les différentes lignes/flèches qui construit le chemin à prendre pour les infrastructures)
- **Danger**
  - Étoile représente le centre de l'action
  - Triangle pointe vers le haut représente un danger
  - Triangle pointe vers le bas représente un point sensible (une réserve de produit dangereux par exemple)

- **Défense** (représente la mise en place d'une barrière défensive pour protéger un point sensible, il y a deux types d'icônes
    - En ligne
    - Encerclé
- **Infrastructure**
    - Aérien (non utilisé)
    - Véhicules
        - Autre (non utilisé)
        - FPT
        - VLCG
        - VSAV
    - Colonne et groupe d'infrastructure (non utilisé)

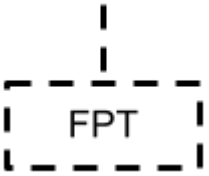
Les icônes possèdent pour la plupart deux états :

Encours



(actuellement sur les lieux)

Prévue



(en attente)

Il y a six couleurs utilisées pour correspondre à différents cas :

Basic **Incendie** **Danger** **Eau** **Civil**

Les icônes que l'on utilise ont un nom créé à partir de mots-clés afin de permettre de les utiliser à partir de paramètres dans l'application.
(S/M)_NomIcone_Couleur_(0/1).png

S correspond aux Symboles, **M** correspond aux Moyen (les infrastructures).
Les valeurs **0** et **1** correspondent à la valeur booléenne de l'état Encours (**1**) ou Prévue (**0**).

3.1.2 - Modèles

Différents modèles sont utilisées par l'application sur Flutter :

- **AddressSuggestion**
    - Class **Feature** (un String **type**, une **Geometry**, une **Properties**) correspond aux adresses suggérées

    - Class **Geometry** (un String **type**, Une liste de double **Coordinates**)
    - Class **Properties** (obtenue à partir d'un fichier JSON)
- **Drone** une classe contenant une class **Position**
- **HydrantData** une classe pour l'emplacement des différents point d'eau sur la carte
- **InterestPoint** une position pour un point de passage pour le drone avec possibilité de prendre une photo (un booléen photo)
- **Intervention** la classe comprenant l'adresse de l'action, son code Sinistre, des listes des moyens, de symboles et de missions et un drone.
- **Mission** classe concernant la mission du drone impliquant des point d'intérêt une liste de photos
- **Moyen** la classe d'un moyen comprenant son id, code couleur et d'une description
- **MoyenIntervention** classe correspondant à l'intervention d'un moyen se qui comprend une date de demande, de départ (demande validé), d'arrivée (arrivée sur les lieux) et de retour (quitte les lieux), ainsi que de sa position
- **PointDrone** similaire à **InterestPoint** mais utilise une longitude et latitude au lieu d'une position
- **Position** class comprenant une valeur longitude et latitude
- **Role** classe ayant l'une des deux valeurs pour l'utilisateur au moment de la connexion (**Intervenant** ou **Operator**)
- **Sinistre**
- **SymbolIntervention** classe servant aux icônes
- **UserData** classe correspondant à l'utilisateur de l'application (nom, mail, login …)

3.1.3 - Pages et Services

L'application comprend différentes pages et services dans trois dossiers :

- **Connection**
    - **SignInPage** la page de connexion
    - **SignUpPage** la page de création de compte pour l'application
- **Intervention**
    - **AddressSearch** page servant à la recherche d'adresse
    - **DronePage** la page du drone sur comprenant la liste des missions d'une intervention
    - **HomePage**
    - **MapPage** l'affichage de la carte sur l'onglet Sitac
    - **MissionFormPage** page inclue dans l'onglet Sitac servant à la création d'une mission d'un drone
    - **MissionPage** page de la mission après l'avoir créer ou sélectionner dans **DronePage**
    - **MissionPhotosPage** inclue dans **MissionPage** affichant les images prise et la vidéo en direct (vidéo prototype)
    - **NewInterventionPage** page servant à l'ajout d'un nouveau moyen

    - **SitacPage** page incluant MapPage ainsi que d'une légende des icônes et de **MissionFormPage**,
    - **UserPage** page donnant le profil de l'utilisateur actuel

- **Services** qui regroupe les méthodes utilisées dans les différentes pages
    - **AccountService**
    - **AdressService**
    - **DataBase**
    - **GeoLocatorService**
    - **HydrantService**
    - **InterventionService**
    - **MissionService**
    - **MoyenService**
    - **NavigatorPage**
    - **SelectorIntervention**
    - **SelectorMoyenSymbol**
    - **SelectorSitac**
    - **SelectorService**
    - **SinistreService**

3.2 - Communication

La communication entre les 3 applications du projet se fait à base de service REST et au travers de trigger sur la base de données. En effet les deux applications annexes mettent des endpoints à disposition des autres pour l'échange de données.

3.2.1 - Communication FlutterApp - mavlinkapp

La communication entre ces deux applications est unidirectionnelle (FlutterApp → mavlinkapp). L'application Flutter après la création d'une mission envoie les points d'intérêts de la mission à l'endpoint *sendMissionCoordonates(@Validated @RequestBody MissionDrone mission)* pour la réalisation d'une mission par le drône. Une copie des données de la mission est sauvegardée dans la base de données (Cloud Firestore).

3.2.2 - Communication mavlinkapp - servicefirebaseapp

Cette communication est unidirectionnelle également. Il s'agit purement d'une communication entre webservices. Le webservice mavlinkapp, une fois les données de la mission reçues (notamment la liste des points d'intérêts), lance la mission sur le drone. Une fois une nouvelle mission lancée sur le drone, son nouvel état est envoyé au webservice

servicefirebaseapp à l'endpoint setMissionState(String idMission, String state). Cette mise à jour de l'état initial est nécessaire dans le sens où le webservice de contrôle du drone utilise un ExecutorService pour exécuter, à un moment donné, une unique mission, les autres pouvant être reçues par le webservice mais sont mises dans une file d'attente. Pour une visualisation du déplacement du drone sur la SitacPage, les coordonnées du drone sont envoyées au webservice servicefirebaseapp à l'endpoint *updateDronePositionIntervention (String idIntervention, double latitude, double longitude)* et ce chaque 1.5 seconde (pour ne pas saturer le webservice cible).
Aussi, à l'arrivée du drone à un point d'intérêt, le drone prend une photo et celle-ci est envoyée au même webservice sous forme de tableau de byte à l'endpoint *uploadFileFromBytes(String idMission, double latitude, double longitude, byte[] bytes, String imagesOrVideos)*.

