Dossier projet

GeoChallenge Tracker

Jean Ceugniet

Table des matières

1 Expression du besoin . . . 3
1.1 Contexte . . . 3
1.2 Problématique . . . 3
1.3 Analyse de l'existant . . . 3
1.3.1 Fonctionnalités proposées par Project-GC . . . 3
1.3.2 Limites de Project-GC . . . 6
1.3.3 Valeur ajoutée de GeoChallenge Tracker . . . 6
1.4 Objectifs . . . 6
1.5 Périmètre fonctionnel et hors-périmètre . . . 6
1.6 Sources de données et intégrations . . . 6
1.7 Utilisateurs et rôles . . . 7
1.8 Parcours utilisateur clés . . . 7
1.9 Indicateurs de succès (KPI) . . . 7
1.10 Exigences non fonctionnelles . . . 7
1.11 Contraintes et dépendances . . . 7
1.12 Contraintes et risques . . . 7
1.13 GeoChallenge Tracker - Descriptif projet . . . 8
1.14 GeoChallenge Tracker - Project description . . . 9
2 Compétences mobilisées . . . 10
2.1 Développer une application sécurisée . . . 10
2.2 Concevoir et développer une application sécurisée organisée en couches . . . 10
2.3 Préparer le déploiement d'une application sécurisée . . . 10
2.4 Technologies utilisées . . . 11
3 Présentation client . . . 12
3.1 Client principal . . . 12
3.2 Attentes utilisateurs . . . 12
3.3 Personas . . . 12
3.3.1 Persona 1 : Le passionné des grands défis . . . 12
3.3.2 Persona 2 : Aïcha, la monitrice d'escalade . . . 12
3.3.3 Persona 3 : L'optimisateur rationnel . . . 13
3.4 Pool de testeurs . . . 13
4 Spécifications fonctionnelles . . . 14
4.1 Fonctionnalités principales . . . 14
4.1.1 Authentification et sécurité . . . 14
4.1.2 Gestion des challenges . . . 14
4.1.3 Import de caches . . . 15
4.1.4 Stockage des données . . . 15
4.1.5 Cartographie . . . 15
4.1.6 Statistiques et projections . . . 16
4.2 Évolutions futures . . . 16
4.3 Contraintes techniques . . . 16
5 Charte graphique . . . 17
5.1 État non loggé . . . 18
5.2 Register / Login . . . 19
5.3 État loggé (partie 1) . . . 20
5.4 Page d'accueil - Contenu complet . . . 21
6 Architecture logicielle . . . 22
6.1 Vue d'ensemble (Contexte) . . . 22
6.2 Vue conteneurs (environnements Dev/Prod) . . . 22
6.2.1 Dev (docker-compose) . . . 22
6.2.2 Prod . . . 22
6.3 Composants backend (modules principaux) . . . 23
6.4 Modèle logique des données (MongoDB) . . . 24
6.5 Modèle physique des données (MongoDB) . . . 25
6.6 Indexes créés (via seeding) . . . 26
6.7 Sécurité (vue architecture) . . . 27
6.8 Modèle objet (diagramme de classe UML) . . . 28

6.9 Séquences clés (diagramme de séquence UML) . . . 30
6.9.1 Import GPX → sync challenges → premiers snapshots . . . 30
6.9.2 Consultation d'un challenge et projection . . . 30
6.10 Performance & cache . . . 30
6.11 Déploiement & configuration . . . 30
6.12 Évolutions d'architecture envisagées . . . 31
7 Choix techniques . . . 32
7.1 Backend . . . 32
7.2 Frontend . . . 32
7.3 Base de données . . . 32
7.4 Intégrations externes . . . 33
7.5 DevOps et qualité . . . 33
8 DevOps — Conteneurisation, Déploiement et Git . . . 34
8.1 Objectifs . . . 34
8.2 Conteneurisation (Docker) . . . 34
8.2.1 Images . . . 34
8.2.2 docker-compose (développement) . . . 34
8.3 Git — Convention de branches et workflow . . . 34
8.3.1 Nommage des branches . . . 34
8.3.2 Politique de fusion . . . 34
8.4 CI/CD et déploiement . . . 34
8.4.1 GitHub Action (build & deploy réels) . . . 35
8.4.2 Déploiement distant sans script local . . . 36
8.4.3 Arborescence VPS & gestion des variables . . . 36
9 Gestion de projet . . . 38
9.1 Introduction . . . 38
9.2 Planning prévisionnel . . . 38
9.2.1 Outils de planification . . . 38
9.3 Suivi réel (commits et branches) . . . 40
9.4 Analyse des écarts . . . 40
9.5 Environnement humain . . . 40
9.6 Objectifs qualité . . . 40
10 Script de création / modification de la base de données . . . 41
10.1 Objectifs . . . 41
10.2 Pré-requis & variables d'environnement . . . 41
10.3 Données seedées . . . 41
10.4 Index créés (via seeding) . . . 41
10.5 Exécution (local, sans Docker) . . . 42
10.6 Exécution (via Docker Compose) . . . 42
10.7 Composants clés (extraits commentés) . . . 42
10.7.1 Assurer les index (idempotent) . . . 42
10.7.2 Seeding des référentiels & admin . . . 43
10.8 Bonnes pratiques & sécurité . . . 43
10.9 Vérification après exécution . . . 43
10.10Tests associés au seeding . . . 44
11 Veille — sécurité & technologies . . . 45
11.1 Cas concrets récents . . . 45
11.1.1 Compromission de paquets npm (attaque de la chaîne d'approvisionnement) . . . 45
11.1.2 Vulnérabilité FastAPI Guard (CVE-2025-46814) . . . 45
11.1.3 Incident de sécurité MongoDB (décembre 2023) . . . 45
11.2 Implications pour *GeoChallenge Tracker* . . . 45
11.3 Sources . . . 45
12 Lexique du géocaching (notions utiles au projet) . . . 47

1 Expression du besoin

1.1 Contexte

Le geocaching est un loisir mondial qui consiste à rechercher des caches dissimulées dans divers environnements, allant des zones urbaines aux milieux naturels les plus reculés. Certaines caches, appelées *challenges*, nécessitent non seulement d'être trouvées physiquement, mais aussi que certaines conditions soient remplies (nombre de caches trouvées, dates précises, attributs spécifiques, etc.).

1.2 Problématique

Le suivi des challenges ne se limite pas à une simple statistique ponctuelle : il devient une véritable gestion de projet personnelle. C'est précisément à ce stade que la problématique apparaît.

Les conditions à remplir sont souvent réparties dans le temps, difficiles à suivre mentalement, et nécessitent un outil de gestion précis. Aujourd'hui, de nombreux utilisateurs s'en remettent à des feuilles de calcul ou à des systèmes peu ergonomiques.

1.3 Analyse de l'existant

Pour comprendre l'intérêt d'un nouvel outil, il est nécessaire d'examiner les solutions déjà disponibles dans l'écosystème.

La seule alternative notable identifiée est **Project-GC**, plateforme externe proposant de nombreuses statistiques, dont une section dédiée aux challenges.

1.3.1 Fonctionnalités proposées par Project-GC

- **Statistiques exhaustives** : nombre de challenges trouvés, distribution par date, par D/T (difficulté/terrain), par difficulté calculée.
- **Tableaux et heatmaps** : visualisation de la couverture D/T et des périodes de complétion.
- **Détail des challenges trouvés** : liste par cache (GC code, nom, D/T, date).
- **Gamification** : badges, médailles, classements visibles.

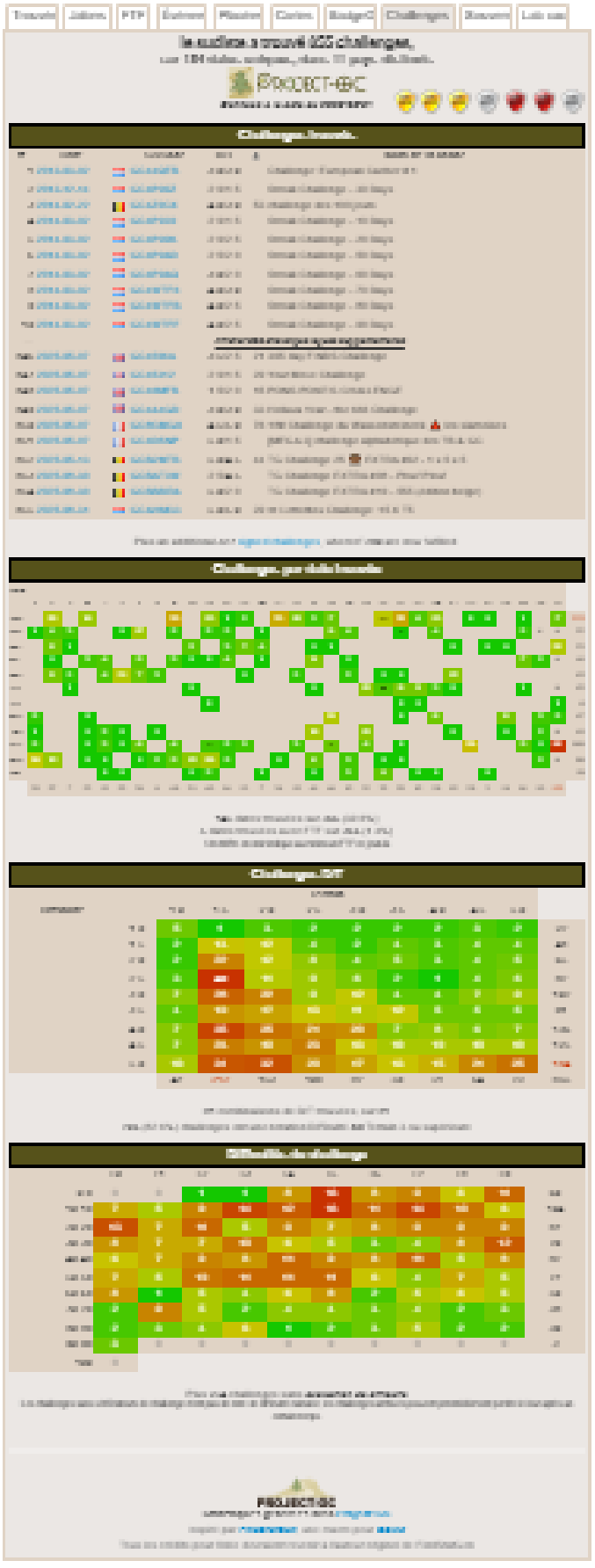


Figure 1: Challenges dashboard

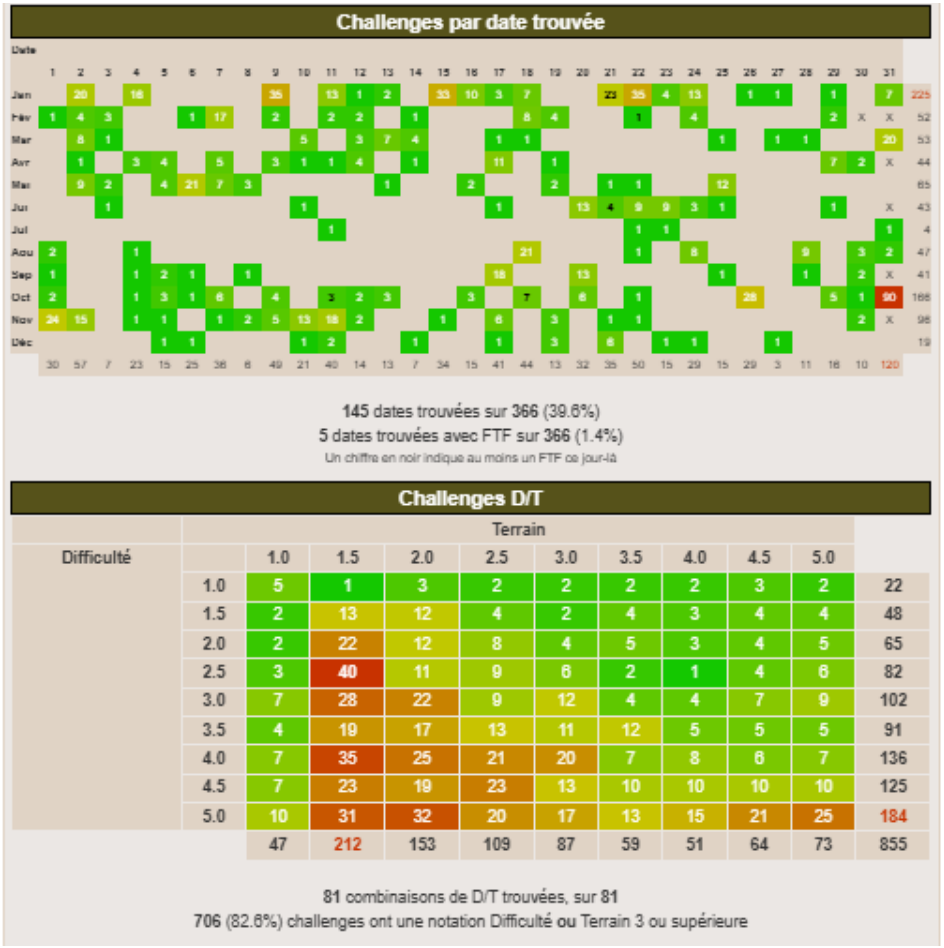


Figure 2: Avancée des challenges

#	Date[1]	GCCode	D/T			Nom de la géocache	
1	2018-09-20	GC4MTCR	3.0/2.0	62		Challenge Accro #11 - L'altitude	8693
2	2019-12-22	GC846Y4	5.0/4.5	77		D5 Challenge 11x7 types Belgique	40950
3	2020-09-29	GC8JW3F	5.0/1.5	53		*9*SDNFC//Challenge du 71	48773
4	2021-02-18	GC7QZ0N	2.5/2.0	10		Ravel 1 - Ile du bosquet - Challenge	35334
5	2021-10-26	GC9FPM8	4.5/5.0	46		24 Master Challenge Auslandsreise	63504
6	2021-11-11	GC44BCA	4.5/2.0	63		Challenge Latitude 47	17395
7	2021-12-06	GC8JNCC	4.5/1.5	81		Challenge - on p'tit toû dins l'Province di Nameur	64537
8	2021-12-06	GC8J6V4	5.0/2.0	86		Challenge Cache – The fisherman	64247
9	2022-01-11	GC57939	5.0/2.0	58		ABC-Multis (Challenge)	449
10	2022-02-06	GC93YZ8	2.5/2.0	42		#16 -Les 1 000'Art - Le Challenge des Milles	57401
11	2022-03-14	GC8JF2W	2.0/3.0	23		Zodiak # 190 R Rak C Cancer - Challenge	65198
12	2022-03-14	GC8JG34	3.0/2.5	23		Zodiak # 228 R Rak C Cancer - Challenge	65199
13	2022-03-14	GC8JW1C	3.0/1.5	75		Zodiak # 250 R Lew A Leo - challenge	65191
14	2022-03-14	GC8JW19	2.5/1.5	42		Zodiak # 251 R Lew A Leo - challenge	65190
15	2022-03-14	GC8JW15	1.5/1.5	17		Zodiakc # 252 R Lew A Leo - challenge	65189
16	2022-03-14	GC8JW13	2.0/1.5	21		Zodiak # 253 R Lew A Leo - challenge	65187
17	2022-03-14	GC8JW12	1.0/1.5	20		Zodiak # 254 R Lew A Leo - challenge	65178

Figure 3: Listes

1.3.2 Limites de Project-GC

- **Interface dense** : de nombreux tableaux et graphiques, peu lisibles sur mobile.
- **Personnalisation limitée** : les conditions de challenge sont imposées par la plateforme, sans possibilité pour l'utilisateur de les adapter ou d'interpréter différemment.
- **Absence de projection** : pas d'outils pour estimer la progression ou prévoir une date de complétion.
- **Accès restreint** : les fonctionnalités challenges nécessitent un abonnement premium.

1.3.3 Valeur ajoutée de GeoChallenge Tracker

- **Personnalisation** : chaque utilisateur définit ses propres tâches (via une grammaire AST), permettant différentes interprétations d'un même challenge.
- **Projections temporelles** : graphiques de progression avec estimation de date de complétion.
- **Ergonomie mobile-first** : affichage sous forme de cartes (OSM, clustering), filtrage simple, utilisation fluide en mobilité.
- **Gratuité et indépendance** : pas d'abonnement premium requis, basé sur APIs ouvertes et contributions de la communauté.

