

L3 Informatique

2022/2023



Projet développement smartphone

WAVE RIDER

Rapport 3/3

Corentin CHEVE, Nicolas FAURE, Baptiste GALLEY,
Jules RABUS, Raphaël REGNAULT, Clément TUTIN

SOMMAIRE

1.	<i>Présentation du projet et compréhension des besoins</i>	3
2.	<i>Membres de l'équipe, répartition des rôles</i>	4
3.	<i>Outils utilisés</i>	5
4.	<i>Maquette de l'application</i>	6
5.	<i>Étude des besoins du projet</i>	9
6.	<i>Développement de l'application et explications des classes</i>	13
a.	Classe Drone	13
b.	Classe Point	13
c.	Classe Trajectoire	13
d.	MainActivity	14
e.	MapsActivity	14
f.	TrajectoryAddActivity	14
g.	ManagedTrajec	15
h.	myAdapter	16
i.	TrajectoireManager	16
7.	<i>Gestion de projet</i>	17
a.	Compréhension et appropriation du projet	17
b.	Premiers diagrammes	17
c.	Planification et répartition des tâches	19
d.	Rencontres et résolutions des problèmes rencontrés	19
8.	<i>Rendus finaux</i>	21
9.	<i>Conclusion</i>	25
10.	<i>Index</i>	26

1. Présentation du projet et compréhension des besoins

Ce projet consiste à créer une application pour smartphone qui permet de piloter un drone marin à distance. Il s'agit d'un projet réalisé dans le cadre d'une évaluation finale pour l'EC *Développement sur smartphone* en L3 Informatique. Nous l'appellerons **Wave Rider**.

Notre application se compose de 4 vues principales :

- Menu / Accueil (Vue n°1)
- Pilotage manuel du drone (Vue n°2)
- Création d'une trajectoire (Vue n°3)
- Affichage de l'ensemble des trajectoires / Sélection d'une trajectoire (Vue n°4)

Pour mettre en place le pilotage du drone, nous avons besoin de récupérer différentes entrées :

- Clavier du smartphone afin de nommer les trajectoires
- Les différents capteurs du smartphone tels que le gyroscope et l'accéléromètre
- Trames NMEA-0183⁽¹⁾ donnant la position initiale du drone lors du lancement de l'application

L'application fournit en sortie l'affichage de la carte sur laquelle est affichée la position du drone en temps réel, ainsi que différents points géographiques en lien avec la trajectoire potentiellement insérée par l'utilisateur.

2. Membres de l'équipe, répartition des rôles

Nous sommes une équipe de 6 personnes, tous issus de L3 Informatique. Nous nous sommes réparti les rôles de cette manière :

- **Clément TUTIN**
Chef de projet
- **Baptiste GALLEY**
SCRUM Master
- **Nicolas FAURE**
Lead Developer
- **Corentin CHEVE**
Développeur
- **Jules RABUS**
Développeur
- **Raphaël REGNAULT**
Développeur

À partir de cette organisation, nous avons mis en place des sous-équipes composées de 2 personnes :

- Clément et Baptiste
- Nicolas et Jules
- Corentin et Raphaël

3. Outils utilisés

Notre application est développée en langage *Kotlin*, il s'agit donc d'une application à destination du système d'exploitation *Android*. Nous utilisons *Android Studio* comme *IDE*.

La convention *camelCase* est utilisée pour l'attribution des noms de nos classes et variables.

L'ensemble de notre programme est publié dans un répertoire privé sur la plateforme *GitHub*⁽²⁾. Cela nous permet de partager le code entre tous les membres du projet, et de mettre en place un système de versioning.

Pour gérer notre avancée de développement, nous utilisons *Trello*⁽³⁾ afin de visualiser l'ensemble des activités à effectuer, en développement, à tester et terminées. Un diagramme de GANTT⁽⁴⁾ est généré à partir de notre espace de travail *Trello*.

Pour générer notre diagramme de classes et d'utilisations, nous utilisons le logiciel *Visual Paradigm*.

Nous avons décidé d'utiliser ces outils pour une raison bien particulière : la connaissance. Dans le cadre de notre formation, nous avons au moins eu affaire une fois à l'ensemble de ces outils. Chaque membre de l'équipe a donc une base commune de connaissance de ces logiciels, ce qui rend la communication et la compréhension des différents éléments créés très naturelle.

4. Maquette de l'application

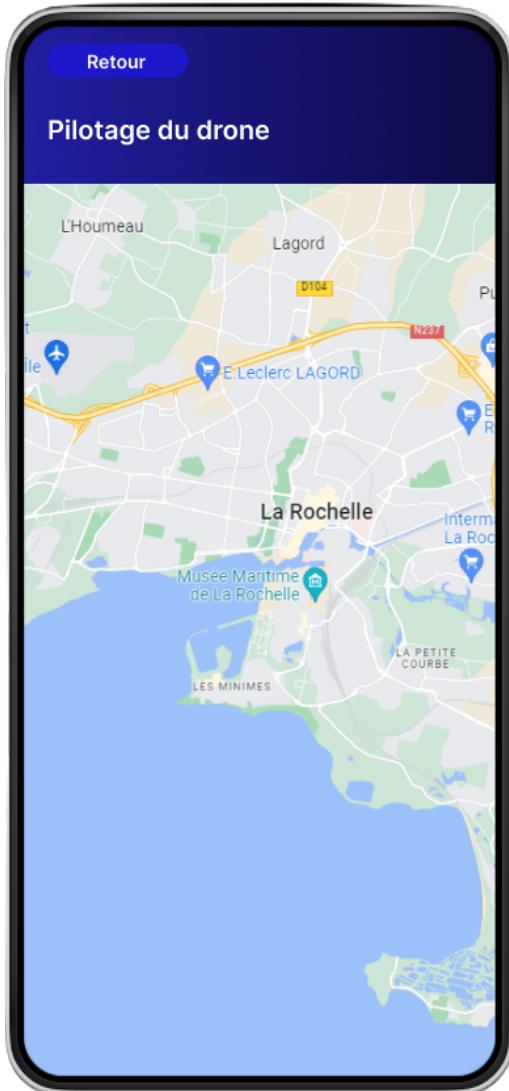
Avant de commencer le développement et l'étude détaillée des besoins du projet, nous avons effectué des maquettes des différentes vues de notre application afin de nous donner une direction de travail, sur le logiciel *Figma*⁽⁵⁾.



Maquette de la vue n°1

Menu d'accueil Vue n°1

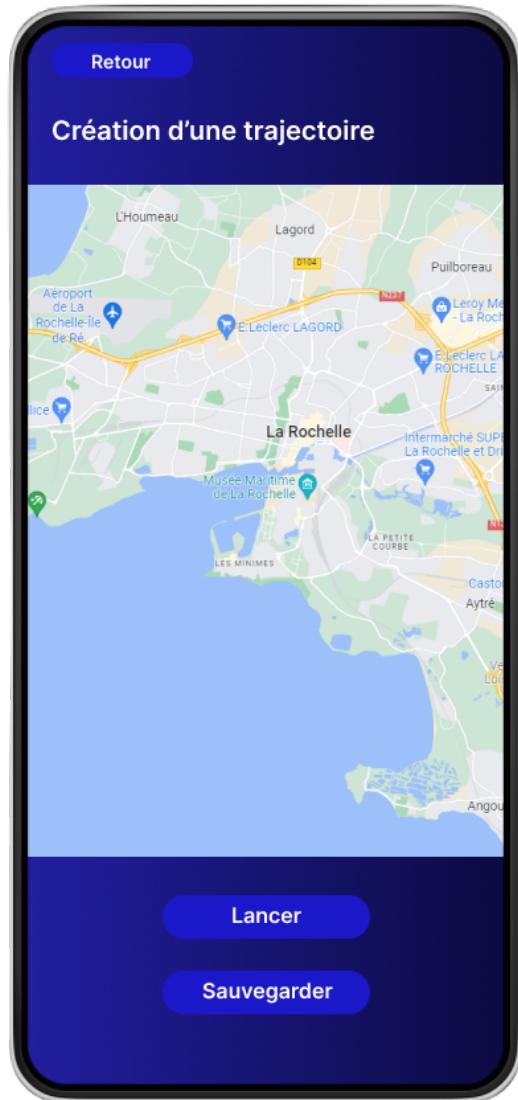
Cette vue est celle sur laquelle l'application se lance. C'est donc celle que l'utilisateur voit en premier. Cette vue permet d'accéder aux 3 autres vues de l'application.