3.2.3 - Communication servicefirebaseapp - Cloud Firestore

L'ensemble des données reçues (état mission, positions drone, photos des points d'intérêt…) par le webservice servicefirebaseapp sont à destination immédiate du backend Cloud Firestore. Les mises à jour de ce backend entraînent par la même celle des principaux widgets de l'application Flutter (ces widgets étant StreamBuilder pour la plupart).

3.3 - Database

Firebase[1] firestore est utilisé comme base de données NoSQL pour la persistence des données de l'application et firebase cloud storage comme serveur de fichier pour stocker les images et vidéos du drone.
Pour prendre en main l'environnement firestore, suivre la documentation google sur : https://firebase.google.com/docs/firestore/quickstart.
Le modèle de données hébergé sur firestore comprend 5 collections:

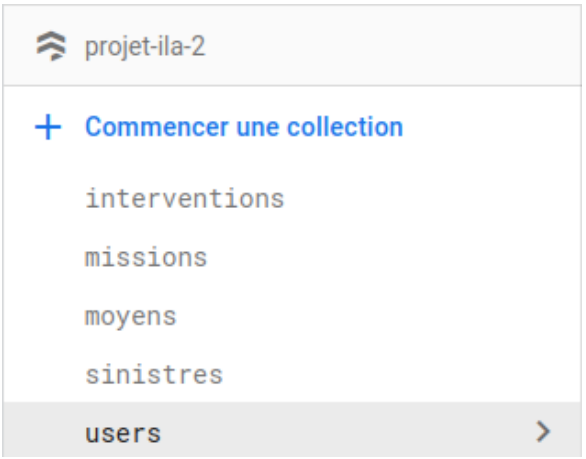


Figure 2 : collections de données firestore

[1] https://firebase.google.com

3.3.1. Collection users

Une fois l'utilisateur inscrit via le service firebase Authentication, il est ajouté à la collections des utilisateurs : **users** dont le schéma est le suivant

users {
email : email de l'utilisateur
firstName : prénom de l'utilisateur
login : login de l'utilisateur
name : nom de l'utilisateur
rôle : le rôle de l'utilisateur : prend deux valeurs : Role.Operator ou Role.Intervener
userID : est l'identifiant de l'utilisateur généré par le service firebase Authentication.
}

Figure 3 : champs d'un document de la collection users

3.3.1. Collection sinistres

Les code sinistres sont stockés dans la collection **sinistres**

sinistres{
codeSinistres : le code sinistre respectant la nomination chez les sapeurs pompiers
description : la description du code sinistre
}

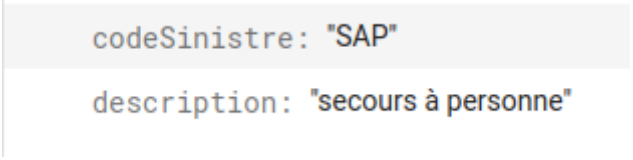


Figure 4 : champs d'un document de la collection sinistres

3.3.1. Collection moyens

Les moyens d'intervention sont stockés dans la collection **moyens**

moyens{
codeMoyen : le code moyen respectant la nomination chez les sapeurs pompiers
couleurDefaut : la couleur par défaut du moyen, pouvant prendre une des 6 valeurs :

- Colors.black
- Colors.green
- Colors.red

- Colors.blue
- Colors.orange
- Colors.purple

description : la description du moyen
}

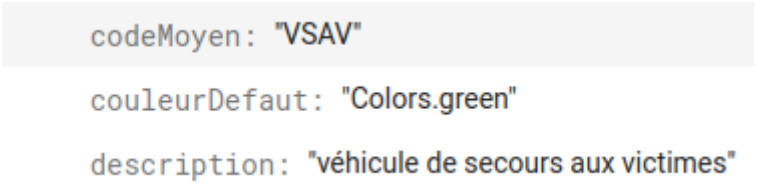


Figure 5 : champs d'un document de la collection moyens

3.3.1. Collection interventions

Les interventions et les différentes données qui lui sont associées : moyen d'interventions, symbols d'actions, missions drone ...etc sont stockés dans la collection **interventions**

interventions{
nom : le nom de l'intervention
adresse : adresse de l'intervention
codeSinistre : le code sinistre de l'intervention
date : la date de l'intervention
futureMission: stocke la mission drone en cours de création
latitudeDrone: la latitude actuelle du drone
longitudeDrone: la longitude actuelle du drone
missions : la liste des identifiants des missions drone liées à cette intervention
moyens : la liste des moyens liés à cette intervention
symbols : la liste des symbols sur la SITAC liés à cette intervention
}

Figure 6 : champs d'un document de la collection interventions

Chaque moyen de la liste des moyens de l'intervention possède ces informations :
{
arriveA : date d'arrivé du véhicule sur le lieu de l'intervention
basePath : chemin de base de l'icône représentant le véhicule
codeMoyen : le code du moyen
couleur : la couleur choisie pour le moyen
couleurDefaut : la couleur par défaut du moyen
demandeA : date de demande du moyen
departA : date de départ du moyen, est elle même la date de demande du moyen pour les moyens de premiers départ, et la date de confirmation par l'opérateur codis pour les moyens supplémentaire demandé par le cos.
description : la description du moyen
etat : l'état actuel du moyen, peut prendre une des valeurs suivantes :

- Etat.enAttente
- Etat.prevu
- Etat.enCours
- Etat.retourne

id : l'identifiant du moyen
latitude : la latitude du moyen sur la SITAC
longitude : la longitude du moyen sur la SITAC
retourneA : date du retour du moyen
}

Figure 7 : champs d'un élément de la liste moyens

Chaque symbol sur la SITAC lié à l'intervention possède ces informations :
{

basePath : chemin de base de l'icône représentant le symbol
couleur : la couleur du symbol
etat : l'état courant du symbol, peut prendre 2 valeurs