Avec cette analyse, il apparaît clairement que Project-GC fournit une base intéressante, mais limitée.
GeoChallenge Tracker se positionne comme un outil complémentaire, gratuit et personnalisable, répondant directement aux besoins exprimés par les géocacheurs.

1.4 Objectifs

Le projet **GeoChallenge Tracker** vise à fournir une application web mobile-first, conviviale et sécurisée. À partir de cette problématique et de l'analyse de l'existant, les fonctionnalités principales du projet peuvent être clairement formulées :

- importer ses listes de caches connues ou trouvées (GPX),
- détecter automatiquement les challenges présents dans les données importées,
- gérer ses challenges et de les exprimer sous forme d'un ensemble de tâches liées à des caches à trouver,
- consulter la progression détaillée par tâche et par challenge,
- visualiser une projection dans le temps de l'avancement vers la complétion,
- identifier les caches restantes permettant de progresser sur ses objectifs.

L'outil doit rendre lisible la progression statistique et projeter, à l'aide de tendances, une date estimée de complétion pour chaque challenge.

1.5 Périmètre fonctionnel et hors-périmètre

La phase initiale de développement étant limitée dans le temps, un périmètre fonctionnel initial suffisant a été défini, tout en anticipant des évolutions futures.

Inclus en V1 :

- Challenges : nombre de caches, D/T, type de caches.
- Import GPX C:Geo avec namespaces supportés : groundspeak, gsak, cgeo, topografix.
- Cartographie via OpenStreetMap avec *tile caching* et *marker clustering* pour gérer les zones à forte densité.
- Version mobile optimisée (le desktop reprend le même rendu).

Prévu en évolution :

- Export GPX des caches utiles.
- Types de challenges supplémentaires.
- Authentification OAuth.
- Multilingue.
- IA de détection : proposer des journées de géocaching optimisées en fonction de la zone, du moyen de transport et du réseau routier, pour maximiser la progression sur les challenges sélectionnés.

1.6 Sources de données et intégrations

La richesse fonctionnelle de l'application repose sur des données fiables et variées. Voici les sources exploitées et la manière dont elles sont intégrées.

- Fichiers GPX format C:Geo utilisant les namespaces :

  - `groundspeak` : http://www.groundspeak.com/cache/1/0/1
  - `gsak` : http://www.gsak.net/xmlv1/6
  - `cgeo` : http://www.cgeo.org/wptext/1/0
  - `topografix` : http://www.topografix.com/GPX/1/0
- Cartographie OpenStreetMap avec cache des tuiles et *marker clustering*.
- Modèle minimal : identifiant, coordonnées, type, D/T, attributs, dates de logs.
- Mise à jour incrémentale, pas de suppression/modification initialement.

1.7 Utilisateurs et rôles

L'application distingue deux types d'utilisateurs, correspondant à des rôles bien définis.

- **Utilisateur** : gère ses challenges, import GPX, visualisation progression.
- **Admin** : mêmes droits + futures prérogatives de modification/suppression.

1.8 Parcours utilisateur clés

Les parcours clés identifiés sont les suivants :

1. Inscription / validation email.
2. Import de fichiers GPX de caches et de caches trouvées.
3. Détection automatique des challenges à partir des données importées.
4. Acceptation ou rejet des challenges détectés.
5. Décomposition des challenges acceptés en tâches via une grammaire AST.
6. Consultation de la progression et proposition de caches utiles.

1.9 Indicateurs de succès (KPI)

Pour mesurer l'impact du projet, des indicateurs de succès ont été définis dès le départ.

- Nombre d'utilisateurs test.
- Taux d'import réussi.
- Nombre moyen de challenges suivis.
- Panel testeurs expérimentés (>20 000 caches).

1.10 Exigences non fonctionnelles

Au-delà des fonctionnalités visibles, certaines exigences techniques et qualitatives doivent être respectées pour assurer robustesse et pérennité.

Les exigences non fonctionnelles couvrent les aspects techniques, qualité et performance du système :

- **Sécurité** : authentification sécurisée, chiffrement des mots de passe, prévention des injections.
- **Confidentialité** : stockage des données utilisateur dans une base MongoDB distante sécurisée.
- **RGPD** : conformité à la législation française à la sortie publique.
- **Performance** : taille maximale de fichier GPX fixée à 20 Mo, upload possible en ZIP.
- **Compatibilité** : support des navigateurs modernes.
- **Accessibilité** : objectif WCAG 2.1 AA partiel.
- **Internationalisation** : français uniquement en V1.

1.11 Contraintes et dépendances

Le projet doit aussi composer avec un ensemble de contraintes techniques et organisationnelles, ainsi que des risques identifiés.

- Échéance : 19/09/2025.
- API de reverse geocoding (à définir).
- Hébergement Railway en développement.

1.12 Contraintes et risques

Contraintes principales :

- **Échéance** : 19/09/2025.
- **Cartographie** : respect des licences OpenStreetMap et gestion optimisée des appels (tile caching).
- **Hébergement** : environnement Railway en développement, avec MongoDB Atlas pour la base distante.
- **Performance** : taille maximale de fichier GPX fixée à 20 Mo, upload possible en ZIP.
- **Compatibilité** : support des navigateurs modernes, responsive et mobile-first.
- **Accessibilité** : objectif WCAG 2.1 AA partiel.
- **Internationalisation** : français uniquement en V1.

Risques identifiés et mesures de mitigation :

- **Volumétrie élevée** : limitation de la taille des GPX importés par appel, optimisation des parsings et de l'indexation MongoDB.
- **Changement de format GPX** : adaptation du module de parsing grâce à une architecture flexible et basée sur une grammaire AST.
- **Données d'altimétrie et localisation** :
  - **Risque** : dépendance à des APIs opendata externes (changement de format, indisponibilité, quotas).
  - **Mitigation** :
    * recours à des APIs publiques et non contractuelles → possibilité de changer de fournisseur ou d'en combiner plusieurs ;
    * mise en place d'une architecture permettant le remplacement ou la répartition multi-APIs ;
    * **respect strict des politiques de volumétrie et de fréquence** afin d'éviter tout blocage ;
    * stockage en base des données enrichies (altimétrie, commune) pour éviter les appels répétés.
- **Limites de licences cartographiques** : respect des règles OSM, mise en cache des tuiles et clustering des marqueurs.
- **Disponibilité des services externes** : stratégie de fallback et modularité pour maintenir le service même en cas d'indisponibilité partielle.

Références :

- Site officiel geocaching.com
- GPX namespaces :
  - http://www.groundspeak.com/cache/1/0/1
  - http://www.gsak.net/xmlv1/6
  - http://www.cgeo.org/wptext/1/0
  - http://www.topografix.com/GPX/1/0

1.13 GeoChallenge Tracker - Descriptif projet

GeoChallenge Tracker est une application web conçue pour la communauté des géocacheurs souhaitant aller au-delà de la simple recherche de caches en participant à des **challenges** thématiques. Elle fournit un environnement complet pour **définir, suivre et analyser** l'avancement de ces défis, tout en intégrant des fonctionnalités modernes de visualisation et d'automatisation.

L'application permet aux utilisateurs d'**importer leurs trouvailles au format GPX** (fichier ou archive ZIP). Les caches sont automatiquement reconnues et, le cas échéant, associées à des challenges existants. Des mécanismes d'auto-création de challenges à partir des caches importées facilitent la mise en route pour l'utilisateur. L'API expose également des fonctions de filtrage avancé des caches par type, taille, attributs, période de placement ou encore par périmètre géographique (bounding box ou rayon de recherche via un index 2dsphere).

Une fois les caches importées, l'utilisateur peut accéder à la liste de ses **UserChallenges**, suivre leur statut (pending, accepted, completed…), et gérer leurs tâches associées. Chaque challenge est défini sous forme d'arbres logiques (AST) décrivant les conditions à remplir : par exemple nombre minimal de caches d'un type donné, difficulté cumulée, altitude totale, ou combinaison de plusieurs critères. L'API évalue régulièrement la progression et génère des **snapshots** horodatés, permettant de visualiser l'évolution dans le temps comme une **série temporelle**. Des estimations de complétion sont calculées et présentées à l'utilisateur.

L'application ne se limite pas au suivi global : elle calcule aussi des **targets** (caches candidates à rechercher) pour maximiser les chances de réussite d'un challenge. Ces targets sont filtrables par proximité géographique (autour d'un point ou selon la localisation enregistrée de l'utilisateur), par score ou par pertinence, et peuvent être consultées challenge par challenge ou dans une vue consolidée.

Côté technique, GeoChallenge Tracker repose sur un backend en **FastAPI** couplé à **MongoDB Atlas**, un frontend moderne basé sur **Vue.js** et **Vite**, et des services conteneurisés via **Docker**. La cartographie est assurée par **OpenStreetMap** et les traitements incluent la mise à jour automatique des altitudes de caches. Des tests TDD et E2E garantissent la robustesse de l'ensemble.

En résumé, GeoChallenge Tracker apporte aux géocacheurs un **outil libre, moderne et puissant**, qui combine suivi personnalisé, recommandations intelligentes et visualisations géographiques pour relever des défis toujours plus ambitieux.

1.14 GeoChallenge Tracker - Project description

GeoChallenge Tracker is a web application designed for the geocaching community eager to go beyond simple cache hunting by taking part in thematic **challenges**. It offers a comprehensive environment to **define, monitor, and analyze** challenge progress, with modern visualization tools and automated workflows.

Users can **import their finds in GPX format** (single file or ZIP archive). Imported caches are automatically recognized and, when relevant, linked to existing challenges. The system also supports auto-creation of challenges based on imported caches, making onboarding straightforward. The API provides advanced filtering options to search caches by type, size, attributes, placement date, or geographical scope (bounding box or radius search powered by a 2dsphere index).

Once caches are loaded, users can explore their list of **UserChallenges**, track their status (pending, accepted, completed, etc.), and manage the associated tasks. Each challenge is defined as a logical tree (AST) representing the rules to meet—such as a minimum number of caches of a certain type, cumulative difficulty thresholds, altitude sums, or combinations of multiple conditions. The API evaluates progress on a regular basis and produces **timestamped snapshots**, enabling users to view their evolution as a **time series**. Estimated completion dates are also provided to help planning.

Beyond overall progress tracking, the application computes **targets** (candidate caches to look for) to maximize a user's chance of completing a challenge. These targets can be filtered by geographical proximity (around a given point or based on the user's last recorded location), sorted by score or relevance, and displayed either per challenge or in a consolidated view.

From a technical perspective, GeoChallenge Tracker is built on a **FastAPI** backend with **MongoDB Atlas**, a modern frontend using **Vue.js** and **Vite**, and a containerized deployment with **Docker**. Maps are rendered with **OpenStreetMap**, while elevation data is updated automatically. The project relies on TDD and E2E testing to ensure reliability and robustness.

In short, GeoChallenge Tracker delivers a **modern, open, and powerful tool** for geocachers, combining personalized tracking, intelligent recommendations, and geographical visualizations to help them take on ever more ambitious challenges.

2 Compétences mobilisées

2.1 Développer une application sécurisée

La première catégorie de compétences concerne la capacité à développer une application sécurisée, depuis l'installation de l'environnement de travail jusqu'à la gestion du projet.

- **Installer et configurer son environnement de travail en fonction du projet**
  - Mise en place d'un environnement Dockerisé pour assurer la reproductibilité.
  - Configuration du backend avec FastAPI et du frontend avec Vue.js.
- **Développer des interfaces utilisateur**
  - Interfaces responsives construites avec Vue 3 et Tailwind CSS.
  - Intégration de composants cartographiques interactifs (Leaflet) avec clustering et *tile caching*.
  - Optimisation des performances dans une démarche de *green computing via performance optimizing*.
- **Développer des composants métier**
  - Parsing de fichiers GPX multi-namespaces (C:Geo, Groundspeak, GSAK, Topografix).
  - Conception et manipulation de grammaires AST pour exprimer et traiter les conditions des challenges.
  - Calculs de projections d'avancement et génération de séries temporelles.
  - Moteur de filtrage par condition et gestion des doublons.
- **Contribuer à la gestion d'un projet informatique**
  - Suivi sur GitHub (issues, milestones, kanban).
  - Organisation du planning et intégration continue via GitHub Actions.

2.2 Concevoir et développer une application sécurisée organisée en couches

La deuxième catégorie porte sur la conception d'une application structurée, respectant une architecture en couches et intégrant une base de données adaptée.

- **Analyser les besoins et maquetter une application**
  - Rédaction du cahier des charges (expression du besoin).
  - Réalisation de wireframes des vues principales.
- **Définir l'architecture logicielle d'une application**
  - Architecture découpée : backend FastAPI (API REST) et frontend Vue.js.
  - Séparation stricte des responsabilités (contrôleurs, composants métier, données).
- **Concevoir et mettre en place une base de données relationnelle**
  - Transposition aux besoins du projet avec MongoDB (NoSQL).
  - Modélisation documentaire adaptée à la variabilité des caches et de leurs attributs.
  - Indexation pour requêtes multi-critères et optimisation de la volumétrie.
- **Développer des composants d'accès aux données SQL et NoSQL**
  - Accès aux données utilisateur via MongoDB (CRUD, indexation, agrégations).
  - Sécurisation des accès et séparation stricte par utilisateur.

2.3 Préparer le déploiement d'une application sécurisée

Enfin, les compétences mobilisées concernent la préparation du déploiement et la mise en production dans une démarche DevOps.

- **Préparer et exécuter les plans de tests d'une application**
  - Tests unitaires backend (Pytest) avec approche TDD.
  - Tests end-to-end frontend (Cypress).
  - Tests fonctionnels de bout en bout.
- **Préparer et documenter le déploiement d'une application**
  - Fichiers `Dockerfile` et `docker-compose.yml` pour la conteneurisation.
  - Documentation des variables d'environnement et procédures de lancement.

- **Contribuer à la mise en production dans une démarche DevOps**
  - CI/CD via GitHub Actions (tests, build, déploiement automatisé).
  - Déploiement en environnement Railway et base MongoDB Atlas.
  - Conteneurisation pour homogénéité dev/prod et réduction des erreurs humaines.

2.4 Technologies utilisées

Ces compétences se traduisent concrètement par l'utilisation des technologies suivantes :

- **Backend** : FastAPI (Python).
- **Frontend** : Vue.js 3, Tailwind CSS.
- **Base de données** : MongoDB (NoSQL).
- **Cartographie** : Leaflet + OpenStreetMap (clustering, *tile caching*).
- **Conteneurisation / DevOps** : Docker, docker-compose, GitHub Actions.
- **Tests** : Pytest (backend), Cypress (frontend).
- **Sécurité** : JWT, chiffrement des mots de passe, validation des entrées.
- **APIs externes** : services opendata (altimétrie, localisation).

3 Présentation client

3.1 Client principal

Avant de détailler les attentes précises, il est essentiel de présenter le public cible auquel l'application s'adresse.

La communauté visée regroupe des géocacheurs passionnés, pour qui la recherche de caches est à la fois un loisir, un défi personnel et un moyen de dépassement de soi. Certains voient dans les *challenges* un « jeu dans le jeu » : un niveau supplémentaire qui pousse à explorer davantage, à varier les types de caches et à repousser ses limites. Ces challenges imposent souvent des critères exigeants (difficulté/terrain, type de cache, localisation, attributs spécifiques) et s'étalent sur des périodes longues, nécessitant rigueur et organisation.

Pour les géocacheurs les plus investis, la gestion de ces contraintes devient rapidement un défi en soi. Les solutions existantes, comme certaines plateformes spécialisées, offrent des outils puissants mais souvent payants (ex. Project-GC en version premium). Sans outil adapté, le suivi d'une telle quantité d'informations relève d'une tâche chronophage et complexe, avec un risque élevé d'erreurs ou d'oubli.

GeoChallenge Tracker se positionne comme une solution accessible et intuitive, permettant à ces utilisateurs d'optimiser leur expérience sans multiplier les feuilles de calcul ou s'appuyer uniquement sur leur mémoire.

3.2 Attentes utilisateurs

De cette analyse des pratiques, ressortent plusieurs attentes fortes de la part des utilisateurs :

- un outil simple d'utilisation, mais puissant dans le traitement et le croisement de données,
- un suivi automatisé et visuel de leur progression,
- la possibilité d'identifier les caches qui permettent d'avancer sur plusieurs challenges à la fois,
- une cartographie efficace permettant de filtrer, de regrouper et de localiser les caches pertinentes,
- un respect strict de la confidentialité et de la sécurité de leurs données.