Maquette de la vue n°2

Pilotage Manuel du drone Vue n°2

C'est à travers cette vue que le drone sera piloté manuellement. Il n'apparaît pas sur notre maquette mais un repère indiquera la position du drone en instantanée. Pour le piloter, il suffira de faire bouger le smartphone sur différents axes, comme sur un jeu vidéo. Par exemple, incliner le smartphone vers la gauche fera aller le drone sur la gauche.



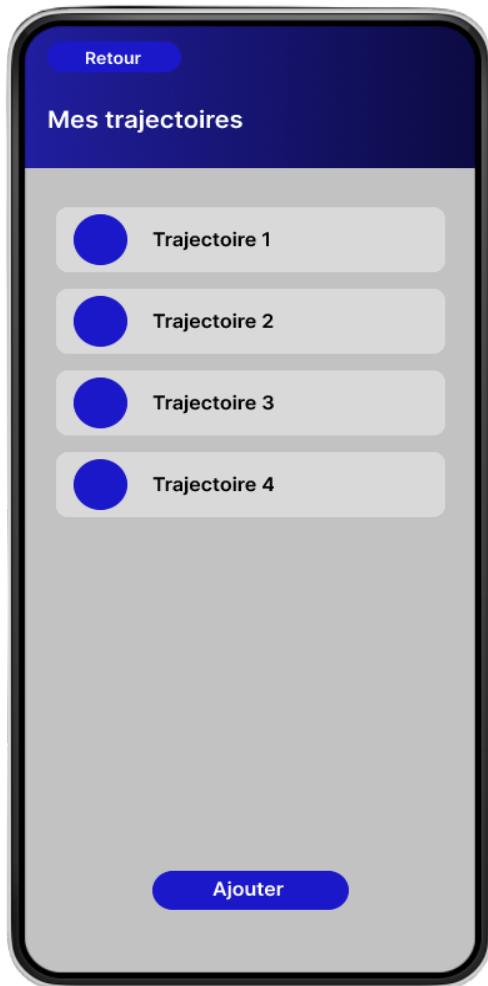
Maquette de la vue n°3

Affichage des trajectoires sauvegardées Vue n°4

Dans cette vue sont référencées toutes les trajectoires enregistrées, et donc créées au préalable sur la vue n°3 (ci-dessus). L'utilisateur peut sélectionner une trajectoire et le drone se mettra alors en mouvement, ou alors accéder à la vue n°3 à travers le bouton *ajouter*.

Création d'une trajectoire Vue n°3

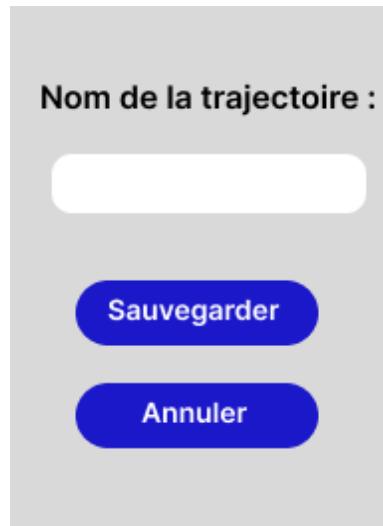
La création d'une trajectoire s'effectuera sur cette vue. Le principe est que l'utilisateur appuie sur la carte pour créer un repère. Il doit donc en créer au minimum 2 afin de créer une trajectoire. Lorsque celle-ci est créée, il peut soit la lancer et le drone la suivra automatiquement, ou alors la sauvegarder pour une utilisation future, l'utilisateur sera redirigé sur la vue n°4 (ci-dessous).



Maquette de la vue n°4

**Pop-up
Liée à la vue n°3**

Lorsque l'utilisateur crée une trajectoire, un pop-up apparaît pour que celui-ci affecte un nom à la trajectoire en cours de création. Ce nom peut être par exemple la destination (La Pallice).

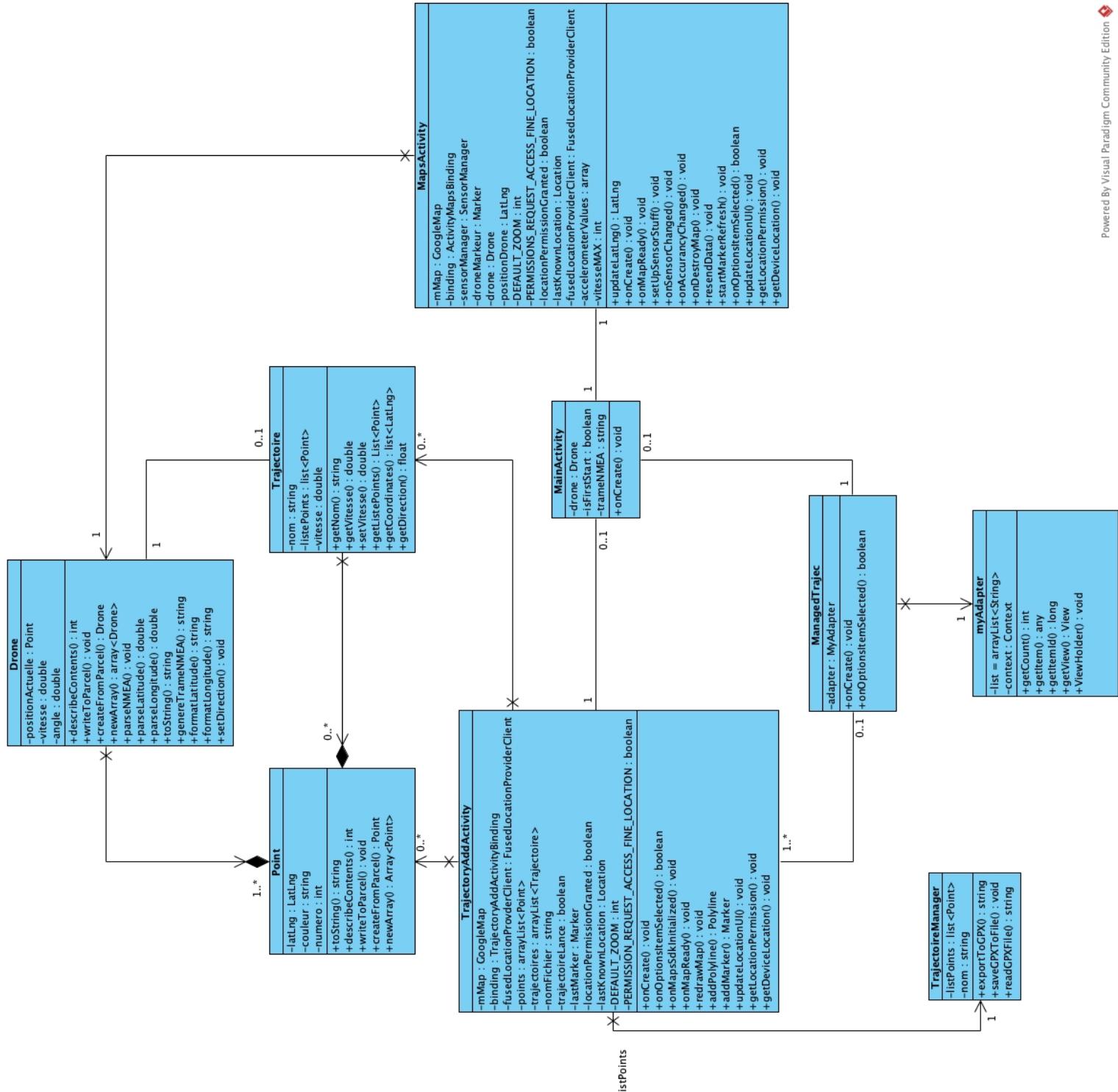


*Maquette du pop-up dans la vue
n°3*

5. Étude des besoins du projet

Afin de cerner l'ensemble des besoins du projet, nous avons modélisé différents diagrammes. Nous savons qu'ils comportent des erreurs, mais nous avons préféré les laisser afin de ne pas obtenir un avantage sur les autres équipes de la promotion qui ont effectué leur soutenance après la nôtre.