- Etat.prevu
- Etat.enCours

id : l'id du symbol
latitude : la latitude actuelle du symbol sur la SITAC
longitude : la longitude actuelle du symbol sur la SITAC
nomSymbol : nom du symbol
}

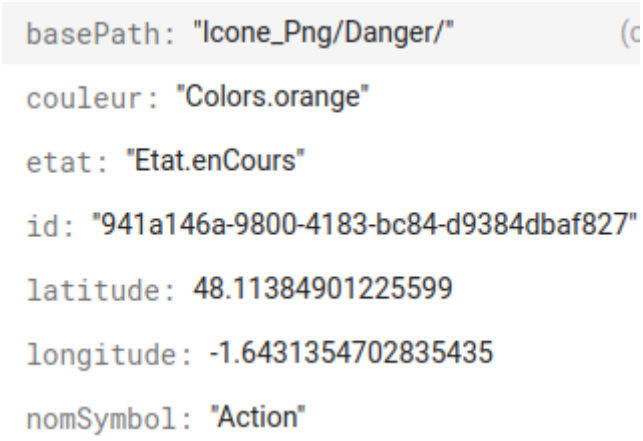


Figure 8 : champs d'un élément de la liste symbols

3.3.1. Collection missions

Les missions du drone et les différentes données qui lui sont liées sont stockées dans la collection **missions**

{
id : l'id de la mission
idIntervention : l'id de l'intervention à laquelle est liée la mission
interestPoints : liste des points d'intérêt définissant un trajet drone
photos : liste des url des images dans firebase cloud storage
name : nom de la mission
segment : permet de paramétrer une mission, true si par segment, false si par zone
state : etat courant de la mission, peut prendre 3 valeurs:

- StateMission.Starting
- StateMission.Running
- StateMission.Ending

streamVideo : permet de paraméter le stream vidéo du drone , si true mission avec stream, false sans stream vidéo.
video : lien de la vidéo drone (ici lien de la dernière photo de simulation de vidéo drone)
}

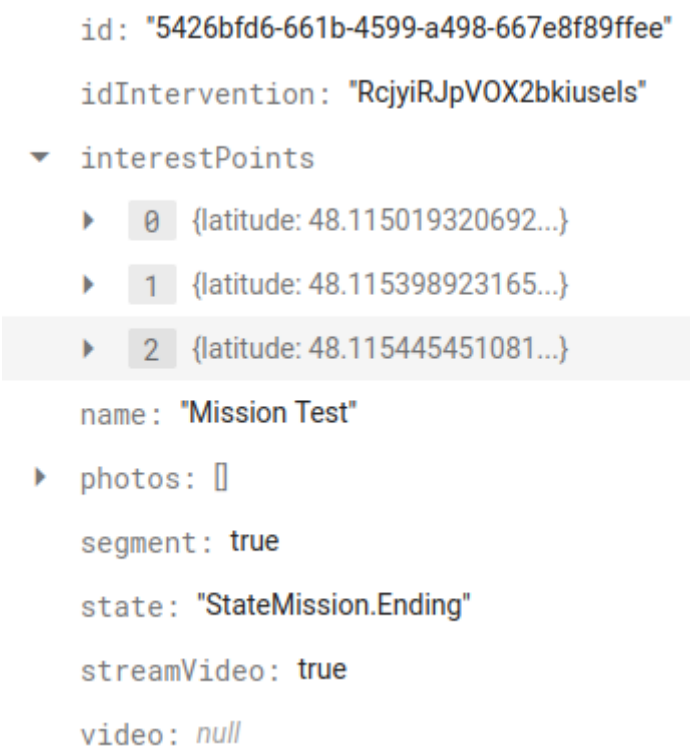


Figure 9 : champs d'un document de la collection missions

Chaque point d'intérêt de la liste interetPoints possèdes ces informations :
{
latitude : latitude du point d'intérêt
longitude : longitude du point d'intérêt
photo : paramètre qui permet de spécifier si le point d'intérêt est un point de prise de photo si true, false sinon.
}

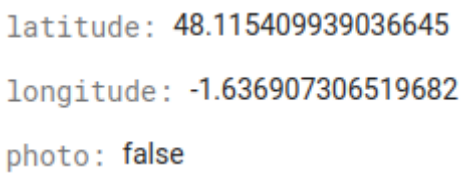


Figure 10 : champs d'un élément de la liste interestPoint

3.4 - Service MavlinkApp

Cette application est celle chargée du contrôle du drone. Son fonctionnement global consiste au lancement d'une mission sur le drone à chaque fois qu'elle en reçoit venant de l'application Flutter ou tant qu'elle en a en attente dans son ExecutorService. Le contrôle même du drone est effectué avec la bibliothèque java mavsdk comme vu lors de l'UE PIT.

3.4.1 Généralités

Application spring boot version : 2.4.4
Version java : 11
dépendances :

- spring-boot-starter-web
- spring-boot-starter-test
- mavsdk version 0.5.1
- httpclient version 4.5.13
- springfox-swagger2 version 2.7.0
- springfox-swagger-ui version 2.7.0

Le lancement de l'application se fait depuis la racine du projet avec la commande ***./mvnw spring-boot:run***

3.4.2 Modèles de données

Le modèle de données de cette application consiste en l'ensemble des données échangées entre les différentes applications et celle-ci.
Ce sont les classes :

- **CurrentPicture** : Elle sert de modèle d'envoie d'une photo prise à un point d'intérêt lors d'une mission. Elle a ainsi l'id de la mission, la position de prise de la photo et la photo en tableau de bytes ;
- **CurrentPosition** : modèle d'envoie de la position courante du drone au webservice 2 ;
- **InterestPoint** : modèle de réception d'un point d'intérêt. Il définit la latitude, la longitude et un booléen indiquant si une photo doit-être prise à ce point d'intérêt ;
- **MissionDrone** : modèle de réception d'une mission par l'application Flutter. Il définit l'id de la mission, l'id de l'intervention, le nom de la mission, la liste des points d'intérêts associés etc..
- **StateMission** : elle sert uniquement à définir l'état de la mission courante. Pour ce faire elle encapsule un identifiant de mission et un l'état de celle comme un String.