Afin de garantir que l'outil réponde réellement aux besoins, un **pool de testeurs expérimentés** est en cours de recrutement. Il inclut des géocacheurs aux profils variés :

- plusieurs joueurs, dont certains ayant trouvé **plus de 50 000 caches**, classés parmi les premiers au niveau national,
- des figures reconnues de la communauté (ex. membres actifs, auteurs, géocacheurs classés dans le top 5 national),
- des profils diversifiés pour couvrir différents styles de jeu (grands voyageurs, spécialistes des caches T5, optimiseurs rationnels, etc.).

Ce panel permettra de confronter l'application à des pratiques intensives et variées, et de valider sa pertinence auprès de la communauté cible.

3.3 Personas

Pour mieux incarner ces attentes, plusieurs profils types (personas) permettent d'illustrer la diversité des besoins.

3.3.1 Persona 1 : Le passionné des grands défis

- **Âge** : 45 ans
- **Profession** : Cadre supérieur
- **Profil** : Géocacheur depuis plus de 15 ans, engagé dans des challenges de grande ampleur, impliquant plusieurs régions ou pays. Planifie ses voyages en fonction des caches à trouver.
- **Besoins** : Un suivi précis de l'avancement sur des objectifs étendus dans le temps et l'espace, avec des projections claires. Centralisation des données éparses provenant de multiples zones géographiques.

3.3.2 Persona 2 : Aïcha, la monitrice d'escalade

- **Âge** : 34 ans
- **Profession** : Monitrice d'escalade
- **Profil** : Fan de caches T5, adore les défis nécessitant des compétences techniques en hauteur ou en spéléologie. Voyage souvent et recherche l'adrénaline.
- **Besoins** : Filtrer efficacement les caches selon les attributs extrêmes, voir rapidement ce qui reste à faire pour valider un challenge T5.

3.3.3 Persona 3 : L'optimisateur rationnel

- **Âge** : 39 ans
- **Profession** : Ingénieur en logistique
- **Profil** : Participe à de nombreux challenges simultanément, mais privilégie l'efficacité. Cherche à réduire les déplacements inutiles en identifiant des caches qui permettent d'avancer sur plusieurs challenges à la fois.
- **Besoins** : Un outil capable de croiser les critères pour trouver les "caches multi-bénéfices", et de calculer le meilleur compromis entre distance parcourue et progression réalisée.

3.4 Pool de testeurs

On peut retrouver l'ensemble des caractéristiques de ces personas en s'appuyant sur un panel de testeurs représentatifs de la communauté, constitué pour valider l'outil en conditions réelles.

Il regroupe des géocacheurs aux profils variés, allant de joueurs passionées à des figures reconnues du classement national. Ces retours permettent de confronter l'outil à des cas d'usage concrets et exigeants.

Pseudo	Caches trouvées	Caches posées	Rang national
Almani06	10 921	42	–
Arnokovic	86 909	303	4 (Fr)
audeclar	9 351	304	–
falbala20220	17 336	152	335 (Fr)
Kidoulo	10 932	96	–
le sudiste	35 203	46	70–75 (eq Fr)
MLRFamily	6 279	7	–
Orchidée83	4 842	9	–
oTo66	4 382	89	–
Phiphi13	11 290	292	–

Ce panel de testeurs apporte :

- la vision de **joueurs très expérimentés** (dont un classé **n°4 en France** en nombre de caches trouvées),
- l'expertise de **poseurs prolifiques**,
- et l'expérience de profils plus **familiaux ou intermédiaires**, garantissant une couverture diversifiée.

Ces retours permettent de vérifier à la fois la **pertinence fonctionnelle**, la **fiabilité technique** et l'**ergonomie** de l'outil.

4 Spécifications fonctionnelles

4.1 Fonctionnalités principales

Les fonctionnalités principales de GeoChallenge Tracker couvrent l'ensemble du cycle d'utilisation, de l'authentification à la visualisation des projections de progression.

4.1.1 Authentification et sécurité

La sécurité est au cœur du projet, avec un système d'authentification robuste et des mécanismes de protection contre les attaques classiques.

- Inscription avec règles strictes de mot de passe (longueur, majuscules, minuscules, chiffres et caractères spéciaux).
- Stockage des mots de passe en **bcrypt** avec sel variable.
- Validation par email avec lien unique et limite temporelle de validité.
- Connexion sécurisée avec **JWT** et **refresh token** (durées distinctes).
- Gestion des rôles (`user`, `admin`) avec séparation stricte des droits.
- Prévention des injections : utilisation exclusive des méthodes sûres de MongoDB (`find_one`, `find_many`, `insert_many`, etc.).

4.1.2 Gestion des challenges

Les challenges constituent la brique centrale de l'application. Leur gestion suit un processus en plusieurs étapes, de la détection automatique à l'évaluation personnalisée.

- **Détection automatique** : lors de l'import GPX, les caches portant l'attribut *challenge* sont automatiquement repérées.
- **Référentiel de challenges (`challenges`)** : ces caches sont insérées/actualisées dans la collection `challenges` (identifiant GC, métadonnées minimales, liens, etc.).
- **Liste par utilisateur (`userChallenges`)** : chaque utilisateur dispose de sa propre liste dérivée du référentiel :
  - **acceptation / rejet** d'un challenge détecté ;
  - **personnalisation** : définition des **tâches** (conditions) par l'utilisateur via un éditeur basé sur une **grammaire AST** ; ces tâches peuvent **différer d'un utilisateur à l'autre** pour un même challenge ;
  - **notes** et méta associées au challenge.
- **Moteur d'évaluation** :
  - les **tâches définies par l'utilisateur** sont évaluées en **faisant correspondre** leurs conditions avec les **caractéristiques des caches** en base (trouvées ou non selon les logs) ;
  - calcul des **progress points** et de la progression globale à partir des tâches.
- **Traçabilité et mises à jour** : recalcul automatique après nouvel import ou modification des tâches.

Note : Chaque tâche est exprimée sous forme d'AST et compilée en requête Mongo. **Exemple (JSON) — deux tâches personnalisées pour un même challenge :**

```
{
  "tasks": [
    {
      "title": "Série D/T équilibrée",
      "expression": {
        "kind": "and",
        "nodes": [
          { "kind": "difficulty_between", "min": 2.0, "max": 3.5 },
          { "kind": "terrain_between",    "min": 2.0, "max": 3.5 },
          { "kind": "type_in", "types": [ { "cache_type_code": "traditional" }, { "cache_type_code
    ↪ ": "mystery" } ] }
        ]
      }
    },
    {
      "title": "Picnic en altitude",
      "expression": {
        "kind": "and",
        "nodes": [
          { "kind": "attributes", "attributes": [ { "code": "picnic", "is_positive": true } ] },
```

```
          { "kind": "aggregate_sum_altitude_at_least", "min_total": 5000 }
        ]
      }
    }
  ],
  "uc_logic": { "kind": "and", "task_ids": ["<id-task-1>", "<id-task-2>"] }
}
```

4.1.3 Import de caches

L'import des caches repose sur la lecture de fichiers GPX enrichis de plusieurs namespaces, garantissant compatibilité et exhaustivité.

- Support des fichiers GPX C:Geo avec multi-namespaces :
  - `groundspeak` : http://www.groundspeak.com/cache/1/0/1
  - `gsak` : http://www.gsak.net/xmlv1/6
  - `cgeo` : http://www.cgeo.org/wptext/1/0
  - `topografix` : http://www.topografix.com/GPX/1/0

Parsing GPX pur. GPXCacheParser lit gpx/groundspeak/gsak, extrait les champs clés et normalise les attributs (id, is_positive, name).

```
# Extrait : lecture des waypoints finaux et extraction des champs utiles
for wpt in tree.xpath("//gpx:wpt", namespaces=self.namespaces):
    cache_elem = wpt.find("groundspeak:cache", namespaces=self.namespaces)
    if cache_elem is None:
        continue

    cache = {
        "GC": self.find_text_deep(wpt, "gpx:name"),
        "title": self.find_text_deep(wpt, "gpx:desc"),
        "latitude": float(wpt.attrib["lat"]),
        "longitude": float(wpt.attrib["lon"]),
        "cache_type": self.find_text_deep(wpt, "groundspeak:type"),
        ...
        "description_html": self.sanitizer.clean_description_html(
            self.find_text_deep(wpt, "groundspeak:long_description")
        ),
        ...
        "placed_date": self.find_text_deep(wpt, "gpx:time"),
        "found_date": self.find_text_deep(wpt, "gsak:UserFound"),
        "attributes": self._parse_attributes(cache_elem),
    }
    self.caches.append(cache)
```

- Distinction entre caches connues et caches trouvées.
- Import incrémental, sans suppression automatique des caches déjà stockées.
- Détection et gestion des doublons.

4.1.4 Stockage des données

Toutes les données sont centralisées et structurées dans MongoDB, avec une séparation stricte entre utilisateurs.

- Stockage centralisé en MongoDB (caches, challenges, userChallenges, AST).
- Séparation stricte des données par utilisateur.
- Indexation et requêtes optimisées pour recherche rapide.

4.1.5 Cartographie

La cartographie permet de visualiser efficacement les caches et challenges, tout en restant performante sur mobile.

- Carte affichée par challenge.
- Filtrage par condition et distance.
- Recentrage manuel possible.
- Clustering automatique des marqueurs (*marker clustering*) pour zones à forte densité.

- Bibliothèque cartographique avec gestion du *tile caching* (OpenStreetMap).

4.1.6 Statistiques et projections

L'utilisateur dispose d'outils de suivi statistique avancés, permettant à la fois une vision d'ensemble et une estimation des échéances.

- Courbes cumulées de progression par tâche et par challenge.
- Stockage et datation de chaque **progress point**.
- Affichage simultané des courbes de chaque tâche et de la courbe globale.
- Estimation de la date de complétion (projection de tendance).
- Possibilité de filtrer l'affichage par courbe.

4.2 Évolutions futures

Plusieurs évolutions sont envisagées pour enrichir l'outil et anticiper les besoins futurs de la communauté.

- Affichage multi-challenges sur une même carte (optimisation des itinéraires et croisements de conditions).
- IndexedDB côté client pour un mode offline plus performant.
- Export GPX, CSV ou PDF des caches pertinentes.
- Authentification OAuth (Geocaching.com).
- IA d'assistance : proposition d'itinéraires optimisés selon zone, moyens de transport, réseau routier et progression.
- Optimisation des flux entre MongoDB et le client pour réduire la latence.
- Multilingue.

4.3 Contraintes techniques

Enfin, certaines contraintes techniques fixent les limites de l'application et orientent ses choix d'implémentation.

- Interface responsive, mobile-first.
- Performance mobile optimisée.
- Taille maximale de fichier GPX : 20 Mo (zip accepté pour optimisation transfert).
- Nombre maximum de challenges suivis par utilisateur : 200.
- Objectif d'accessibilité : WCAG 2.1 AA partiel.
- **Métriques de performance prévues** :
  - Import massif de **10 000 caches** en <30 sec.
  - Gestion fluide de **200 challenges actifs** par utilisateur.
  - Affichage cartographique interactif de **5 000 points** avec clustering en <2 sec.

5 Charte graphique

Dans la conception de l'interface, une approche **mobile first** a été privilégiée.
Ce choix repose sur plusieurs arguments. Tout d'abord, le mobile first conduit naturellement à une interface centrée sur le **contenu** et une **ergonomie simple et efficace**, ce qui correspond à l'objectif du projet. Cette orientation a été pensée en opposition à certains outils existants, comme *Project-GC*, dont le contenu est riche mais présenté de manière très dense : abondance de tableaux, polices réduites, couleurs juxtaposées dans de petites cases. Le résultat est puissant mais peu lisible sur mobile, et difficilement exploitable en situation de mobilité.

Ensuite, l'application est amenée à être utilisée en **itinérance**, notamment pour la consultation cartographique lors de déplacements. Une ergonomie pensée pour le bureau (desktop) n'aurait donc pas été adaptée. Au contraire, l'interface mobile first assure une **utilisation fluide en contexte de terrain**, tout en restant **parfaitement utilisable sur desktop**. De plus, il est toujours plus aisé d'ajouter des **media queries** pour enrichir l'expérience sur grand écran que de tenter de linéariser une interface multi-colonnes afin de la rendre utilisable sur smartphone.

Le design graphique repose volontairement sur une base **sobre et lisible**. Les polices choisies sont exclusivement **sans-serif**, le contraste entre texte et arrière-plan est maximal, et des nuances de gris ainsi que des variations de taille viennent hiérarchiser certains éléments. Ce travail permet une lecture confortable dans la majorité des situations. Une **passe d'accessibilité dédiée** pourra être envisagée dans une version ultérieure, afin de prendre en compte des besoins spécifiques (daltonisme, contrastes renforcés, lecteurs d'écran, etc.).

La **navigation** a été conçue pour optimiser l'espace : des **menus dépliants** permettent de libérer la surface utile, tandis que des **icônes thématiques par section** facilitent le repérage et la mémorisation visuelle des fonctionnalités. Cet équilibre entre sobriété et guidage visuel constitue un gage d'ergonomie pour des utilisateurs variés, du joueur occasionnel au géocacheur expérimenté.

Enfin, le **design épuré** retenu a aussi une incidence sur les performances. Il limite le nombre d'éléments graphiques et réduit donc les **temps de chargement**, la **bande passante consommée**, et même le **temps de mise en page (layout)** dans le navigateur. Cela correspond à une démarche naturellement plus efficiente sur le plan environnemental : un premier pas vers le **green IT**. Si la gestion efficace de la cartographie (notamment via le **tile caching**) constitue déjà un levier important, d'autres pistes d'optimisation (caching applicatif, traitement côté client mieux maîtrisé) pourront être explorées dans la suite du projet. Le design évite aussi toute lourdeur inutile : **aucune image lourde** n'est utilisée et les animations sont réduites au strict minimum.

Le **nom** et le **logo** de l'application ont fait l'objet d'une réflexion spécifique. Le nom *GeoChallenge Tracker* exprime de manière explicite l'objectif du logiciel : aider l'utilisateur à suivre sa progression dans des challenges géocaching. Il se simplifie facilement en *GC Tracker*, une abréviation à la fois courte et parlante. L'acronyme *GC* résonne immédiatement auprès de la communauté, puisqu'il est déjà largement utilisé pour désigner *geocaching* ou *geocache*.

Le logo reprend la base d'un **marqueur de position**, symbole universel de la géolocalisation, afin de rappeler immédiatement le contexte de l'activité. Plusieurs alternatives avaient été envisagées (escalier stylisé, coupe sportive, médaille, badge, ou encore dégradé de couleur), mais elles ont été écartées : trop complexes pour rester lisibles à petite taille, ou trop voyantes pour conserver une interface sobre. La solution finalement retenue repose sur un **histogramme intégré dans le marqueur**, exprimant à la fois la **notion de progression** et celle de **succès**. Les trois barres colorées montantes (rouge → jaune → vert) traduisent visuellement plusieurs idées : le **changement de statut** (échec → progression → réussite), l'**augmentation du taux de réalisation**, et la symbolique d'un **escalier**, associée à l'idée d'avancement et d'accomplissement. L'usage des couleurs s'appuie sur des codes universels, proches de ceux des feux de circulation ou des indicateurs sportifs, ce qui renforce l'immédiateté de la compréhension.

Ainsi, l'ensemble des choix graphiques répond à un double objectif : proposer une **interface claire et efficace en mobilité**, tout en véhiculant une **identité visuelle forte** qui parle directement à la communauté geocaching.