- #### - Diagramme de classes :



- Diagramme de cas d'utilisation des fonctionnalités :

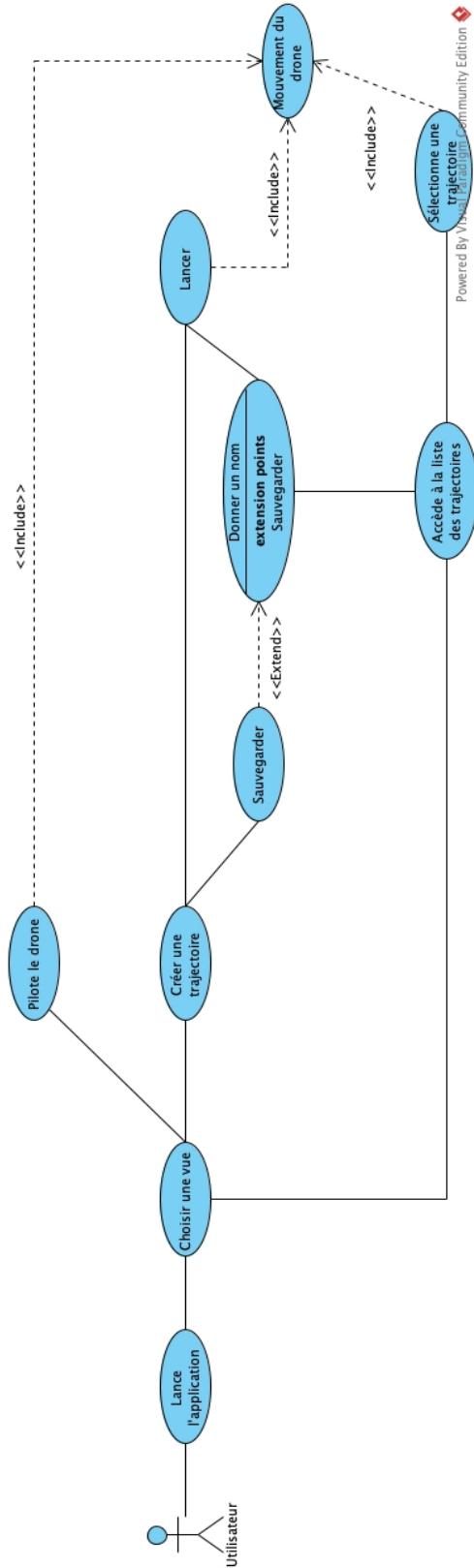


Diagramme des fonctionnalités – Version du 06/04/2023

Diagramme de GANTT⁽⁴⁾ (aperçu) :

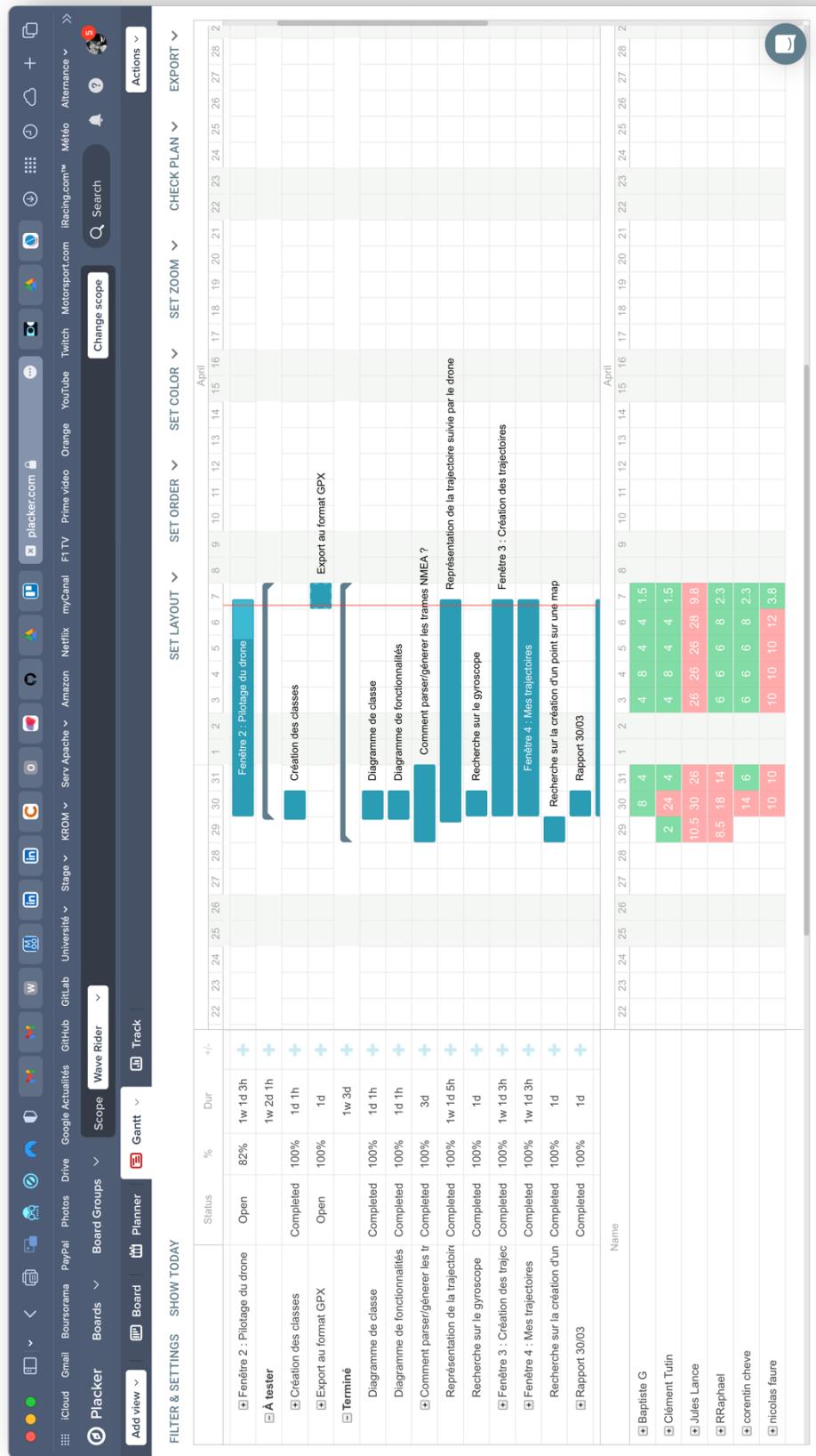
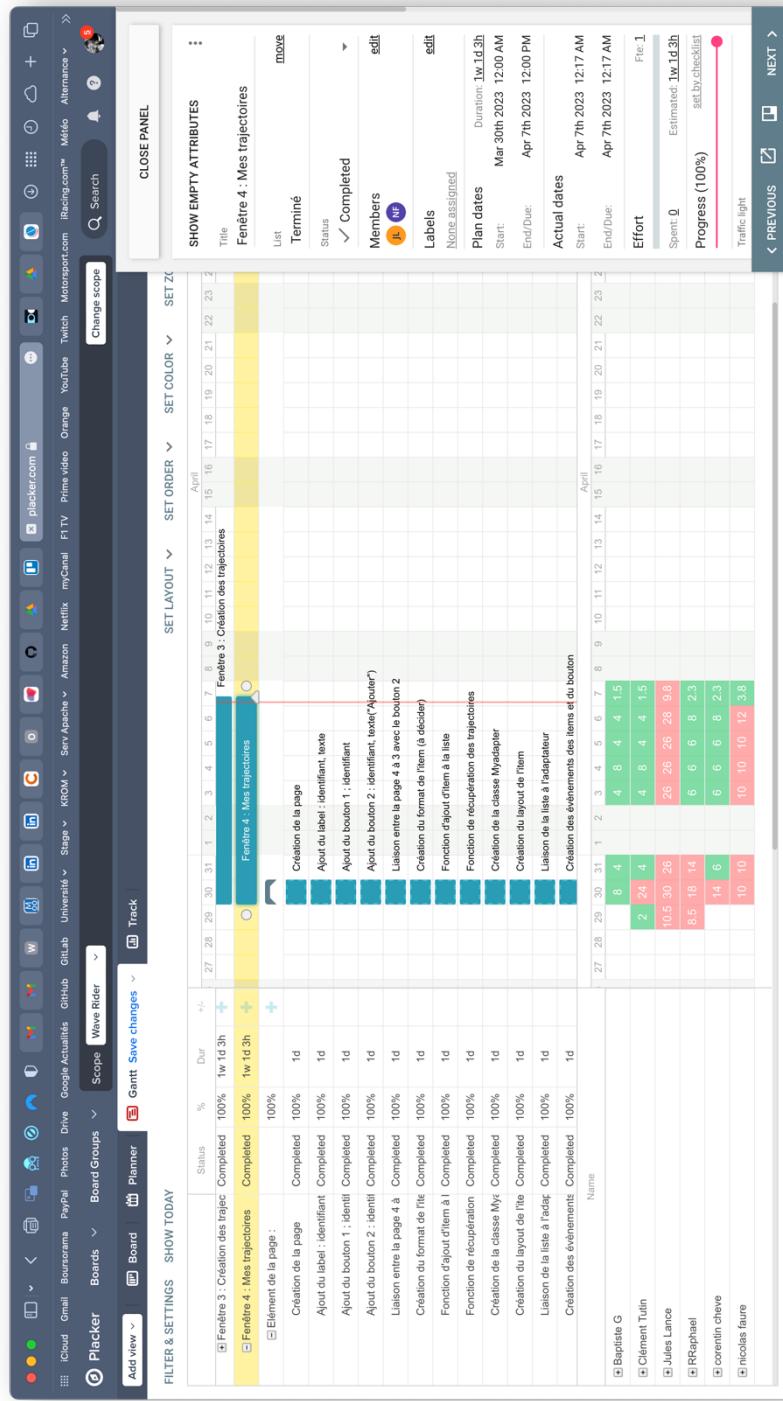