3.4.3 Web Services

Cette application expose principalement deux endpoints :

- "/api/mission/sendMissionCoordonates" : cet endpoint prend en paramètre un json de type MissionDrone correspondant à une mission programmée sur l'application Flutter. L'application lance derrière la mission sur le drone à partir des commandes de gestion fournies par mavsdk ;
- "/api/mission/cancel" : appelée pour annuler une mission déjà envoyée par l'application Flutter au webservice.

3.5 - Service Firebase Application

Cette Application sert à communiquer indirectement avec l'application Flutter, pour transmettre les données drone en temps réel. Cette communication consiste concrètement à stocker dans Firebase firestore : les positions du drone communiquer par l'application Mavlink Application et dans cloud storage les photos et vidéos transmises par le drone. L'application flutter récupère ainsi ses données en temps réel.

Remarque: les photos/vidéos réelles prises par le drone n'ont pas pu être récupérées. Les photos stockées sont récupérées en utilisant l'API google places[2].

3.5.1 Généralités

Application spring boot version : 2.4.4
Version java : 11
dépendances :

- spring-boot-starter-web
- spring-boot-starter-test
- firebase-admin version 7.1.1
- commons-io version 2.4
- springfox-swagger2 version 2.7.0
- springfox-swagger-ui version 2.7.0

L'arborescence du projet est la suivante :

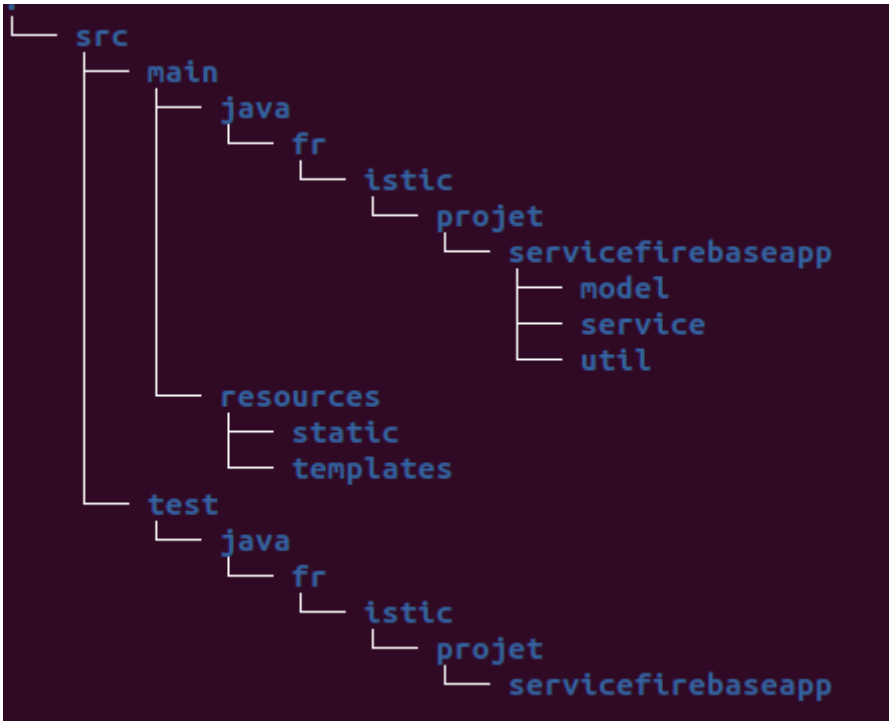


Figure 11 : arborescence du projet ServicefirebaseappApplication

[2] API google places : https://developers.google.com/maps/documentation/places/web-service/photos

L’application est lancée via la méthode main de la classe ServicefirebaseappApplication se trouvant dans le fichier ServicefirebaseappApplication.java à la racine du projet.
Ceci lance un service web sur le port 8080 (valeur par défaut, pouvant être modifiée en ajoutant la ligne suivante dans src/main/resources/application.properties:
server.port = [LE PORT]).

3.5.2. Modèle de données

Toutes les classes définissant le modèle de données se trouvent dans le package model. Les classes du modèle sont implémentées de telle façon à correspondre au modèle des deux autres applications flutter Application et MAvlink Application, on y trouve :

- **Intervention** la classe qui représente une intervention, comprenant l’**adresse** de l’intervention, son **code Sinistre**, des listes des **moyens**, de **symboles** et de **missions**, position courante du drone : **latitudeDrone, longitudeDrone**
- **Mission** classe concernant la mission du drone, permet de définir les points d’intérêt qui définissent le trajet du drone **interestPoints**, les points de prise de photos **photos, l’état courant de la mission.**
- **Moyen** la classe d’un moyen comprenant son id, code couleur et d’une description
- **MoyenIntervention** classe correspondant à l’intervention d’un moyen ce qui comprend une date de demande, de départ (demande validé), d'arriver (arrive sur les lieux) et de retour (quitte les lieux), ainsi que de sa position
- **InterestPoint** représente les points d’intérêt du trajet drone, un boolean **photo** permet de spécifier si le point est un point de prise de photo.
- **Position** classe comprenant une valeur longitude et latitude
- **SymbolIntervention** comprend les informations nécessaires à la création d’une icône de symbole.
- **FileDTO, MissionInfos, MissionBody, DroneInfosBody, StateMission :** classes servant à sérialiser/désérialiser les entrées/sorties des web services.