5.1 État non loggé

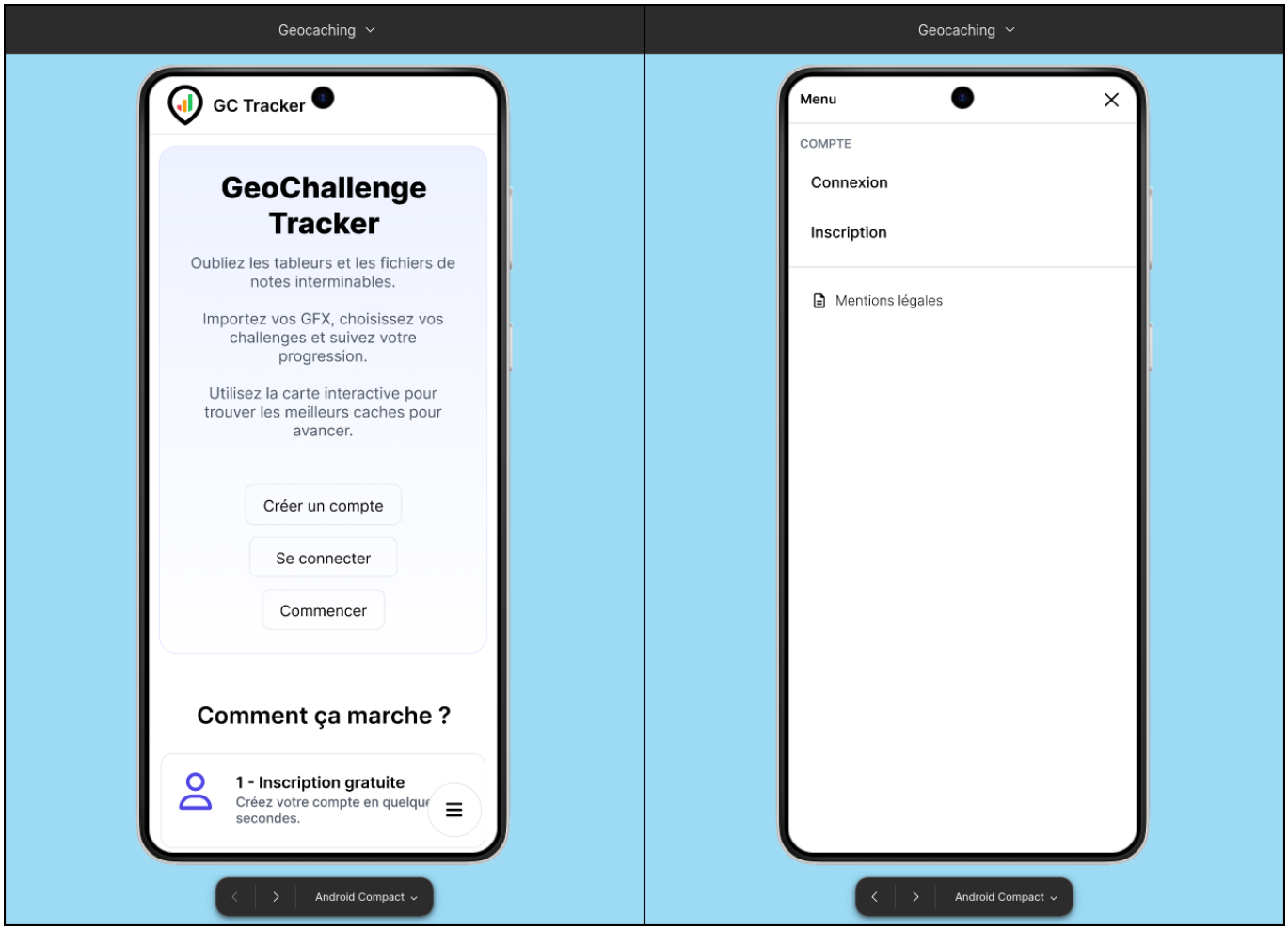


Figure 4: Accueil+Menu non loggé

L'écran d'accueil, accessible sans connexion, sert avant tout de **teaser pour les géocacheurs**. Le haut de page met en avant la **simplicité du logiciel**, sa capacité à **remplacer avantageusement l'existant**, ainsi que son caractère **interactif** et l'intérêt de sa **cartographie**. Le menu haut propose immédiatement les **actions essentielles** : créer un compte, se connecter, ou commencer. Le contenu principal illustre les **étapes d'utilisation typiques**, avec une explication succincte de chaque phase. Enfin, le **menu, volontairement réduit**, ne contient que les liens connexion / inscription, ainsi que les *mentions légales*, qui doivent rester *disponibles en toutes circonstances*.

5.2 Register / Login

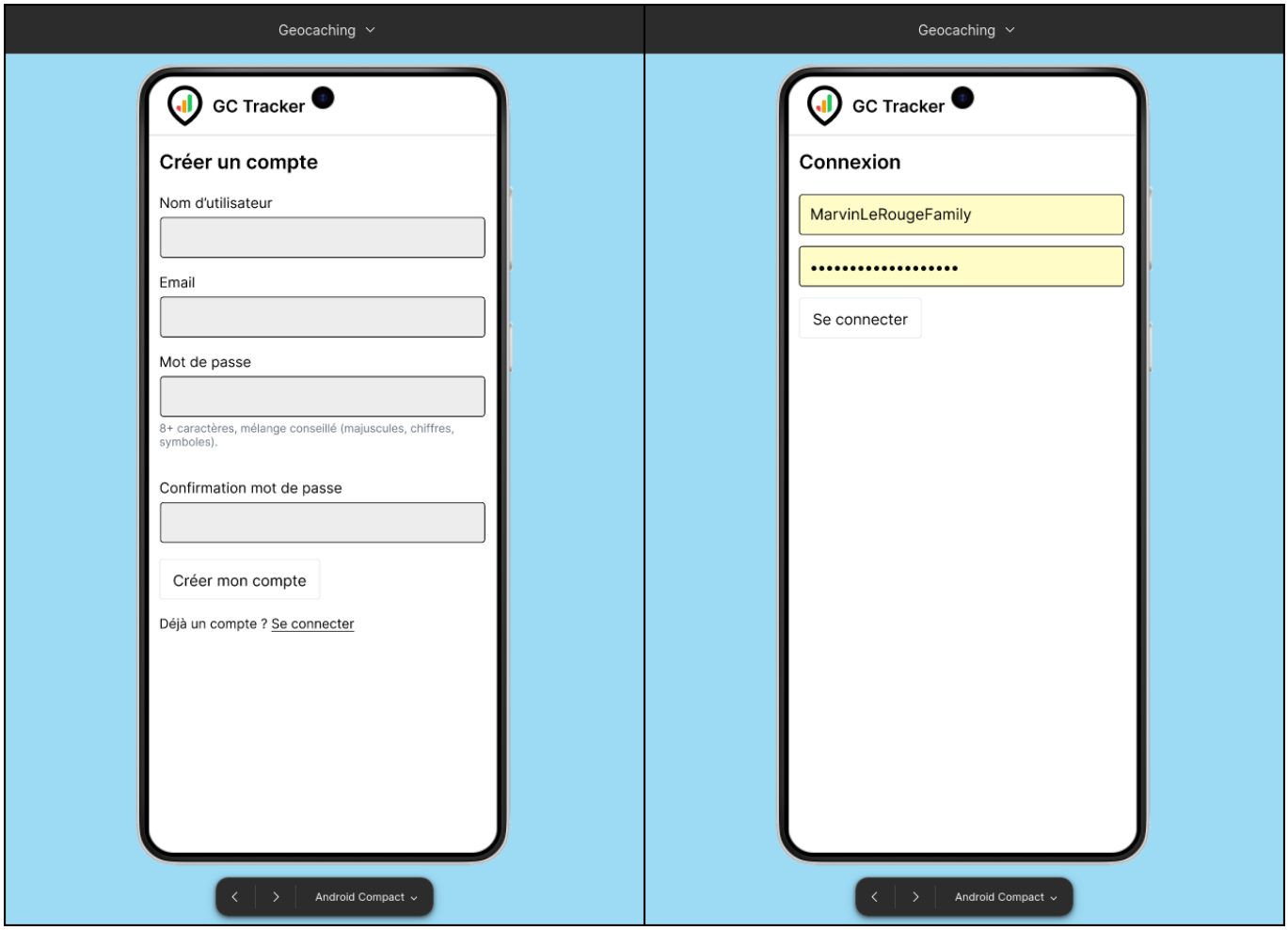


Figure 5: Register / Login

Les écrans d'inscription et de connexion suivent volontairement une **approche minimaliste**. Leur objectif est d'aller droit au but, conformément aux bonnes pratiques du domaine. Le formulaire d'inscription demande une confirmation du mot de passe, afin de **réduire les risques d'erreur** et d'**éviter la frustration** liée à un premier échec de connexion. La **sobriété visuelle** met en avant la **lisibilité** et la **fluidité** du parcours utilisateur.

5.3 État loggé (partie 1)

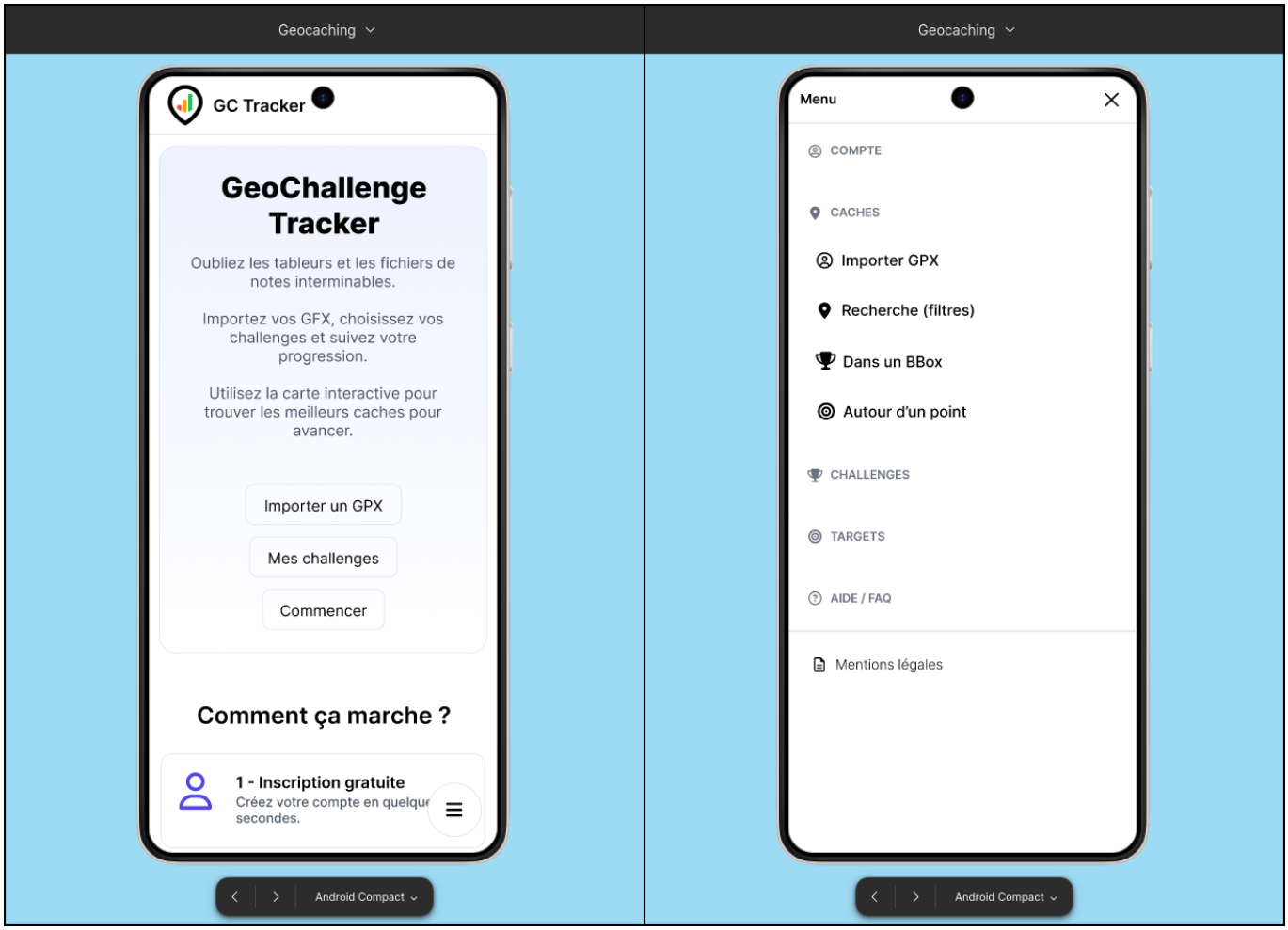


Figure 6: Accueil + Menu loggé

5.4 Page d'accueil - Contenu complet

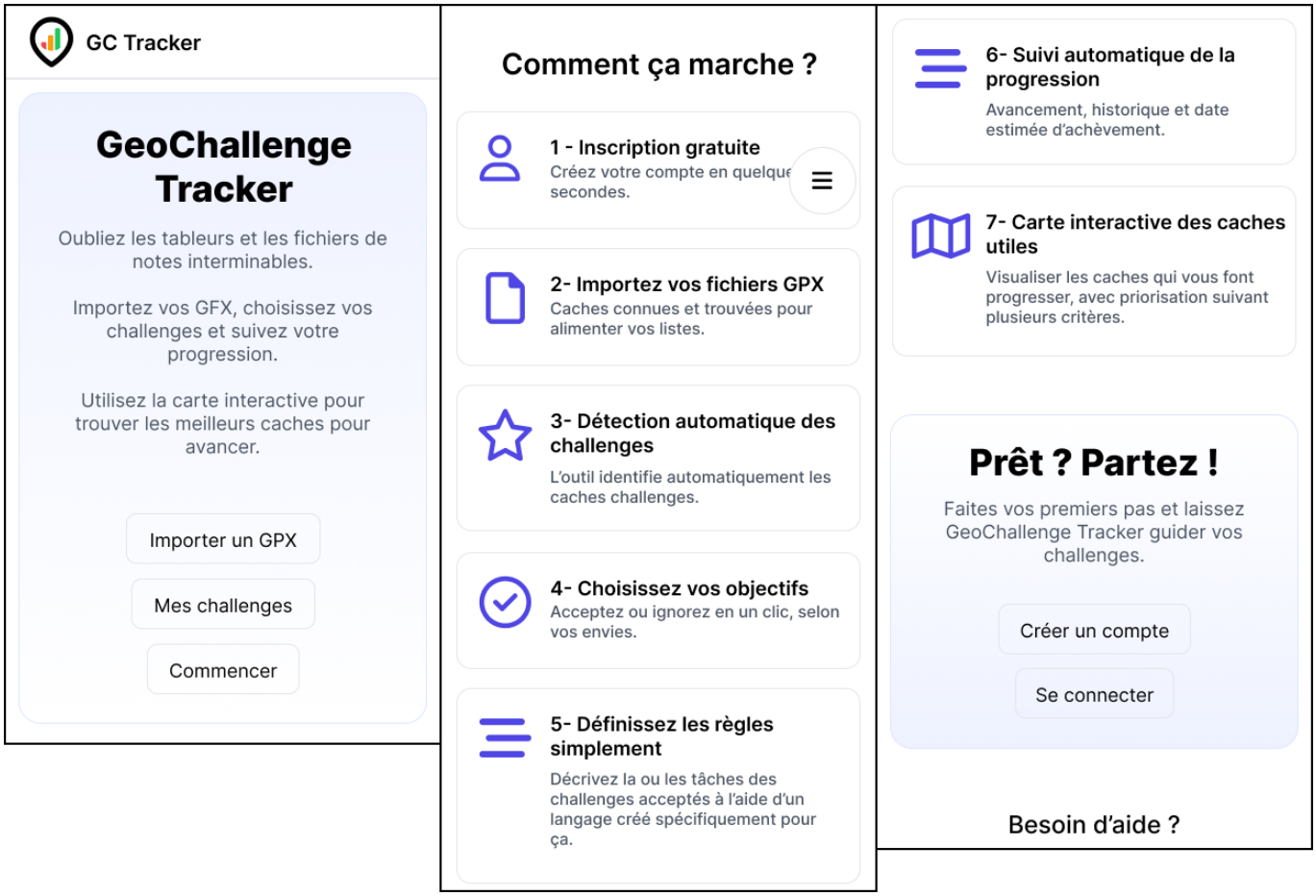


Figure 7: Accueil loggé - Contenu complet

La page d'accueil en version complète met en avant une **vision panoramique des fonctionnalités** disponibles. Le design conserve une structure claire, tout en affichant davantage d'éléments contextualisés pour l'utilisateur connecté. L'objectif est de fournir une **vue globale** qui associe lisibilité et exhaustivité, sans surcharger visuellement la page. Cette maquette met en lumière la **progression naturelle du parcours utilisateur**, depuis l'accès invité jusqu'à l'exploration complète des fonctionnalités.