Diagramme de GANTT – Version du 30/03/2023

Détail pour une activité :



Activité du diagramme de GANTT – Version du 30/03/2023

Ce diagramme est généré depuis l'application *Trello*, grâce à l'extension *Placker*. Ce diagramme ne pouvant pas être exporté sous format *pdf*, vous trouverez le lien⁽⁴⁾ dans [l'index](#).

6. Développement de l'application et explications des classes

La répartition du développement s'est effectuée de la manière suivante :

- **Nicolas et Jules :**

MainActivity
TrajectoryAddActivity
ManagedTrajec
myAdapter

- **Corentin et Raphaël :**

MapsActivity
TrajectoireManager

Voici l'explication des différentes classes contenues dans l'application, avec les principales fonctions :

a. Classe Drone

La classe drone utilise une trame NMEA stockée dans une variable String afin de donner la position du Drone.

La fonction *parseNMEA()* prend la trame et analyse les données NMEA pour en extraire différentes propriétés :

- La latitude
- La longitude
- La vitesse

La fonction *genereTrameNMEA()* récupère les données évoquées ci-dessus et met à jour les propriétés du drone en les renvoyant sous forme d'une chaîne de caractères. À chaque mouvement du drone, une trame NMEA est générée dans la vue *MapsActivity*.

La fonction *setDirection()* permet de mettre à jour la direction du drone à partir de l'angle en radian venu du gyroscope. Celle-ci retourne alors l'angle du drone en degré.

b. Classe Point

La classe *Point* représente un point créé dans *TrajectoryAddActivity*.

c. Classe Trajectoire

La classe *Trajectoire* est composée de plusieurs points.

La fonction *getCoordinates()* permet de générer toutes les coordonnées de la trajectoire, avec le point de départ, les potentiels points intermédiaires, et le point d'arrivée.

La fonction `getDirection()` permet de calculer la direction vers le prochain point de la trajectoire, en degré.

d. MainActivity

Cette vue est le centre névralgique de notre application, c'est elle qui permet de naviguer à travers les différentes vues de l'application.

La fonction `onCreate()` permet la création de l'activité et de son *layout*. La navigation entre les différentes vues à partir de *MainActivity* se fait via l'ajout de `clickListener` sur les boutons de navigation, et via l'utilisation d'`intent` afin de passer l'objet *Drone* sans perdre sa position actuelle.

e. MapsActivity

Cette vue permet de piloter le drone en récupérant les données du gyroscope du smartphone.

La fonction `onCreate()` initialise la carte Google Maps, récupère les permissions de localisation de l'utilisateur puis crée et affiche le marqueur du drone sur la carte avec sa position et direction actuelle.

La fonction `onMapReady()` récupère l'instance de *GoogleMap*, met à jour l'interface utilisateur en fonction de l'état des permissions, crée le marqueur du drone sur la carte et crée des *listeners* pour les boutons du zoom et de suivi de caméra.

La fonction `onDestroyMap()` supprime le fragment de la carte et renvoie l'objet *drone* à l'activité *MainActivity*.

La fonction `startMarkerRefresh()` est appelée toutes les secondes pour mettre à jour la position du marqueur du drone sur la carte et calculer la vitesse du drone en utilisant les données de l'accéléromètre.

La fonction `updateLatLng()` est utilisée pour mettre à jour la position du drone sur la carte. Cette fonction prend en paramètre la position actuelle du drone, la direction et la vitesse du drone, et calcule la nouvelle position du drone sur la carte.

La fonction `onSensorChanged()` est appelée lorsque la valeur d'un capteur change. Cette fonction est utilisée pour détecter le mouvement de l'appareil et mettre à jour les capteurs.

La fonction `onOptionsItemSelected()` est appelée lorsque l'utilisateur appuie sur le bouton de retour. Cette fonction renvoie l'utilisateur et le drone à l'écran principal de l'application.

f. TrajectoryAddActivity

C'est cette vue qui permet la création d'une trajectoire.

La fonction `onCreate()` permet la création de l'activité et de son layout, avec des `clickListener` sur les boutons *Lancer* et *Sauvegarder* afin de lancer le pilote automatique ou le pop-up de

sauvegarde. Cette fonction fait également en sorte que le dernier moteur de rendu de la carte soit utilisé.

La fonction `onOptionsItemSelected()` permet de naviguer vers le menu principal.

La fonction `addMarker()` permet l'ajout d'un point sur la carte avec un *id* et sa position renseignés, puis est ensuite retourné.