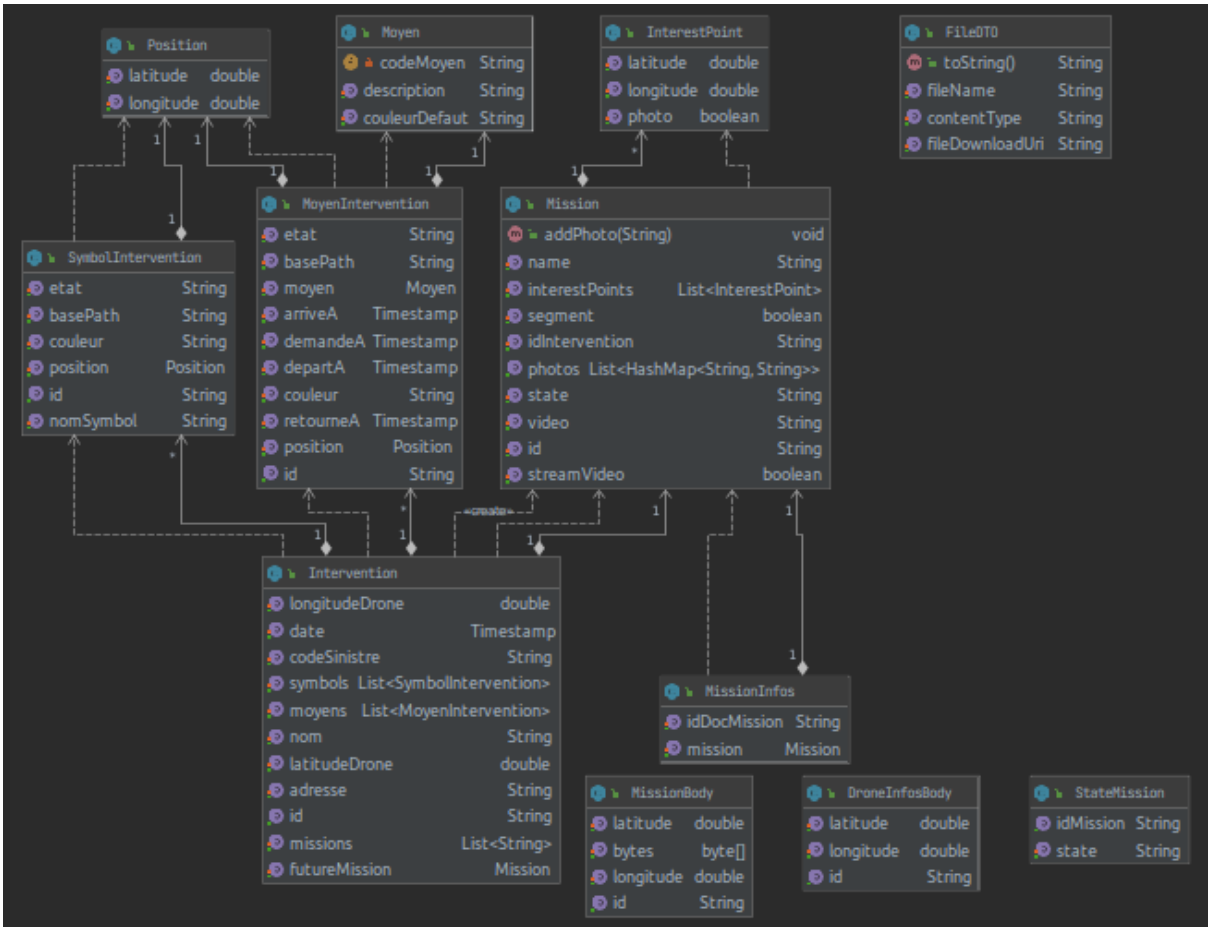


Figure 12 : Architecture statique ServicefirebaseappApplication

3.5.3. Web Services

L'application expose des services REST définis dans le package service, au niveau de la classe FirebaseController. la documentation de l'API est accessible sur [http://[URL]:[PORT]/swagger-ui.html](http://[URL]:[PORT]/swagger-ui.html)

URL : domaine où l'application est déployée, **PORT** : le port sur lequel l'application est accessible.

La description de chacun des services est données ci-dessous :

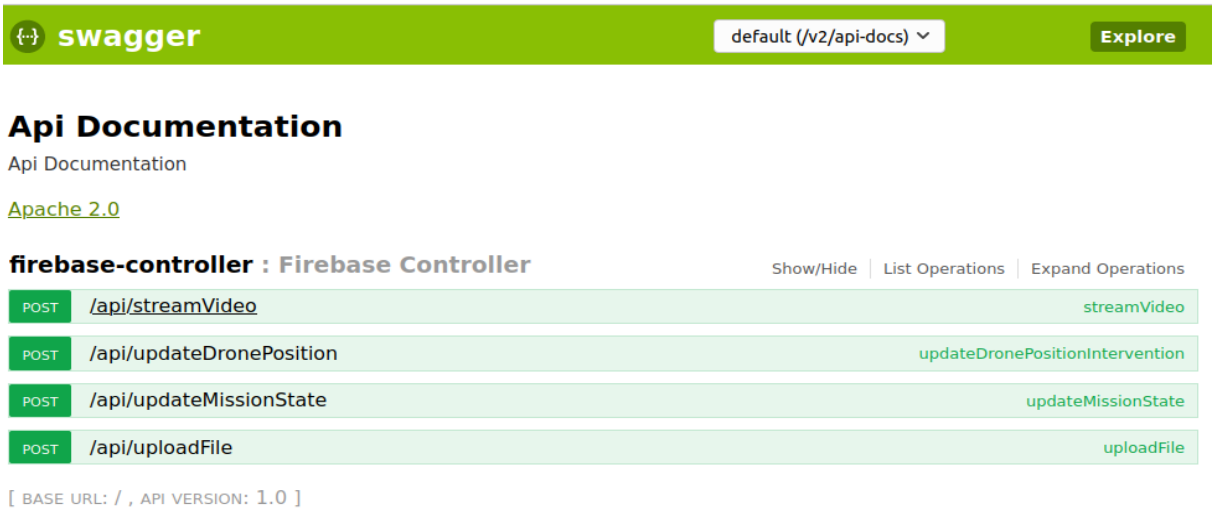


Figure 13 : API documentation swagger-ui ServicefirebaseappApplication

3.5.3.1. /api/updateMissionState

Cette méthode permet de mettre à jour le statut d'une mission drone, en appelant la méthode setMissionState de la classe FirebaseService, setMissionState récupère la mission sur firestore à partir de son id dans un objet MissionInfos puis met à jour son statut et pousse la référence mise à jour sur firestore.