En conclusion, la conception graphique de GeoChallenge Tracker illustre une démarche pragmatique : partir des besoins des utilisateurs, proposer une interface simple et claire, et s'assurer que l'expérience reste fluide sur mobile comme sur desktop. L'usage de maquettes Figma a permis de matérialiser ces choix très tôt, en validant les parcours essentiels sans chercher à couvrir exhaustivement toutes les pages de l'application.

Cette sobriété va de pair avec une réflexion sur les performances et l'impact environnemental. Dans une logique de **green IT**, toutes les ressources graphiques ont été optimisées, depuis les maquettes elles-mêmes jusqu'au logo en SVG, afin de réduire la taille des fichiers et d'accélérer leur affichage. L'ensemble contribue à un rendu plus léger et donc plus respectueux des contraintes de bande passante et de consommation.

Ces choix traduisent une volonté de concilier **ergonomie, efficacité et responsabilité**, en offrant à la communauté des géocacheurs un outil accessible, lisible et durable, conçu dès l'origine avec une attention particulière portée à la simplicité et à la performance.

L'ensemble du prototype est consultable en ligne, sur Figma.

6 Architecture logicielle

6.1 Vue d'ensemble (Contexte)

Cette première vue présente le contexte global de l'application et les principales briques qui la composent.

- Application **web mobile-first** : frontend Vue communiquant avec une **API REST** FastAPI.
- Données persistées dans **MongoDB Atlas** (modèle documentaire).
- Intégrations externes : **OpenStreetMap** (tuiles), **APIs OpenData** (altimétrie, commune), **SMTP** (validation email).

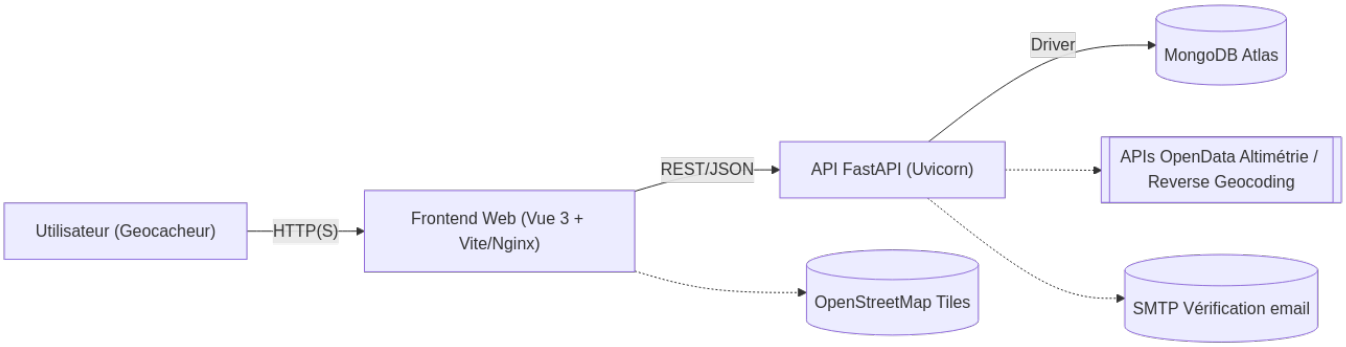


Figure 8: Diagramme de contexte général

6.2 Vue conteneurs (environnements Dev/Prod)

L'architecture varie légèrement entre développement et production, mais repose toujours sur une séparation claire des services.

6.2.1 Dev (docker-compose)

- **frontend** : Vite (HMR) sur `5173/tcp`.
- **backend** : FastAPI/Uvicorn sur `8000/tcp`.
- **MongoDB** : Atlas (DB managée) — accès via variables d'environnement.

6.2.2 Prod

- **frontend** : build statique servi par **Nginx**.
- **backend** : Uvicorn derrière reverse proxy (Nginx/ingress selon hébergeur).
- **CI/CD** : GitHub Actions (tests → build images → déploiement).

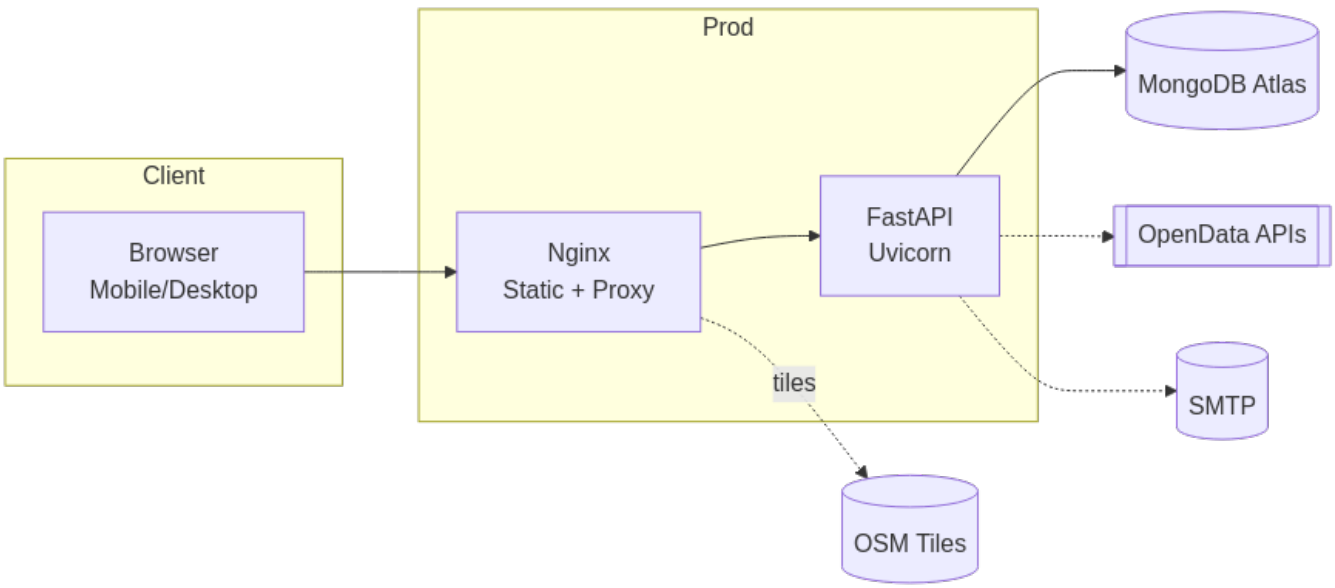


Figure 9: Diagramme des conteneurs (déploiement dev/prod)

6.3 Composants backend (modules principaux)

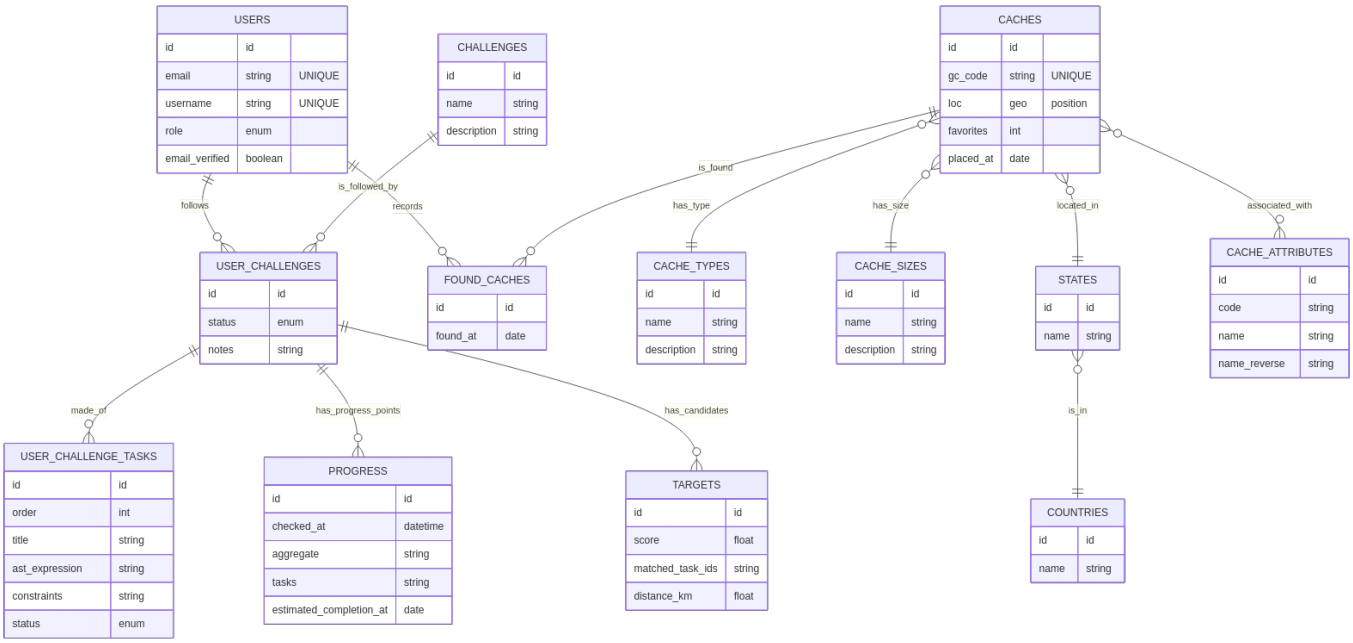
Le backend est structuré en modules cohérents, correspondant aux routes, aux services métiers et aux composants techniques de base.

- **API / Routes**
  - `auth` : inscription, login, refresh, vérification email, renvoi code.
  - `caches` : upload GPX/ZIP, recherche par filtres, BBox / rayon, get par GC/ID.
  - `caches_elevation` : backfill altimétrie (admin).
  - `challenges` : (re)construction depuis caches *challenge* (admin).
  - `my-challenges` : lister/synchroniser/patcher mes *UserChallenges*.
  - `my-challenge-tasks` : lire/remplacer/valider les **tâches** d'un *UserChallenge*.
  - `my-challenge-progress` : lire/évaluer snapshots de progression.
  - `targets` : calculer/lister/effacer les **targets** (caches utiles).
  - `my_profile` : localisation utilisateur (get/put).
  - `maintenance` : endpoints réservés (diagnostic/maintenance).
- **Services**
  - `gpx_importer` : parsing GPX multi-namespaces, upsert caches, dédoublonnage.
  - `challenge_autocreate` : détection *challenge caches* → upsert `challenges`.
  - `user_challenges` : liste par utilisateur, sync `pending`.
  - `user_challenge_tasks` : validation/PUT complet des tâches (AST + contraintes).
  - `progress` : évaluation, snapshots, projections (courbes cumulées, estimations).
  - `targets` : agrégation caches satisfaisant ≥1 tâche, scoring, filtrage géo.
  - `elevation_retrieval` : backfill altimétrie (quotas, pagination, dry-run).
  - `providers` : adaptateurs vers APIs OpenData (altimétrie, reverse geocoding).
  - `referentials_cache` : référentiels (types, tailles, attributs, pays/états).
  - `query_builder` : construction de filtres Mongo à partir de règles/AST.
- **Core / Infra**
  - `core/settings` : configuration (env vars, secrets).
  - `core/security` : JWT + refresh, rôles `user`/`admin`, règles password, bcrypt.
  - `db/mongodb` : connexion, index, seeds.

6.4 Modèle logique des données (MongoDB)

Le modèle de données, implémenté dans MongoDB, est conçu pour représenter les entités principales du geocaching et leurs relations. On peut par exemple ici observer la relation entre les entités **Caches** et **Cache_Attributes**, indiquant qu'une cache peut avoir 0 à n attributs, et qu'un cache_attribute donné peut exister dans 0 à n caches, ce type de relation se traduisant potentiellement par une table de jointure (si on souhaite conserver l'ensemble des couples).
On y voit également la relation tripartite des entités Users - User_Challenges - Challenges. En effet, User_Challenges est basiquement l'association d'un User et d'un Challenge ; mais, dans a mesure où cette relation est porteuse de contenu propre (ici, un statut et des notes), elle sera nécessairement exprimée par une table de jointure.
L'attribut **loc** de l'entité **caches** est ici d'un type abstrait **position** qui représente la notion de position géographique.

Pour des questions d'ordre pratique, le schema ci-dessous est un extrait de la structure de la base de données, et non son intégralité.



6.5 Modèle physique des données (MongoDB)

On peut ici observer le MPD, qui est une concrétisation du MLD pour un moteur choisi de base de données. Dans notre cas, on constate par exemple que les id sont sous forme de **uuid** (ce qui est le type réel sous-jacent des ObjectId MongoDB), que toutes les collections ont été dotées d'un champ **created_at** et d'un champ **updated_at** (*seuls certains sont représentés ici*), qu'on a uniquement conservé la relation **de caches vers cache_attributes** sous la forme d'un **array de uuid** car on n'utilise pas la réciproque d'un point de vue métier, ou qu'un type **GeoJSON** a été sélectionné pour **caches.loc**.

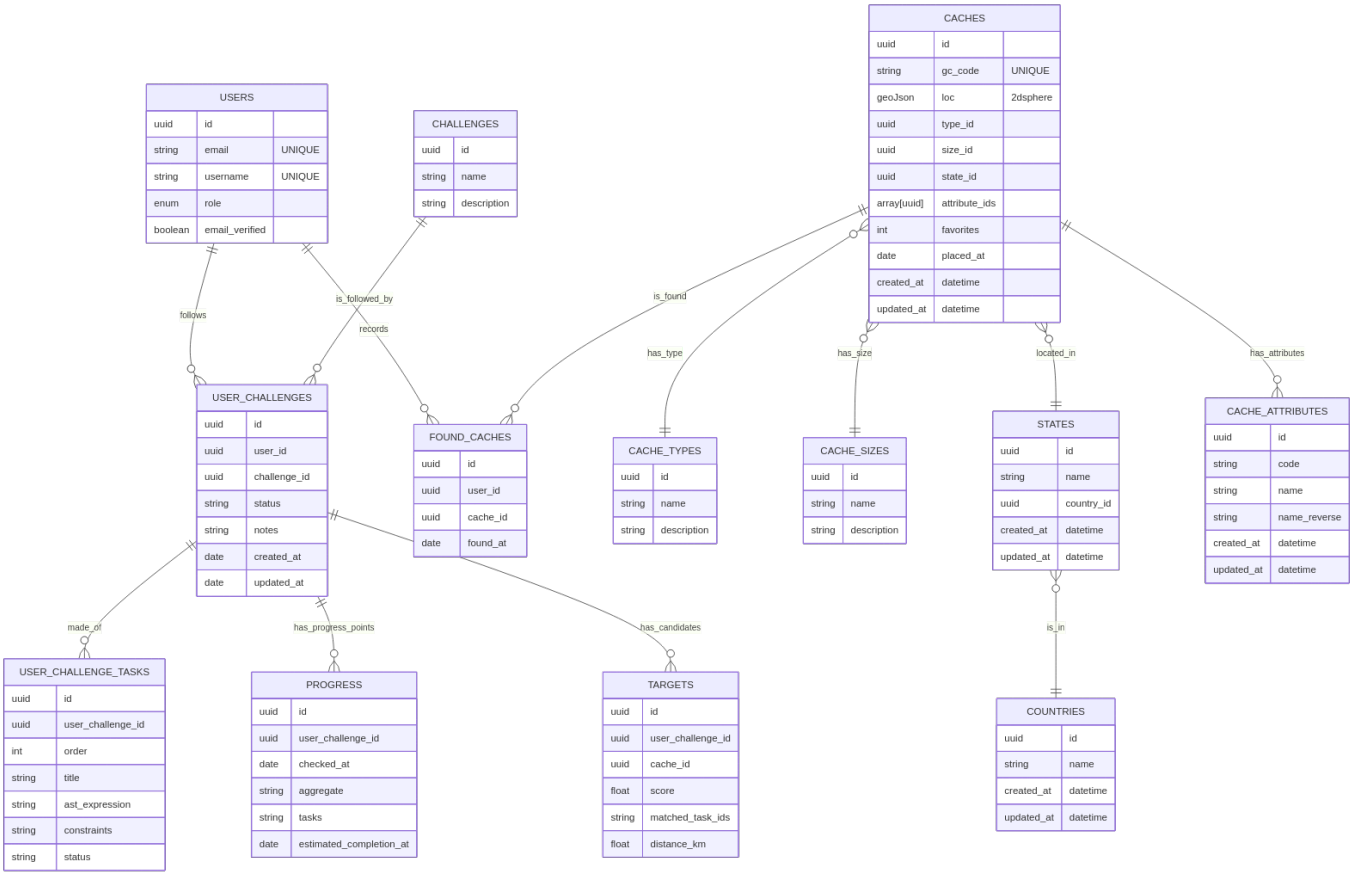


Figure 10: Modèle physique de données (MongoDB)

Note. L'index **2dsphere** sur le champ *caches.loc* est crucial ici : il permet à MongoDB d'exécuter efficacement des requêtes géospatiales (`$geoWithin`, `$nearSphere`) sur les coordonnées des caches, même à grande échelle. Sans cet index, les recherches par rayon (ex. "toutes les caches dans un périmètre de 25 km") seraient un simple scan linéaire de la collection, donc inutilisable dès quelques milliers d'entrées.