La fonction `addPolyline()` permet d'ajouter un *polyline* qui est ajouté sur la carte à partir des coordonnées du point A vers le point B (ou *Marker*). Le *polyline* généré est ensuite retourné.

La fonction `onMapReady()` appelle les fonctions externes de Google afin de mettre en place la localisation du téléphone de l'utilisateur sur la carte. Elle récupère le *intent* afin que le trajectoire récupère le fichier *.traj*.

Un *clickListener* est mis en place sur la carte pour permettre à l'utilisateur de placer un point. Un autre est également placé sur une trajectoire qui a été générée afin de modifier la vitesse du drone sur le tronçon sélectionné, grâce à l'apparition d'un pop-up.

Un *dragLister* est ajouté sur un *marker* dans le but de pouvoir modifier la trajectoire. L'affichage de celle-ci est alors modifié, la position des *markers* et des lignes avec.

g. ManagedTrajec

L'objectif de cette vue est de permettre à l'utilisateur de visualiser toutes les trajectoires qu'il a créé grâce à leur nom. Ceci est possible grâce à la création d'une *listView*, qui est complétée avec des items qui représentent chacun une trajectoire.

Pour ce faire, nous effectuons les étapes suivantes :

- Initialisation de la liste de trajectoires
- Lecture du fichier où sont sauvegardées toutes les trajectoires
- Création d'un attribut *Trajectoire*
- Ajout de cet item à la liste

La liste est ensuite passée en paramètre à l'*adapter* qui permet la création de la *listView*. Il faut ensuite créer un *onClickListener* pour le bouton *ajouter*. Celui-ci permet de retourner sur la vue n°3. Il permet également à l'utilisateur de créer une nouvelle trajectoire.

Il faut de plus créer un *onItemClickListener* pour visualiser une trajectoire dans sa globalité. Chaque item (trajectoire) de la liste devient cliquable, et devient alors une passerelle jusqu'à la vue n°3 qui charge ainsi la bonne trajectoire. L'utilisateur peut alors voir tous les points qui la composent et lancer le drone en mode automatique.

La fonction `onCreate()` participe à l'implémentation de l'affichage du drone, et en particulier sur la récupération des fichiers *.traj*.

h. myAdapter

La classe *myAdapter* a pour but de créer un item d'une liste de façon personnalisée. Lors de l'apprentissage de ce langage, nous avons créé des adaptateurs standards avec uniquement du texte en paramètre. Ici, nous avons du texte et une image avec un numéro. Nous devons donc créer notre propre classe adaptateur et créer notre propre format d'item (*item_layout.xml*). Cette classe permet de créer les différents items en fonction des données de la liste.

La fonction *getView()* permet la création de la vue de chaque item, en fonction des données fournies dans la liste. La classe *ViewHolder* est l'élément composant l'item, ici *textView* et *button*. Ici le bouton est non cliquable et n'a aucune utilité propre (car il n'est raccordé à aucune fonction). C'était un moyen simple, rapide, et esthétique pour afficher l'identifiant de la trajectoire.

i. TrajectoireManager

Cette classe permet d'exporter les données d'une trajectoire que l'on enregistre dans la vue *TrajectoryAddActivity* dans un fichier au format GPX.

La fonction *exportToGPX()* génère une chaîne de caractères représentant les données de la trajectoire au format GPX. On utilise des balises pour structurer les données.

La fonction *saveGPXToFile()* sauvegarde le contenu de la chaîne de caractères générée par la méthode *exportToGPX()* dans un fichier avec un nom donné par l'utilisateur.

La fonction *readGPXFile()* lit le contenu d'un fichier GPX représentant une trajectoire sélectionnée par l'utilisateur, et le retourne sous forme d'une chaîne de caractères.

7. Gestion de projet

a. Compréhension et appropriation du projet

Afin de comprendre les objectifs et attentes du projet, nous avons d'emblée créé les rendus des vues, afin d'obtenir une idée concrète des besoins de celui-ci. C'est à partir de ces rendus que nous nous sommes décidés sur la conception du projet, avec entre autres la création des diagrammes, la mise en place des sous-équipes et des tâches attribuées.

b. Premiers diagrammes

- Diagramme de classes :

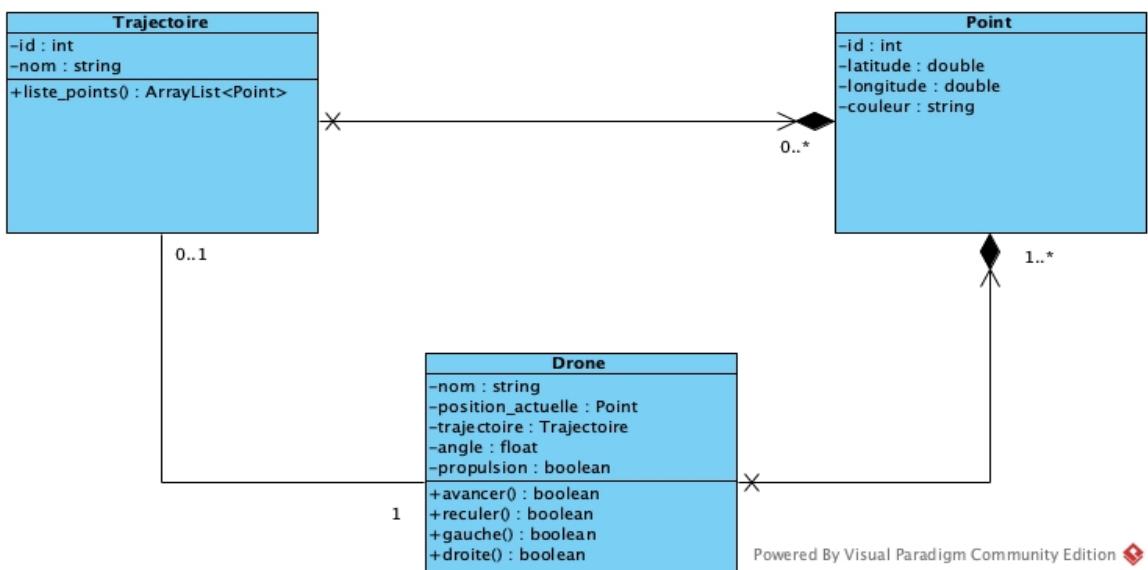
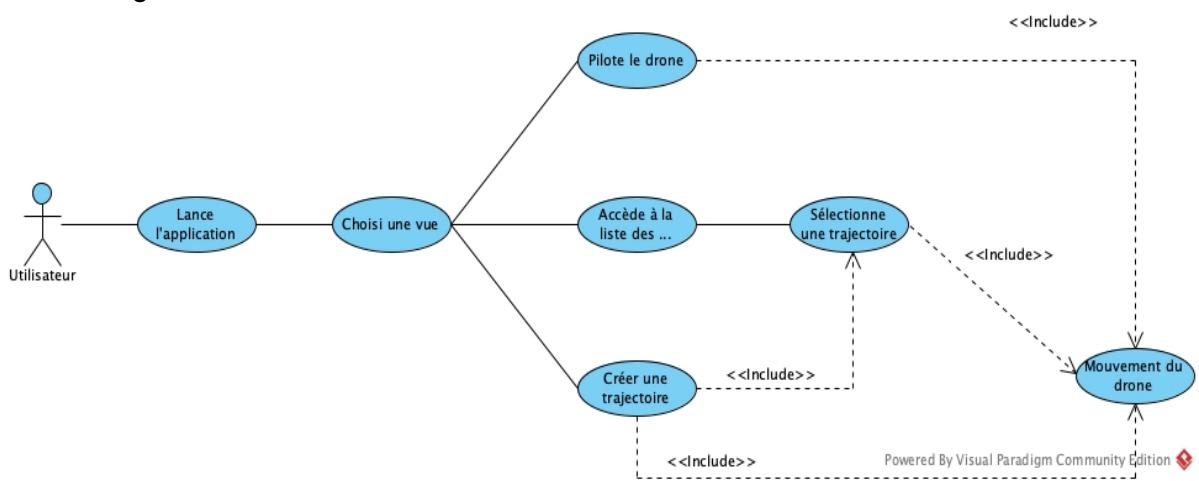