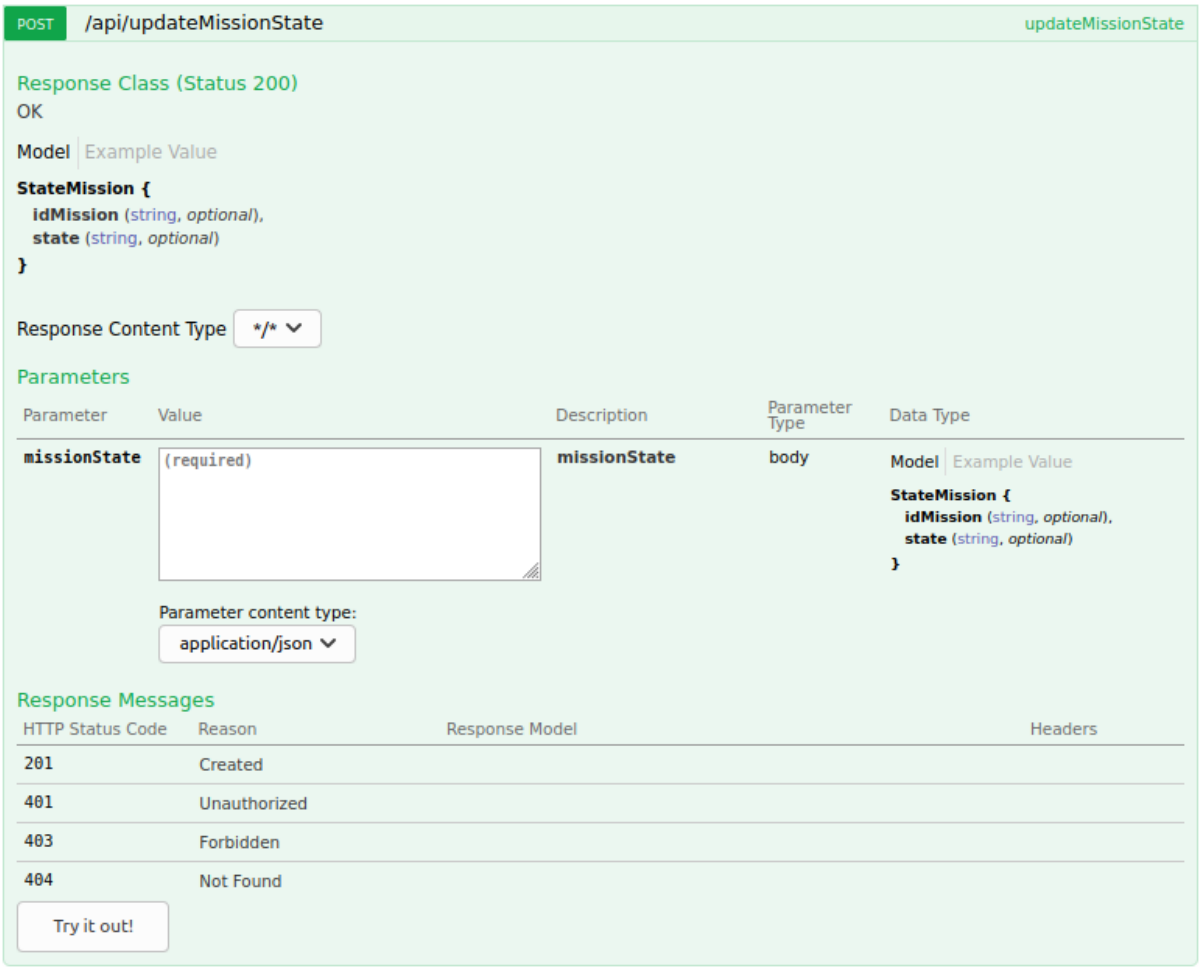


Figure 14 : Endpoint /api/updateMissionState

Exemple de body :

```
{
  "idMission": "5426bfd6-661b-4599-a498-667e8f89ffee",
  "state": "StateMission.Ending"
}
```

Exemple de réponse :

```
{
  "idMission": "5426bfd6-661b-4599-a498-667e8f89ffee",
  "state": "StateMission.Ending"
}
```

3.5.3.2. /api/updateDronePosition

Cette méthode permet de mettre à jour la position du drone, en appelant la méthode updateDronePositionIntervention de la classe FirebaseService, updateDronePositionIntervention récupère l'intervention sur firestore à partir de son id dans un objet Intervention puis met à jour les deux champs latitudeDrone, longitudeDrone et pousse la référence mise à jour sur firestore.