6.6 Indexes créés (via seeding)

- **users**
  - `username` (unique, collation insensible à la casse)
  - `email` (unique, collation insensible à la casse)
  - `is_active`, `is_verified`
  - `location` (2dsphere)
- **countries**
  - `name` (unique)
  - `code` (unique, partial string)
- **states**
  - `(country_id, name)` (unique)
  - `(country_id, code)` (unique, partial string)
  - `country_id`
- **cache_attributes**
  - `cache_attribute_id` (unique)
  - `txt` (unique, partial string)
  - `name`
- **cache_sizes**
  - `name` (unique)
  - `code` (unique, partial string)
- **cache_types**
  - `name` (unique)
  - `code` (unique, partial string)
- **caches**
  - `GC` (unique)
  - `type_id`, `size_id`, `country_id`, `state_id`, `(country_id, state_id)`
  - `difficulty`, `terrain`, `placed_at`
  - `title + description_html` (index texte)
  - `loc` (2dsphere)
  - `(attributes.attribute_doc_id, attributes.is_positive)`
  - `(type_id, size_id)`
  - `(difficulty, terrain)`
- **found_caches**
  - `(user_id, cache_id)` (unique)
  - `(user_id, found_date)`
  - `cache_id`
- **challenges**
  - `cache_id` (unique)
  - `name + description` (index texte)
- **user_challenges**
  - `(user_id, challenge_id)` (unique)
  - `(user_id, status, updated_at)`
  - `user_id`, `challenge_id`, `status`
- **user_challenge_tasks**
  - `(user_challenge_id, order)`
  - `(user_challenge_id, status)`
  - `user_challenge_id`
  - `last_evaluated_at`
- **progress**
  - `(user_challenge_id, checked_at)` (unique)
- **targets**
  - `(user_challenge_id, cache_id)` (unique)
  - `(user_challenge_id, satisfies_task_ids)`
  - `(user_challenge_id, primary_task_id)`

- `cache_id`
- `(user_id, score)`
- `(user_id, user_challenge_id, score)`
- `loc` (2dsphere)
- `(updated_at, created_at)`

6.7 Sécurité (vue architecture)

La sécurité a été intégrée dès la conception, tant côté backend que frontend, et couvre l'ensemble de la chaîne d'authentification.

- **Auth** : OAuth2 password flow (login) → **JWT** access + **refresh** (durées distinctes).
- **Rôles** : `user` / `admin` (endpoints d'admin protégés).
- **Comptes** : règles password strictes, **bcrypt + sel variable**, vérification email par lien **temporaire**.
- **Front** : stockage tokens (access en mémoire, refresh en stockage sécurisé si nécessaire), gardes de routes.
- **Back** : contrôle systématique `userId` côté serveur (aucune confiance dans le client), CORS configuré.
- **APIs externes** : clés en variables d'environnement ; timeouts, retries, backoff ; respect des quotas.

6.8 Modèle objet (diagramme de classe UML)

Le diagramme de classes ci-dessous propose une vision **objet/métier** de l'application.
Il ne décrit pas le stockage des données (comme les MLD/MPD), mais la manière dont les **entités interagissent** dans le code et dans la logique métier.

Chaque **classe** représente une entité principale (utilisateur, challenge, cache, tâche, etc.) avec ses **attributs clés**.
Les **liens** indiquent les relations entre ces entités, en précisant leur **multiplicité** (par exemple : un utilisateur peut suivre plusieurs challenges).

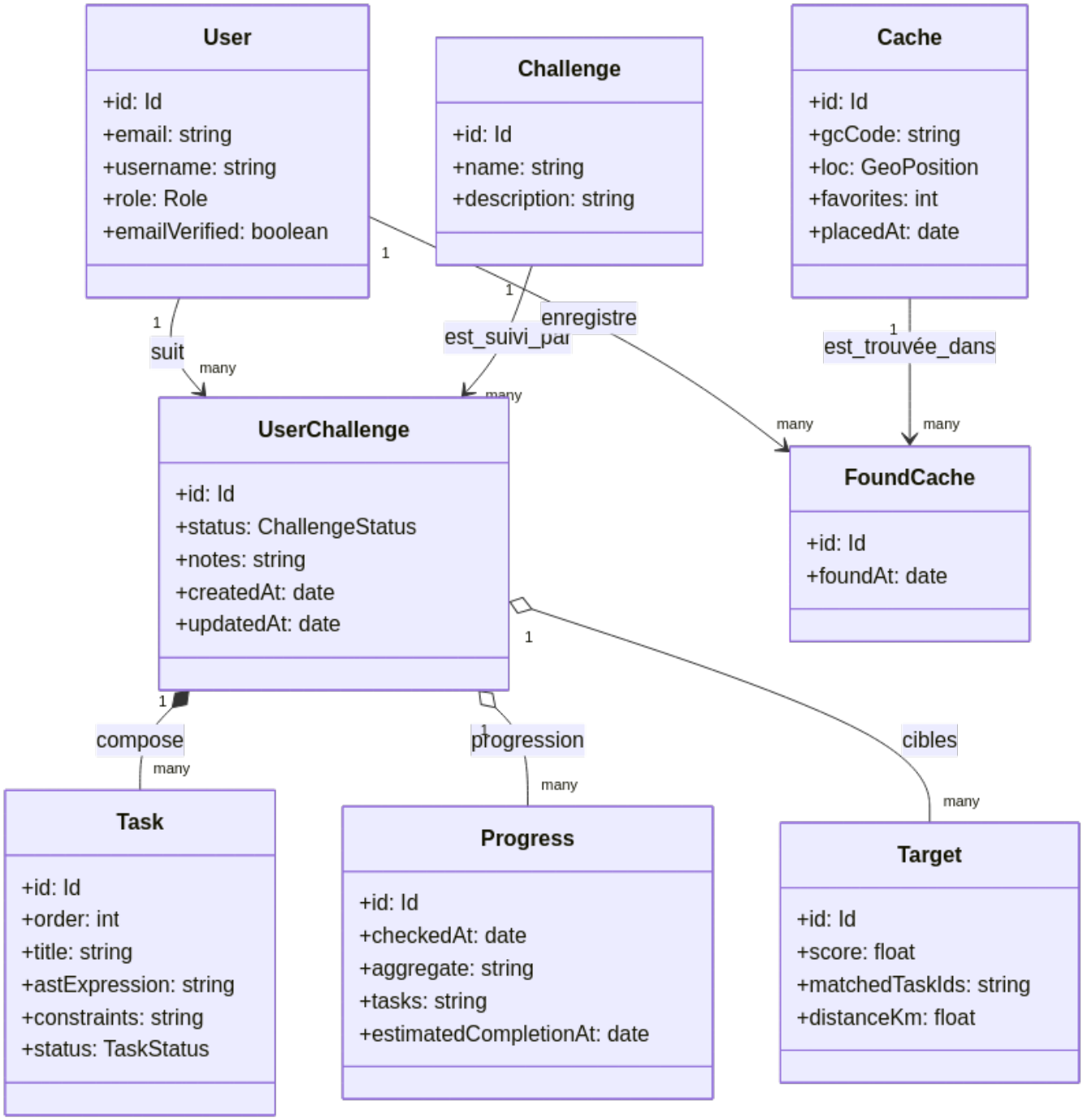


Figure 11: Modèle objet — Noyau métier

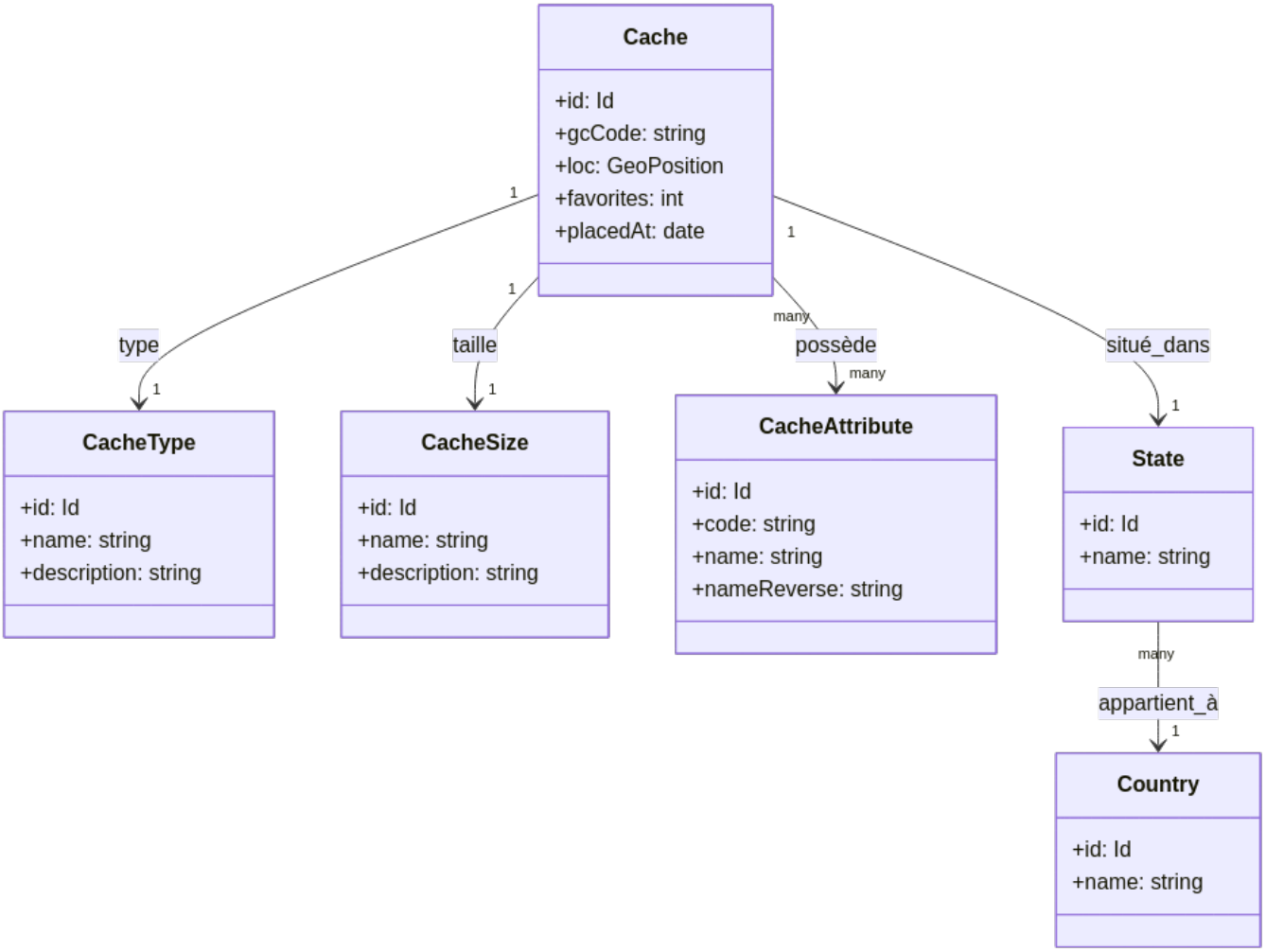


Figure 12: Modèle objet — Référentiels (catalogues)

Concrètement, ce diagramme se lit de la manière suivante :
- Un **User** peut suivre plusieurs **UserChallenges**, chacun étant lié à un **Challenge**.
- Chaque **UserChallenge** est composé de plusieurs **Tasks**, et possède une série de **Progress** ainsi qu'une liste de **Targets** calculés.
- Les **Caches** sont décrits par leurs référentiels (type, taille, attributs, localisation).
- Les **FoundCaches** enregistrent la relation entre un utilisateur et les caches qu'il a trouvées, avec la date correspondante.

6.9 Séquences clés (diagramme de séquence UML)

Les diagrammes suivants illustrent deux scénarios essentiels du fonctionnement de l'application.

6.9.1 Import GPX → sync challenges → premiers snapshots

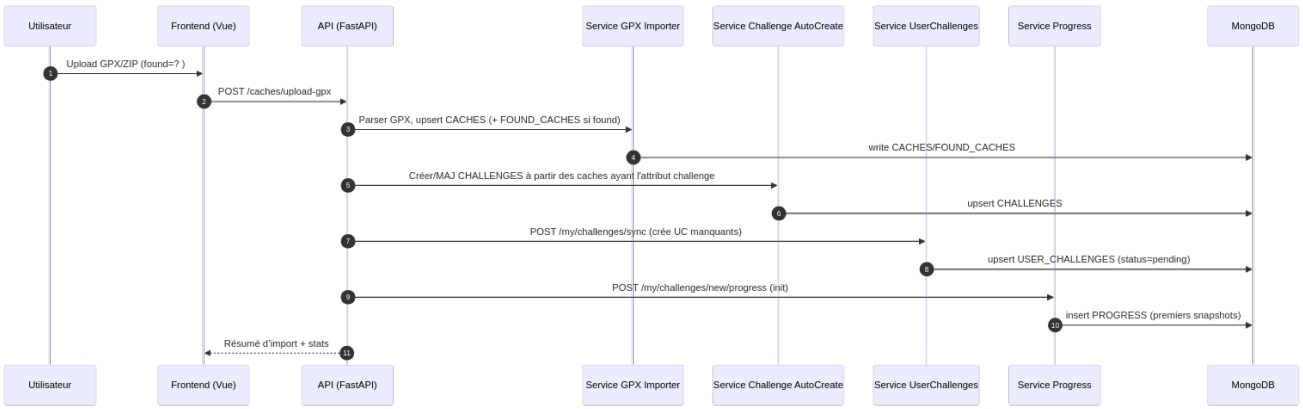


Figure 13: Diagramme de séquence : import GPX et synchronisation

6.9.2 Consultation d'un challenge et projection

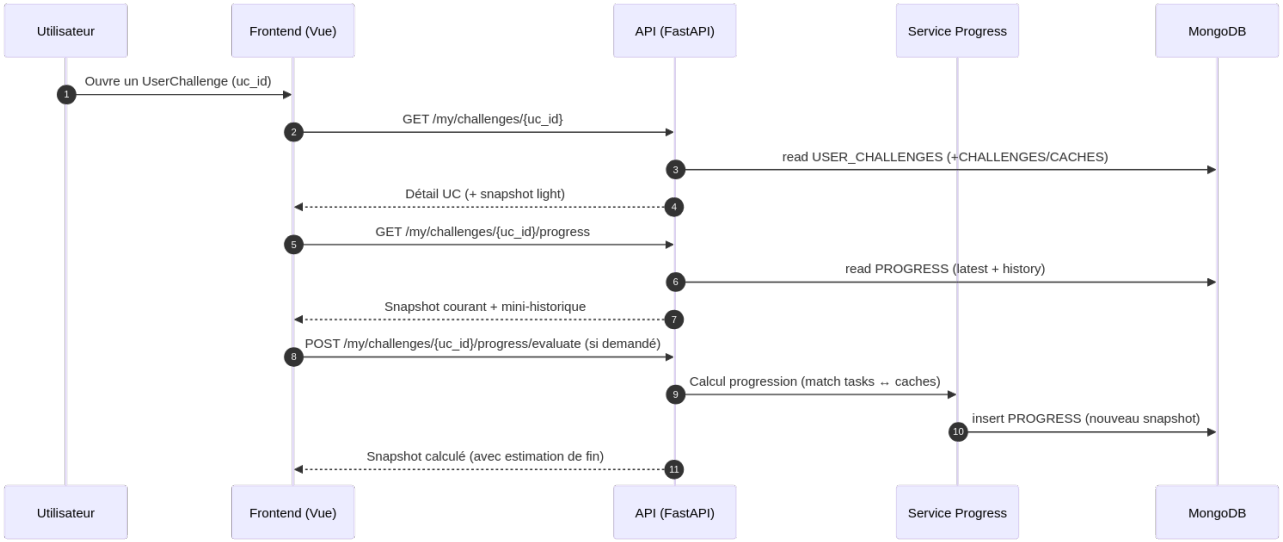


Figure 14: Diagramme de séquence : Consultation d'un challenge et projection

6.10 Performance & cache

Plusieurs mécanismes de performance et de mise en cache ont été intégrés pour assurer la fluidité de l'expérience utilisateur.