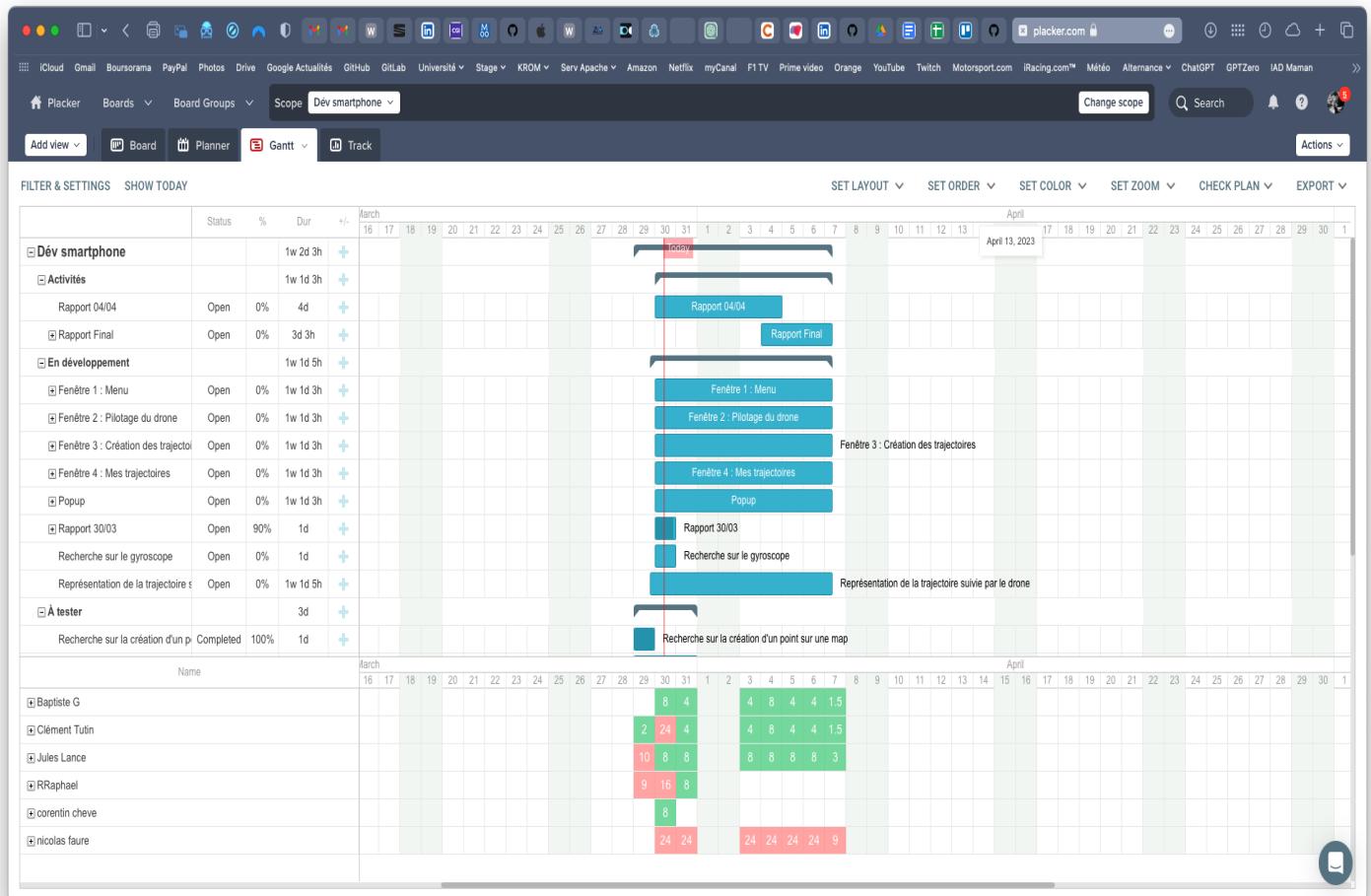
Diagramme de classes – Version du 30/03/2023

- Diagramme de cas d'utilisation des fonctionnalités :



Diagrammes des fonctionnalités – Version du 30/03/2023

- Diagramme de GANTT :



Diagrammes de GANTT – Version du 30/03/2023

c. Planification et répartition des tâches

Nous avons utilisé la méthodologie *Agile*⁽⁶⁾ et l'outil *Trello*⁽³⁾ afin de mettre en relations toutes les tâches à réaliser, et visualiser les avancées de chaque grand axe du développement de notre application, avec la répartition du travail pour chacune des personnes du groupe. Comme indiqué précédemment, nous avons formé 2 sous-équipes de 2 développeurs, puis une sous-équipe avec le Scrum master et le chef de projet qui se sont occupés des tâches autres que du développement telles que la mise en place des diagrammes, des rapports, etc...

Nous avons régulièrement organisé des réunions, qu'elles soient en présentiel lors des séances de cours ou en distanciel sur un groupe Discord créé pour l'occasion, afin de pouvoir travailler tout en échangeant ensemble sur des questions posées, des problèmes rencontrés, des avis demandés, etc... Le fait de mettre en place des réunions régulières nous a permis d'obtenir un rythme de travail conséquent et efficace.

d. Rencontres et résolutions des problèmes rencontrés

Nous avons rencontré quelques problèmes, voici comment nous les avons abordés.

Affichage de la carte, du drone et visualisation du déplacement :

Afin de récupérer les données depuis le simulateur, il a fallu nous documenter sur la norme NMEA-0183⁽¹⁾ et comprendre son fonctionnement.

Pour le déplacement du drone virtuel, nous voulions qu'il se déplace de façon fluide et fidèle à la réalité. Nous devions donc prendre en compte les distances et le temps de parcours de celles-ci afin d'ajuster la vitesse du drone.

Le passage de données entre les vues nous a posé quelques soucis. *Intent* possédant plusieurs méthodes, nous en avons donc essayé plusieurs afin de trouver celle qui correspondait le mieux à nos besoins.

La carte, lors du changement d'une vue, ne se fermait pas correctement et posait donc des soucis lors de sa réouverture. Nous avons donc implémenté une méthode qui ferme la carte puis en réouvre une lors de l'ouverture de la nouvelle vue.

Afin de créer une interface utilisateur répondant à nos besoins, nous avons dû créer nos propres boutons pour modifier le zoom de la carte sur la vue n°2.

Affichage de la liste des trajectoires sauvegardées :

Le problème rencontré sur l'affichage de plusieurs éléments (texte et bouton) dans un seul item de la *listView* a été résolu grâce à la création de notre classe adaptateur (*myAdapter*) qui inclue notre propre personnalisation.

La gestion du clique de l'utilisateur sur la liste dans la vue n°4 nous a posé problème. Afin d'entourer le numéro de la trajectoire, nous avons utilisé un bouton circulaire mais celui-ci prenait le dessus sur le bouton principal de sélection de la trajectoire. Il a donc fallu faire en sorte que le clique du bouton circulaire soit en dessous du bouton principal comportant le numéro et le nom de la trajectoire.

Dans l'explication de la classe *TrajectoryAddActivity*, nous avons précisé que nous faisions en sorte de forcer l'utilisation de la dernière version du moteur de rendu de la carte Google Map. Ceci est dû au fait qu'il était impossible de générer un polyline avec un motif dessus.