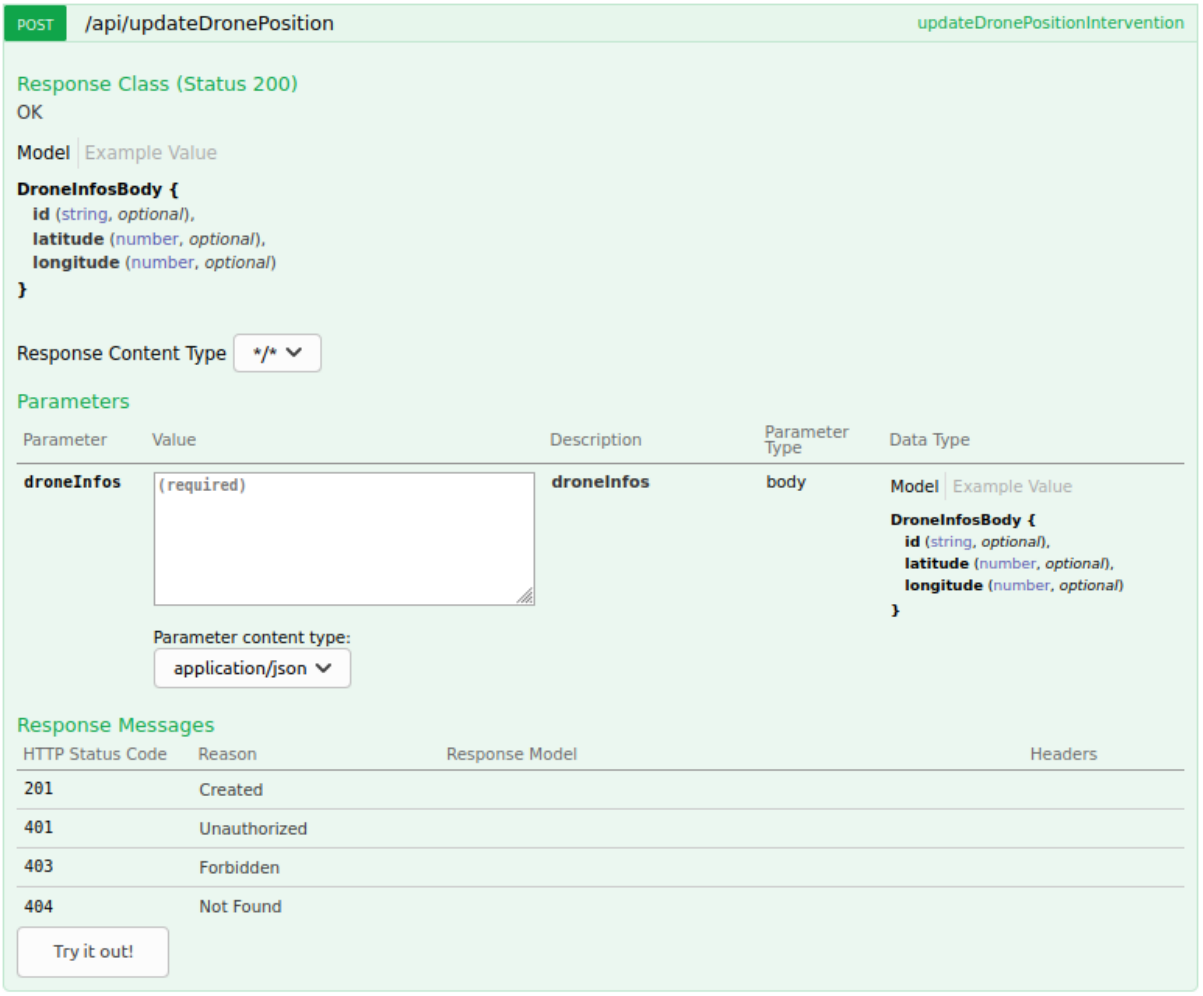


Figure 15 : Endpoint /api/updateDronePosition

Exemple de body :

```
{
  "id": "to27Og2xMFjos1DMyGoA",
  "latitude": 48.11003924083917,
  "longitude": -1.679389217470662
}
```

Exemple de réponse :

```
{
  "id": "to27Og2xMFjos1DMyGoA",
  "latitude": 48.11003924083917,
  "longitude": -1.679389217470662
}
```

3.5.3.3. /api/uploadFile

Cette méthode permet de stocker une photo dans cloud firestore, en appelant la méthode uploadFileFromBytes de la classe FirebaseService, uploadFileFromBytes fait appel à la méthode getImageForPosition qui joue le rôle d'un mock (à défaut de ne pas avoir les photos réelles prises par le drone, elle utilise l'API google places pour récupérer une image statique

de la position passée en paramètres), puis ajoute l'image dans le bucketName du storage configuré dans la méthode initializeFirebase, sous le dossier images/[ID_DE_LA_MISSION]

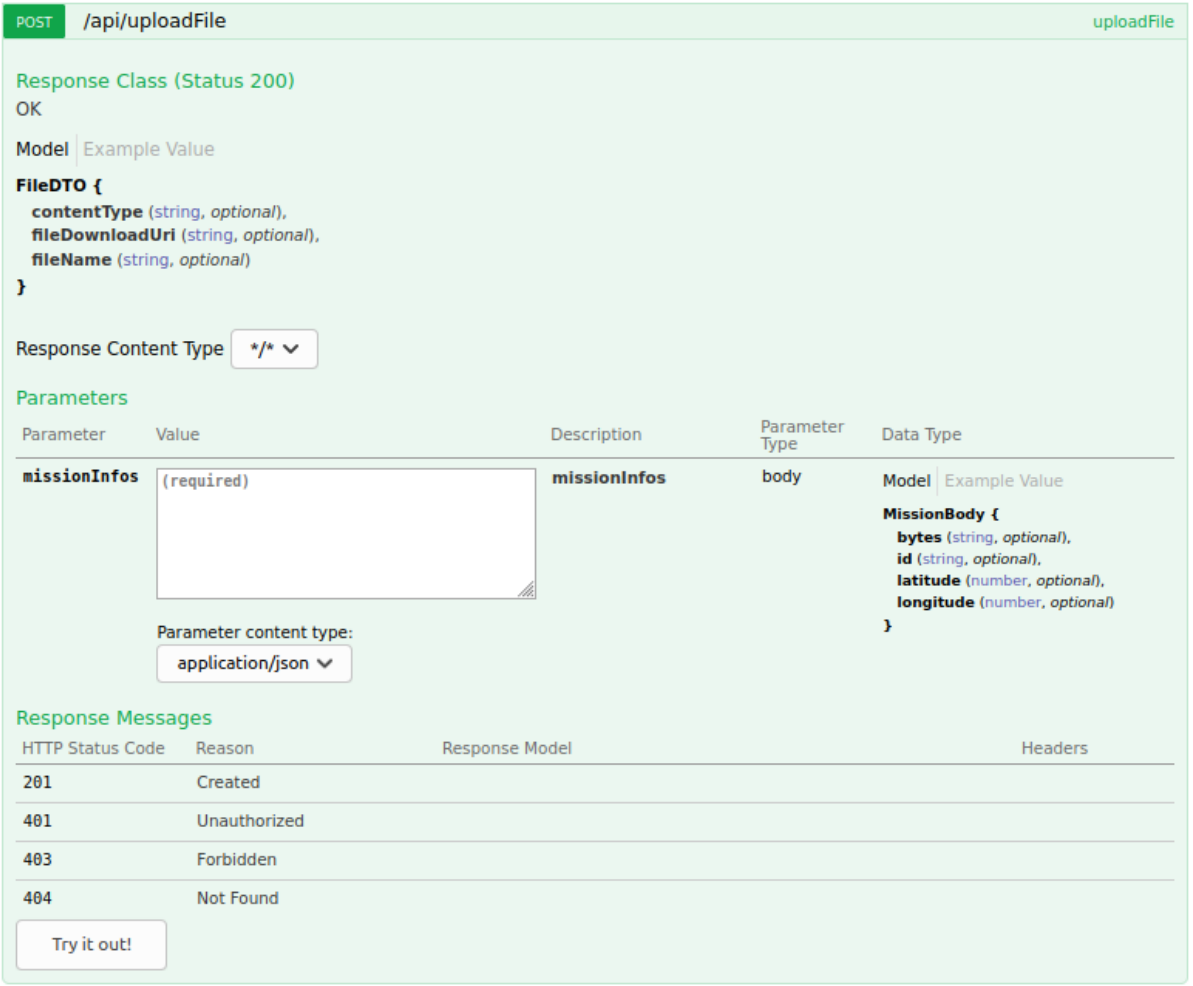


Figure16 : Endpoint /api/uploadFile

Exemple de body :