- **Clustering** des marqueurs sur carte, **tile caching** OSM côté front.
- **Caps** sur les calculs de targets (`limit_per_task`, `hard_limit_total`).
- **Pagination** standardisée (listes, nearby, bbox, radius ; page/limit ≤ 200).
- **Indices** adaptés (cf. 5.4) pour requêtes géo + filtres combinés.

6.11 Déploiement & configuration

L'architecture logicielle est complétée par une configuration claire des environnements et du pipeline de déploiement.

- **Docker** : images séparées frontend/backend ; compose pour dev.
- **Variables d'environnement** : DB (URI/credentials), secrets JWT, SMTP, fournisseurs OpenData.
- **CI/CD** : GitHub Actions — lint/tests → build → déploiement (Railway/équivalent).
- **Fichiers** : uploads GPX (répertoire dédié, nettoyage planifié recommandé).

6.12 Évolutions d'architecture envisagées

Enfin, certaines évolutions sont déjà identifiées pour enrichir l'architecture et anticiper les besoins futurs.

- **Mode offline** : IndexedDB + stratégie de sync.
- **Multi-challenges map** : vue combinée (optimisation multi-objectifs).
- **Cache applicatif** côté API (targets/progress récents).
- **Observabilité** : métriques Prometheus, traces OpenTelemetry.

7 Choix techniques

7.1 Backend

Le choix du backend s'est porté sur FastAPI, pour des raisons de rapidité, de sécurité et de simplicité d'intégration.

- **FastAPI (Python)** : choisi pour sa rapidité de développement, sa clarté syntaxique et son support natif d'OpenAPI.
- **Points forts** : asynchrone, validation avec Pydantic, gestion simple des dépendances.
- **Sécurité** : intégration native d'OAuth2, JWT, dépendances paramétrées pour gérer rôles et droits.

```
# Validation de la complexité du mot de passe
def validate_password_strength(password: str) -> bool:
    """Valide la complexité du mot de passe (MIN_PASSWORD_LENGTH+ chars, A/a/0/special). Lève HTTP
    ↪  400 sinon."""
    if len(password) < MIN_PASSWORD_LENGTH \
        or not re.search(r"[A-Z]", password) \
        or not re.search(r"[a-z]", password) \
        or not re.search(r"[0-9]", password) \
        or not re.search(r"[\W_]", password):
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=f"Password must be at least {MIN_PASSWORD_LENGTH} characters and include
    ↪ uppercase, lowercase, number, and special character."
        )
    return True
```

```
# Vérification du mot de passe (Passlib/bcrypt)
def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Compare le mot de passe en clair au hash stocké."""
    return pwd_context.verify(plain_password, hashed_password)
```

```
# Création 'dun access token JWT
def create_access_token(data: dict, expires_delta: dt.timedelta | None = None):
    """Encode un JWT signé avec 'exp' (par défaut +15 min si non précisé)."""
    to_encode = data.copy()
    expire = now() + (expires_delta or dt.timedelta(minutes=15))
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, settings.jwt_secret_key, algorithm=settings.jwt_algorithm)
    return encoded_jwt
```

- **Exemple d'usage** : définition rapide des endpoints REST, testables automatiquement via `/docs`.

7.2 Frontend

Le frontend repose sur Vue.js 3, accompagné de bibliothèques spécialisées pour le style et la cartographie.

- **Vue.js 3** : framework progressif, adapté au rendu dynamique de données complexes.
- **Librairies** : Tailwind CSS pour le style rapide, Leaflet pour la cartographie interactive.
- **Optimisation** : clustering des marqueurs, tile caching pour réduire la charge réseau et améliorer la fluidité sur mobile.
- **Approche "green computing via performance optimizing"** : chaque interaction doit minimiser le rendu inutile, afin de préserver batterie et ressources CPU.

7.3 Base de données

Pour la persistance des données, le choix de MongoDB s'explique par la variabilité inhérente aux caches géocaching.

- **MongoDB (NoSQL)** : choisi pour sa souplesse documentaire. Les caches étant très hétérogènes (attributs variables, champs facultatifs), le modèle documentaire colle mieux que le relationnel.
- **Index** : `2dsphere` pour les recherches géospatiales, indexes uniques pour GC et utilisateurs, indexes combinés pour les filtres métiers (D/T, attributs).
- **Avantage** : structure flexible, évolutive selon l'apparition de nouveaux types de caches ou de challenges.

7.4 Intégrations externes

L'application s'appuie également sur des services externes, principalement des APIs OpenData.

- **APIs OpenData** : utilisées pour l'altimétrie et la localisation (communes).
- **Choix stratégique** : indépendance contractuelle (pas de dépendance propriétaire), souplesse (changement ou combinaison d'APIs si besoin), résilience (répartition de charge possible).
- **Mitigation des risques** : respect strict des quotas d'appel, stockage en base pour limiter la redondance des requêtes.

7.5 DevOps et qualité

Enfin, la démarche DevOps et les objectifs qualité viennent compléter l'architecture en assurant la robustesse et la maintenabilité du projet.

- **Docker / docker-compose** : reproductibilité entre environnements dev/prod.
- **CI/CD GitHub Actions** : linting, tests Pytest, build images, déploiement automatisé.
- **Tests** : Pytest (backend), Cypress (frontend).
- **Sécurité** : secrets gérés via variables d'environnement, pas en dur dans le code.
- **Observabilité** : logs structurés, métriques de performance prévues.

En résumé : Les choix techniques privilégient la **sécurité**, la **performance** (y compris mobile et éco-conception), et la **souplesse** pour s'adapter à la variabilité du geocaching et à l'évolution des APIs externes.

8 DevOps — Conteneurisation, Déploiement et Git

8.1 Objectifs

- **Cohérence d'exécution** : garantir le même comportement entre dev et prod, bien que les **configurations diffèrent** (montages & hot reload en dev, images figées et Nginx en prod).
- **Intégration continue** : tests automatiques et builds sur GitHub Actions.
- **Déploiement continu** : mise en production automatisée sur un VPS.
- **Sécurité** : gestion des secrets hors code (JWT, MongoDB, SMTP).
- **Suivi du code** : workflow Git structuré par contexte (backend / frontend).

8.2 Conteneurisation (Docker)

8.2.1 Images

- **Backend** : Python + FastAPI, image slim avec Uvicorn.
- **Frontend** : Vue.js (Vite) → artefacts statiques servis par Nginx.
- **Base de données** : MongoDB Atlas (pas de conteneur en prod).

8.2.2 docker-compose (développement)

- Services : backend, frontend, éventuellement reverse proxy.
- Variables d'environnement chargées depuis `.env.development`.
- Hot reload activé pour le confort développeur.

Exemple

```
docker compose up -d --build
docker compose exec backend pytest -q
```

> ***Note dev vs prod*** *— En développement,* `docker-compose` *utilise des* ***volumes montés*** *et le* ***hot reload*** *(ports exposés).*
> *En production, les services sont déployés à partir d'****images versionnées*** *(sans hot reload), le frontend est servi par* ***Nginx*** *et la configuration provient d'un* ***fichier d'environnement dédié****.*

8.3 Git — Convention de branches et workflow

8.3.1 Nommage des branches

Le projet étant séparé en **API** et **interface**, les branches sont préfixées par leur contexte :

- `backend/<section>` : auth, caches, challenges, progress, targets, tests…
- `frontend/<section>` : auth, appshell, ui-foundation, home…
- Maintenance : ex. `backend/fix/sendmail`.
- Branche utilitaire : `install` (initialisation de la structure du projet).

Ce nommage évite les collisions front/back (qui apparaissent avec un simple `feature/*`) et reflète la logique de développement.

8.3.2 Politique de fusion

- Développement sur `backend/<section>` ou `frontend/<section>`.
- Une fois validé, merge dans `backend/main` ou `frontend/main`.
- **Main global** : seule source de vérité pour la production, alimenté uniquement par `backend/main` ou `frontend/main`.

Ce workflow garantit un historique lisible et réduit les risques de régressions.

8.4 CI/CD et déploiement

8.4.1 GitHub Action (build & deploy réels)

Le workflow suivant illustre le **build** des images (backend & frontend) avec **tags `latest` et `sha-<commit>`**, puis un **déploiement SSH** vers un **utilisateur dédié** sur le VPS.
Il met en évidence : - la **normalisation du nom** `owner/repo` en minuscules (pour GHCR), - l'usage d'une **clé SSH dédiée au déploiement**, - des `echo` de **journalisation** de chaque étape côté VPS, - et la **lecture centralisée** des variables d'environnement depuis `../shared/env/app.env`.

```
name: build-and-push

on:
  push:
    branches: ["main"]
  workflow_dispatch: {}

permissions:
  contents: read
  packages: write

jobs:
  build:
    runs-on: ubuntu-latest
    outputs:
      owner_lc: ${{ steps.names.outputs.owner_lc }}
      repo_lc: ${{ steps.names.outputs.repo_lc }}
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Compute lowercase owner & repo
        id: names
        run: |
          REPO_LC="${GITHUB_REPOSITORY##*/}"
          OWNER_LC="${GITHUB_REPOSITORY_OWNER}"
          echo "REPO_LC=${REPO_LC,,}" >> $GITHUB_ENV
          echo "OWNER_LC=${OWNER_LC,,}" >> $GITHUB_ENV
          echo "owner_lc=${OWNER_LC,,}" >> $GITHUB_OUTPUT
          echo "repo_lc=${REPO_LC,,}" >> $GITHUB_OUTPUT

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Login to GHCR (GITHUB_TOKEN)
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Build & Push backend
        uses: docker/build-push-action@v6
        with:
          context: ./backend
          file: ./backend/Dockerfile
          push: true
          tags: |
            ghcr.io/${{ env.OWNER_LC }}/${{ env.REPO_LC }}/backend:latest
            ghcr.io/${{ env.OWNER_LC }}/${{ env.REPO_LC }}/backend:sha-${{ github.sha }}

      - name: Build & Push frontend
        uses: docker/build-push-action@v6
        with:
          context: ./frontend
          file: ./frontend/Dockerfile
          push: true
          tags: |
            ghcr.io/${{ env.OWNER_LC }}/${{ env.REPO_LC }}/frontend:latest
            ghcr.io/${{ env.OWNER_LC }}/${{ env.REPO_LC }}/frontend:sha-${{ github.sha }}
```

```
  deploy:
    runs-on: ubuntu-latest
    needs: build
    env:
      OWNER_LC: ${{ needs.build.outputs.owner_lc }}
      REPO_LC: ${{ needs.build.outputs.repo_lc }}
      SHA_TAG: sha-${{ github.sha }}

    steps:
      - name: Deploy over SSH (compose pull + up) — Variante A (latest)
        uses: appleboy/ssh-action@v1.2.0
        with:
          host: ${{ secrets.DEPLOY_SSH_HOST }}
          username: ${{ secrets.DEPLOY_SSH_USER }}
          key: ${{ secrets.DEPLOY_SSH_PRIVATE_KEY }}
          script_stop: true
          script: |
            set -Eeuo pipefail

            echo "== whoami ==" && whoami
            echo "== date ==" && date -Is
            echo "== go to compose dir =="

            cd /home/deploy/apps/gctracker/compose
            echo "PWD: $(pwd)"
            echo "== list files =="
            ls -la

            echo "== show env entries from ../shared/env/app.env =="
            grep -E '^(IMAGE_BACKEND|IMAGE_FRONTEND|BACKEND_PORT|FRONTEND_PORT)=' ../shared/env/
    ↪ app.env || true

            echo "== docker compose version =="
            docker compose version

            echo "== preview resolved compose (with --env-file) =="
            docker compose --env-file ../shared/env/app.env config

            echo "== pull images =="
            docker compose --env-file ../shared/env/app.env pull

            echo "== up -d (remove orphans) =="
            docker compose --env-file ../shared/env/app.env up -d --remove-orphans

            echo "== ps =="
            docker compose --env-file ../shared/env/app.env ps
```

8.4.2 Déploiement distant sans script local

Le déploiement est **déclenché depuis GitHub Actions** via une connexion **SSH** à un **utilisateur dédié** (`deploy`) sur le VPS.
Les commandes `docker compose` sont exécutées **à distance**, en **chargeant un fichier d'environnement centralisé** (`../shared/env/app.env`).
Les `echo` présents dans le workflow permettent de **tracer finement** chaque étape (qui, quand, quoi, où).

> ***Variante B (optionnelle)*** *: un script local `deploy.sh` peut encapsuler ces commandes si l'on souhaite déclencher un déploiement manuel côté VPS.*

Ce script tire les nouvelles images, relance les services et nettoie les images obsolètes.
Les secrets applicatifs restent stockés localement sur le VPS (`.env` sécurisé).

8.4.3 Arborescence VPS & gestion des variables

L'arborescence suivante centralise la configuration et sépare **compose** (déploiement) des **variables d'environnement** (conf), lesquels ont été réparties entre un fichier de configuration et un fichier de secrets :

/home/deploy/apps/gctracker/ compose/ # docker-compose.yml (prod) docker-compose.yml shared/ env/ app.env # variables d'environnement prod secrets.env # secrets d'environnement prod

`app.env` (extraits) :

```
IMAGE_BACKEND=ghcr.io/<owner>/<repo>/backend:latest
IMAGE_FRONTEND=ghcr.io/<owner>/<repo>/frontend:latest

BACKEND_PORT=8000
FRONTEND_PORT=80
```

`secrets.env` (extraits) :

```
MONGODB_URI=...
JWT_SECRET_KEY=...
JWT_ALGORITHM=HS256
SMTP_HOST=...
SMTP_USER=...
SMTP_PASSWORD=...
```

> *Avantages : séparation nette **code/config**, **rotation** des secrets facilitée, et **reproductibilité** des déploiements.*

> ***Sécurité du déploiement** - Utilisateur **dédié** (`deploy`) avec droits strictement nécessaires. - **Clé SSH** dédiée stockée dans **GitHub Secrets** (`DEPLOY_SSH_PRIVATE_KEY`). - **Aucune** variable sensible transitant par l'Action : tout est chargé via `../shared/env/app.env` côté VPS. - **Images versionnées** (`latest` + `sha-<commit>`) pour permettre un **rollback** rapide.*

9 Gestion de projet

9.1 Introduction

Le projet a été mené seul dans le cadre de la formation CDA.
Les outils utilisés : GitHub Project (roadmap/kanban), Git, commits, et l'appui d'un pool de testeurs.

9.2 Planning prévisionnel

Organisation en lots thématiques :

Lot	Intitulé	Période prévue
1	Conception documents	début juillet
2	Environnement technique	mi-juillet
3	Conception graphisme	août-septembre
4	Backend (modèles & sécu)	juillet-août
5	Déploiement & packaging	août-septembre
6	Ciblage & progression	fin août-sept.
7	Clôture & livrables	septembre

9.2.1 Outils de planification

- **Vue Roadmap** : représentation temporelle, utile pour plan global mais abstraite.
- **Vue Kanban** : colonnes « À faire », « En cours », « Fait », plus intuitive et motivante.