Globalement, nous n'avons pas eu de grandes difficultés dans le développement du code. La plus grande difficulté réside dans le fait d'être 6 personnes au sein de ce projet. Réunir et faire fonctionner un code provenant de plusieurs sources est un exercice plus complexe qu'il n'en a l'air.

8. Rendus finaux

Voici les rendus définitifs des vues de notre application :

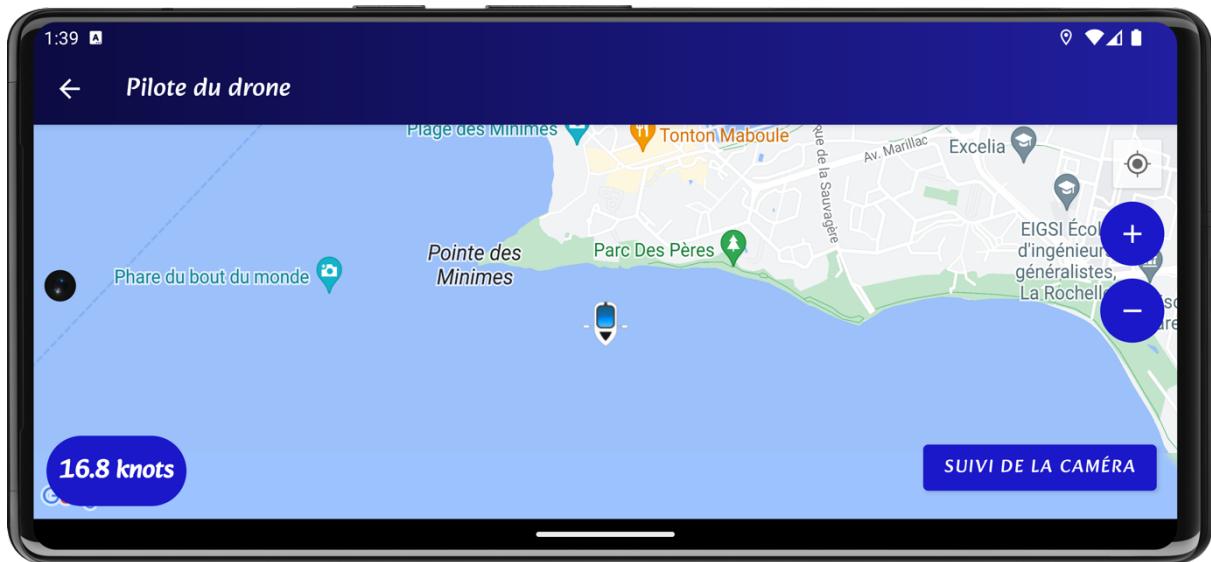


Screenshot de la vue n°1

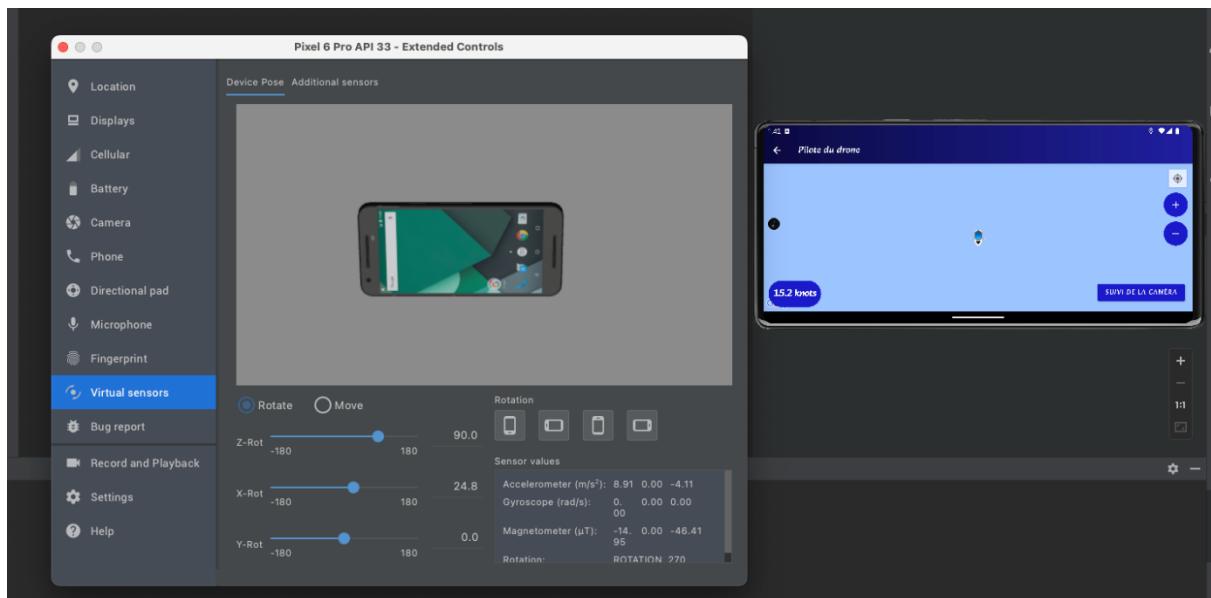
Dans cette vue, l'utilisateur peut sélectionner 3 vues :

- Pilotage du drone
- Création d'une trajectoire
- Mes trajectoires

Pilotage Manuel du drone Vue n°2



Screenshot de la vue n°2



Screenshot de la vue n°2 et du simulateur de mouvement du smartphone sur Android Studio

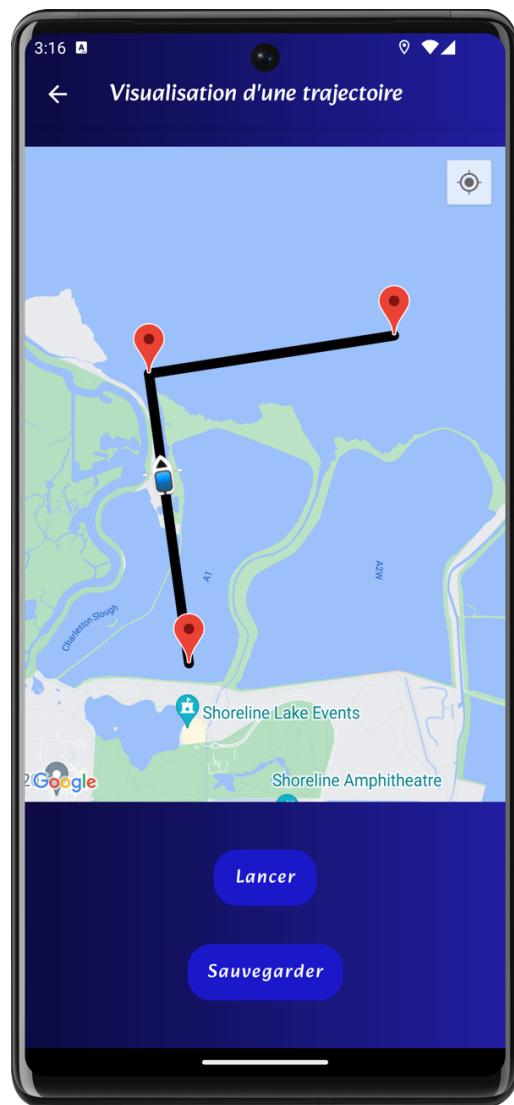
La vue n°2 est consacrée au pilotage manuel du drone. L'utilisateur a la possibilité de contrôler le drone grâce au gyroscope de son smartphone (comme illustré ci-dessus, dans le simulateur intégré à Android Studio) et voit en temps réel les déplacements ainsi que la vitesse de celui-ci. Il lui est également possible de zoomer, de se déplacer sur la carte et de recentrer la vue sur le drone, mais aussi de revenir à l'écran d'accueil (vue n°1).