{
"bytes": null,
"id": "5426bfd6-661b-4599-a498-667e8f89ffee",
"latitude": 48.11003924083917,
"longitude": -1.679389217470662
}

Exemple de réponse :

{
"contentType": "image/png",
"fileDownloadUri": "gs://projet-ila-2.appspot.com/images/138d34e2-2985-4907-ac40-d3e0e4959e4c/ADwbgXIA.png",
"fileName": "BvmWbnXp.png"
}

3.5.3.4. /api/streamVideo

A défaut de ne pas avoir réussi à récupérer le stream vidéo du drone, Cette méthode permet de sauvegarder des images prises à un intervalle de temps très court ce qui permet de simuler un stream vidéo.
Elle permet de stocker une photo dans cloud firestore, en appelant la méthode uploadFileFromBytes de la classe FirebaseService, cette fois l'image est ajouté l'image dans sous le dossier vidéos/[ID_DE_LA_MISSION]

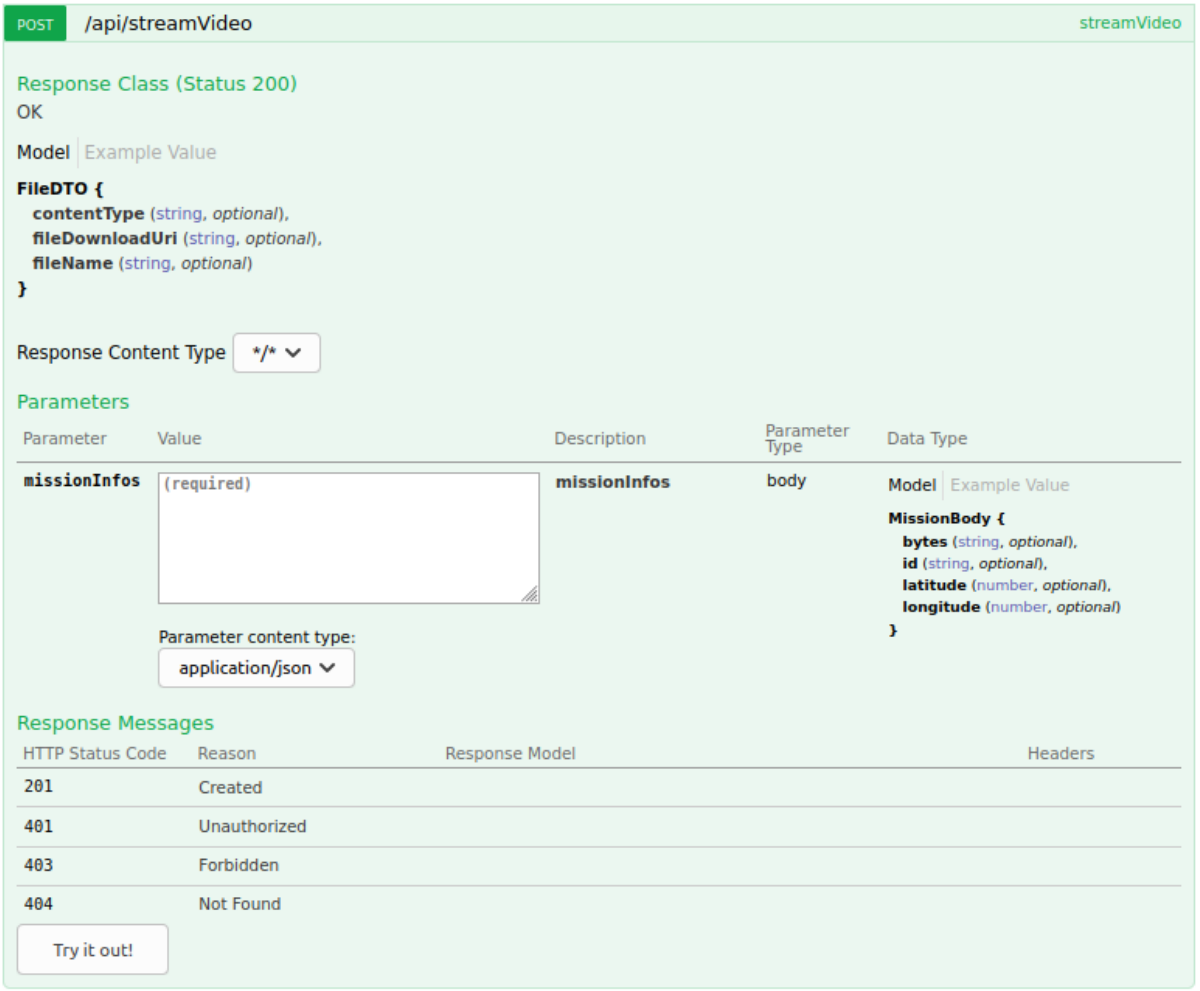


Figure 17 : Endpoint /api/streamVideo

Exemple de body :

```
{
  "bytes": null,
  "id": "5426bfd6-661b-4599-a498-667e8f89ffee",
  "latitude": 48.11003924083917,
  "longitude": -1.679389217470662
}
```

Exemple de réponse :

```
{
```

"contentType": "image/png",
"fileDownloadUri": "gs://projet-ila-2.appspot.com/images/138d34e2-2985-4907-ac40-d3e0e4959e4c/ADwbgXIA.png",
"fileName": "BvmWbnXp.png"
}