Création / Lancement d'une trajectoire

Vue n°3



Vue d'une trajectoire
(Merci M. Mondou pour cette magnifique trajectoire)



Vue d'une trajectoire lancée

Cette vue n°3 est utilisée pour 2 cas :

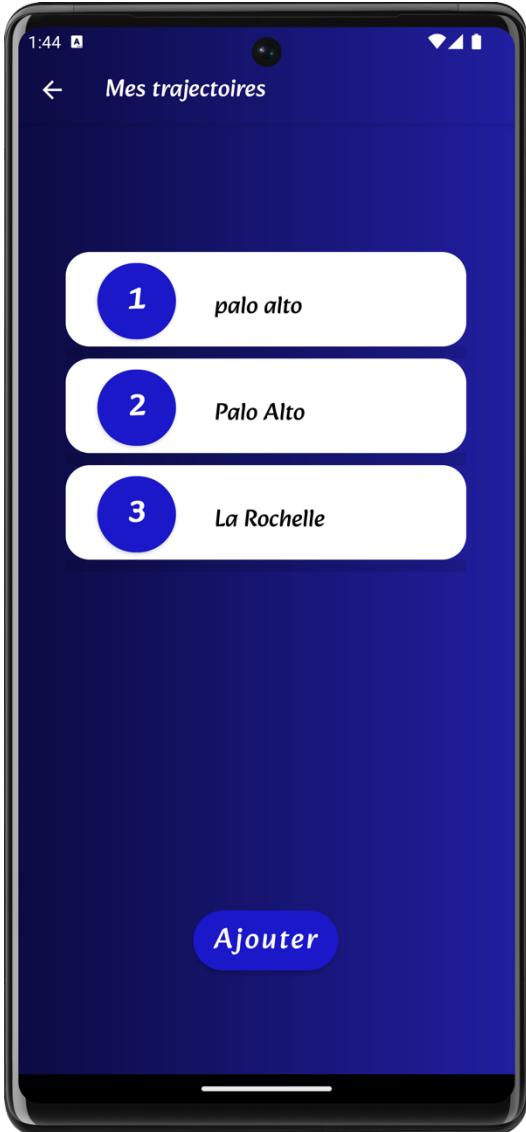
- La création d'une trajectoire

L'utilisateur clique sur la carte pour ajouter un point, puis doit ensuite en ajouter un autre pour créer une trajectoire. Une trajectoire doit contenir au minimum 2 points (1 de départ et 1 d'arrivée) mais peut être composée d'une infinité de points.

- Le lancement d'une trajectoire

Lorsque l'utilisateur appuie sur le bouton *Lancer*, une icône qui représente le drone apparaît. Le drone va alors suivre automatiquement la trajectoire noire, jusqu'au point d'arrivée.

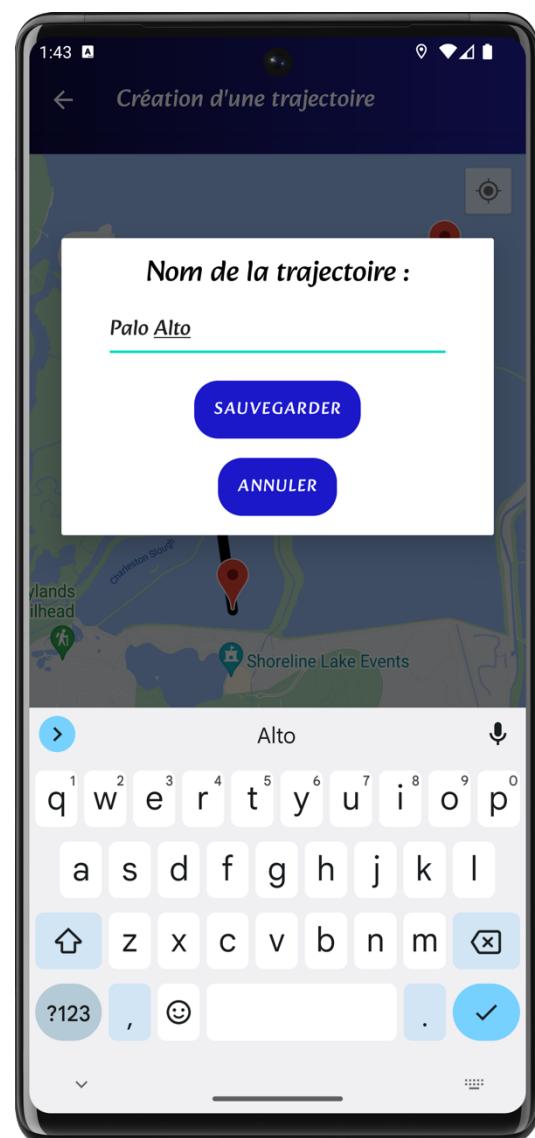
Dans ces 2 situations, l'utilisateur peut modifier la vitesse du drone en appuyant sur une droite entre 2 points. Un pop-up va s'afficher, lui permettant de régler la vitesse du drone sur le tronçon sur lequel il a appuyé. Celle-ci est réglée à 15 noeuds par défaut.



Vue de la sélection des trajectoires

Affichage des trajectoires sauvegardées Vue n°4

Cette vue regroupe toutes les trajectoires sauvegardées par l'utilisateur qui d'un simple clic affiche la trajectoire demandée sur la carte prête à être lancée.



Pop-up lors de la création d'une trajectoire dans la vue n°3

Pop-up
Liée à la vue n°3

Ce pop-up s'affiche lorsque l'utilisateur souhaite sauvegarder une trajectoire qu'il vient de créer. Il doit donc lui affecter un nom, comme dans l'image ci-dessous.

9. Conclusion

Ce projet répond selon nous à l'ensemble des objectifs imposés par le sujet. Nous avons réussi à obtenir une organisation de travail efficace et organisée, grâce aux différentes méthodes et outils évoqués lors de ce rapport.

En termes de développement, nous avons réussi à délivrer une application fonctionnelle, et en lien avec nos attentes évoquées au moment du début de nos discussions. Bien entendu, énormément de fonctionnalités peuvent être ajoutées, modifiées, ou améliorées. Nous avons encore de nombreuses idées en tête, qui pourraient être mises en place si nous devions continuer le développement de notre application.

Au vu du contexte, nous avons manqué de temps pour les mettre en place. Le temps donné pour réaliser ce projet n'était pas très important mais suffisait amplement. Seulement, le contexte actuel de la fin de notre semestre avec un nombre important de projets et d'évaluations fixées à la dernière minute et nos recherches personnelles pour nos poursuites d'études ont rendu la réalisation de ce projet plus difficile qu'à l'origine.

Néanmoins, cela reste une expérience très bénéfique pour chacun d'entre nous, qui nous a permis d'obtenir un premier aperçu d'un projet mené avec la méthode Agile dans des conditions proches du monde du travail, auxquelles nous serons concernés dans peu de temps dans le cadre de notre stage de fin d'études.

10. Index

(1) NMEA-0183

Spécification pour la communication entre équipements marins, dont les équipements GPS.

https://fr.wikipedia.org/wiki/NMEA_0183

(2) Projet GitHub

https://github.com/ClementTtn/Projet_Smartphone

(3) Espace de travail Trello

<https://trello.com/invite/b/hLiTqLS4/ATTIce8d10ff6a6d9a9460910b54a1a4696a1246199A/wave-rider>

(4) Diagramme de GANTT

<https://placker.com/app#/gantt?e=b&s=p40f3f>

(5) Logiciel Figma

<https://www.figma.com/fr/design/>

(6) Méthode Agile

Méthode de travail qui met en avant la collaboration entre des équipes auto-organisées et pluridisciplinaires.

https://fr.wikipedia.org/wiki/M%C3%A9thode_agile