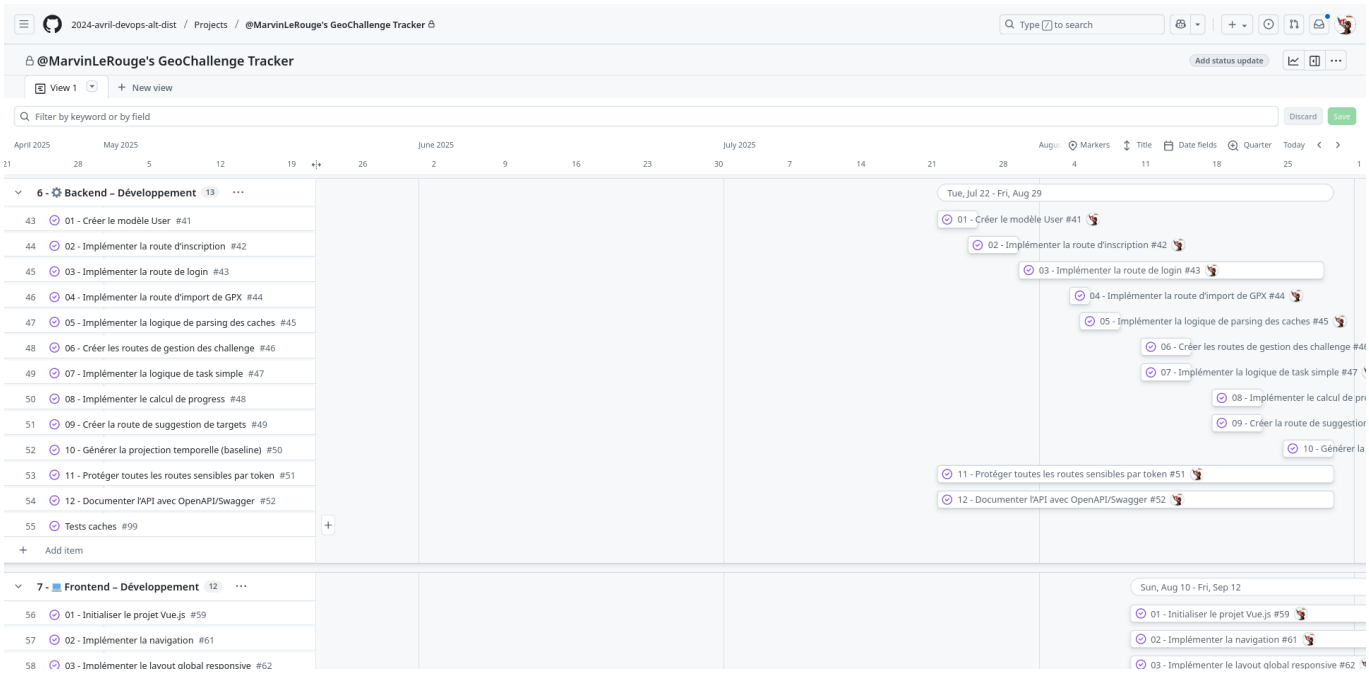


Figure 15: Vue Roadmap

La vue **Roadmap** de GitHub Project permet de représenter les tâches dans une chronologie, avec des jalons et une répartition temporelle. Elle offre une vision d'ensemble intéressante pour situer chaque lot dans le temps, et pour visualiser la progression globale du projet. Toutefois, cette vue reste assez abstraite dans la pratique : les tâches sont alignées dans une frise, mais sans retour direct sur leur état réel d'avancement ni sur le travail concret restant à effectuer.

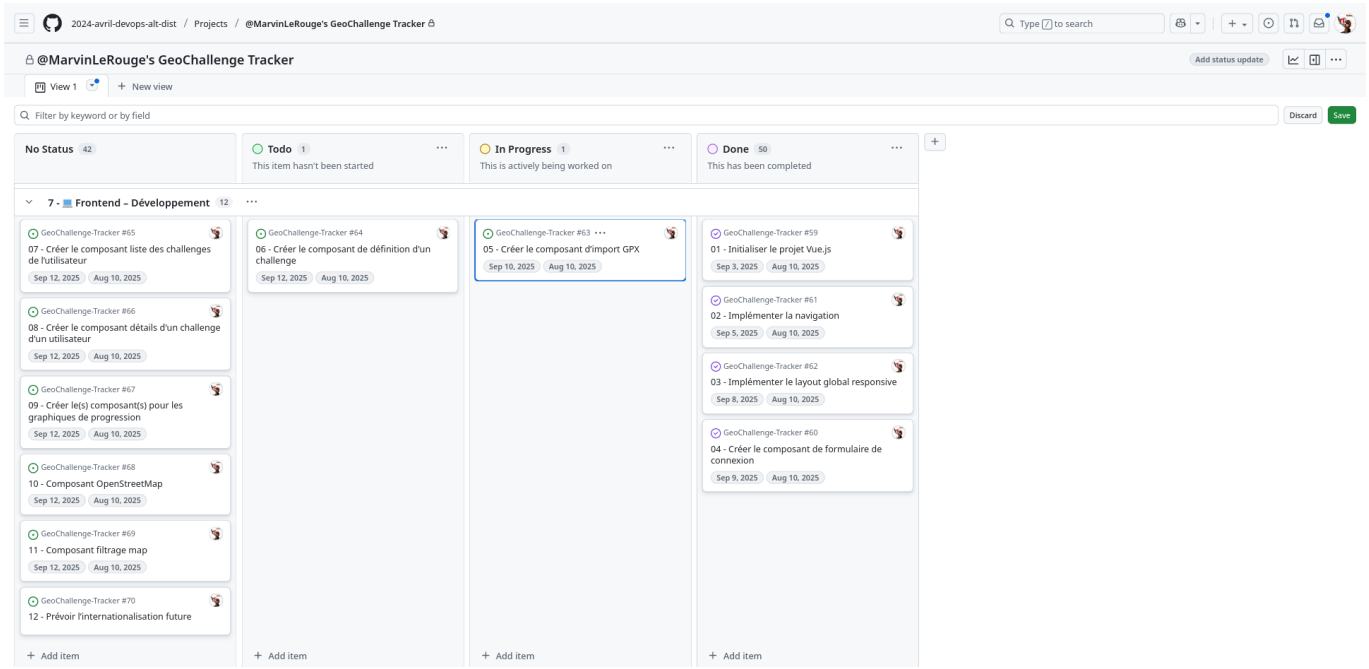


Figure 16: Vue Kanban

La vue **Kanban**, au contraire, repose sur un principe simple : faire passer les cartes d'une colonne à une autre, de "À faire" à "En cours" puis à "Fait". Ce mode de gestion apporte une dynamique plus opérationnelle : il met l'accent sur les actions immédiates et sur la progression visible. Chaque déplacement de carte constitue un retour tangible et motivant, ce qui facilite le suivi au quotidien. Cette approche permet aussi de limiter la dispersion en mettant en avant les tâches encore bloquées dans "À faire" ou "En cours".

Analyse personnelle : Dans le cadre de ce projet, mené avec un **timing serré**, la vue Kanban s'est imposée comme la plus efficace. Elle m'a permis de rester centré sur les actions concrètes, en visualisant en permanence ce qui était achevé et ce qui restait à entreprendre. Le changement de statut entre les colonnes m'a donné un cadre clair et motivant, plus adapté à un projet solo que la vue Roadmap. Cette dernière reste utile pour illustrer la planification initiale, mais le Kanban a constitué l'outil principal de suivi opérationnel.

Je suis également conscient que dans un contexte différent, avec une équipe plus large ou un rôle davantage orienté vers le **pilotage global**, la vue Roadmap aurait été plus pertinente. Elle permet en effet de suivre la cohérence d'ensemble, de contrôler les dépendances entre tâches et de vérifier que les livrables sont alignés avec les échéances prévues. Dans ce projet, mené en solitaire, cette dimension avait moins de sens immédiat. Néanmoins, dans un cadre collaboratif, l'usage de la Roadmap s'imposerait comme un outil de coordination complémentaire au Kanban.

En conclusion, la gestion de projet a reposé sur des outils simples mais adaptés, permettant de concilier planification initiale et suivi opérationnel. L'approche Kanban a favorisé l'efficacité dans un contexte individuel et contraint, tout en gardant à l'esprit que d'autres vues, comme la Roadmap, seraient à privilégier pour une coordination d'équipe à plus grande échelle.

9.3 Suivi réel (commits et branches)

La réalité a montré un fort pic d'activité en août.
Les branches Git ont servi d'outil de suivi par fonctionnalité :

Branche	Fonction principale	Rôle
backend/auth	Authentification, JWT	sécurité API
backend/caches	Import GPX, BDD	ingestion
backend/challenges	AST des tâches	logique métier
backend/progress	Projections	stats, courbes
backend/targets	Suggestions	objectifs
backend/tests	Pytest	qualité
frontend/auth	UI login/register	ergonomie
frontend/appshell	Layout général	navigation
frontend/ui-found.	Design system	styles
frontend/home	Page d'accueil	point d'entrée

Politique de merge :
- branches de travail → `backend/main` ou `frontend/main`,
- puis uniquement depuis ces branches d'intégration → `main`.

> ***Lien avec le DevOps*** *— Chaque intégration dans* `main` *déclenche automatiquement le* ***pipeline CI/CD*** *décrit au chapitre 8, garantissant un cycle* ***code → build → déploiement*** *sans rupture.*

9.4 Analyse des écarts

- Documentation et env technique réalisés en temps voulu.
- Backend concentré sur août (80+ commits).
- Frontend amorcé plus tardivement.
- Finalisation (tests, doc, packaging) sur septembre.

9.5 Environnement humain

Un pool de testeurs variés (ex. **Arnokovic, n°4 français**, poseurs prolifiques, profils familiaux).
Ils garantissent des retours pertinents et représentatifs.

9.6 Objectifs qualité

- **Logicielle** : Pytest, Cypress, CI/CD.
- **Usage** : mobile-first, perf optimisée.
- **Sécurité** : mots de passe forts, JWT, RGPD.
- **Maintenabilité** : archi claire, indexes Mongo, seeding idempotent.

10 Script de création / modification de la base de données

Ce document décrit **comment initialiser** la base MongoDB (référentiels, compte admin) et **assurer la création des index** au démarrage, à partir des scripts fournis dans le backend.

10.1 Objectifs

L'initialisation de la base de données poursuit plusieurs objectifs essentiels, garantissant à la fois la robustesse et la sécurité de l'application.

- **Tester la connexion** à MongoDB et arrêter proprement en cas d'échec.
- **Créer/mettre à jour les index** (unicité, géo, texte, partiels, collation).
- **Seeder les référentiels** (types, tailles, attributs) et **l'utilisateur admin**.

10.2 Pré-requis & variables d'environnement

Avant d'exécuter les scripts, certains prérequis et variables doivent être définis pour assurer une configuration cohérente.

- **MongoDB Atlas** (ou compatible) accessible via URI.
- Variables d'environnement attendues :
  - `MONGODB_URI` (ou équivalent utilisé par l'app)
  - `ADMIN_USERNAME`, `ADMIN_EMAIL`, `ADMIN_PASSWORD`
  - (éventuellement) `SMTP_*` si la vérification email est testée

> ***Sécurité** : ne versionnez jamais vos valeurs réelles. Utilisez un `.env` local ou des* repository secrets *côté CI/CD.*

10.3 Données seedées

Les données de référence sont stockées dans des fichiers JSON et injectées lors du seeding initial.

Répertoires (backend) : `data/seeds/*.json`

- `cache_types.json`
- `cache_sizes.json`
- `cache_attributes.json`

> *Les jeux de données sont insérés **si la collection est vide** (ou **forcés** avec l'option `--force`).*

10.4 Index créés (via seeding)

Le script assure également la création des index nécessaires, de manière idempotente, afin de garantir les performances attendues.

- **users** : unicité insensible à la casse sur `username`, `email` (collation) ; index géo `location (2dsphere)` ; flags `is_active`, `is_verified`.
- **countries** : `name` (unique), `code` (unique partiel si string).
- **states** : `country_id` ; `(country_id, name)` unique ; `(country_id, code)` unique partiel.
- **cache_attributes** : `cache_attribute_id` (unique), `txt` (unique partiel), `name`.
- **cache_sizes** : `name` (unique), `code` (unique partiel).
- **cache_types** : `name` (unique), `code` (unique partiel).
- **caches** : `GC` (unique), `loc (2dsphere)`, `type_id`, `size_id`, `country_id`, `state_id`, `(country_id, state_id)`, `difficulty`, `terrain`, `placed_at (desc)`, index **texte** `(title, description_html)`, et combinaisons métier : `(attributes.attribute_doc_id, attributes.is_positive)`, `(type_id, size_id)`, `(difficulty, terrain)`.
- **found_caches** : `(user_id, cache_id)` (unique), `(user_id, found_date)`, `cache_id`.
- **challenges** : `cache_id` (unique), index **texte** `(name, description)`.

- **user_challenges** : `(user_id, challenge_id)` (unique), `user_id`, `challenge_id`, `status`, `(user_id, status, updated_at)`.
- **user_challenge_tasks** : `(user_challenge_id, order)`, `(user_challenge_id, status)`, `user_challenge_id`, `last_evaluated_at`.
- **progress** : `(user_challenge_id, checked_at)` (unique).
- **targets** : `(user_challenge_id, cache_id)` (unique), `(user_challenge_id, satisfies_task_ids)`, `(user_challenge_id, primary_task_id)`, `cache_id`, `(user_id, score)`, `(user_id, user_challenge_id, score)`, `loc (2dsphere)`, `(updated_at, created_at)`.

L'index **2dsphere** est l'un des éléments centraux du modèle physique de la base. Chaque cache est représentée par un champ `loc` de type GeoJSON (`{ type: "Point", coordinates: [lon, lat] }`). Cet index permet à MongoDB d'exécuter de manière optimale toutes les requêtes géospatiales utilisées par l'application :

- sélection des caches dans un rayon donné (`$geoWithin` + `$centerSphere`),
- recherche des caches les plus proches d'un point (`$nearSphere`),
- filtrage combiné (géolocalisation + critères métiers D/T, attributs, type).

Grâce à l'index 2dsphere, ces requêtes s'exécutent en **temps logarithmique**, indépendamment de la taille totale de la collection, et restent performantes même après l'import massif de dizaines ou centaines de milliers de caches depuis des GPX. Il s'agit donc d'un **pré-requis technique essentiel** pour la fluidité de l'outil, qui doit être capable d'afficher en temps réel les caches pertinentes sur carte et de répondre aux tâches complexes des challenges sans ralentir l'expérience utilisateur.

10.5 Exécution (local, sans Docker)

Les scripts peuvent être exécutés aussi bien en local qu'au travers de Docker Compose, selon le contexte de développement.

Depuis la racine du backend :

```
# Activez votre venv au besoin puis :
python -m app.db.seed_data             # ping + ensure_indexes + seed référentiels + admin
python -m app.db.seed_data --force     # idem, mais vide les collections de référentiels avant
    ↪ réinsertion
```

> ***Attention*** *: `--force` réinitialise les collections de référentiels (pas les données utilisateur).*

10.6 Exécution (via Docker Compose)

Exemples indicatifs (adapter le service et le runner Python selon votre compose) :

```
# 1) Bâtir et lancer les conteneurs
docker compose up -d --build

# 2) Exécuter le seeding dans le conteneur backend
# (remplacez <backend> par le nom réel du service backend dans le docker-compose.yml)
docker compose exec <backend> python -m app.db.seed_data --force
```

> *Les secrets (URI Mongo, admin) doivent être injectés au conteneur via l'environnement (compose, variables Railway, etc.).*

10.7 Composants clés (extraits commentés)

Les extraits suivants illustrent les mécanismes mis en place pour la gestion des index et le seeding des données.

10.7.1 Assurer les index (idempotent)

```
# app/db/seed_indexes.py — extrait simplifié
from pymongo import ASCENDING, DESCENDING, TEXT
```

```
from pymongo.operations import IndexModel
from pymongo.collation import Collation
from app.db.mongodb import get_collection

COLLATION_CI = Collation(locale="en", strength=2)  # insensible à la casse

def ensure_index(coll_name, keys, *, name=None, unique=None, partial=None, collation=None):
    coll = get_collection(coll_name)
    # ... comparaison index existant / options ...
    opts = {}
    if name: opts['name'] = name
    if unique is not None: opts['unique'] = unique
    if partial: opts['partialFilterExpression'] = partial
    if collation is not None: opts['collation'] = collation
    coll.create_indexes([IndexModel(keys, **opts)])

# Exemple : unicité insensible à la casse
ensure_index('users', [('username', ASCENDING)], name='uniq_username_ci', unique=True, collation=
    ↪ COLLATION_CI)
```

10.7.2 Seeding des référentiels & admin

```
# app/db/seed_data.py — extrait simplifié
from app.db.mongodb import db as mg_db, get_collection
from app.db.seed_indexes import ensure_indexes
from app.core import security

def seed_collection(file_path, collection_name, force=False):
    count = mg_db[collection_name].count_documents({})
    if count > 0 and not force:
        return
    if force:
        mg_db[collection_name].delete_many({})
    mg_db[collection_name].insert_many(json.load(open(file_path)))

def seed_admin_user():
    coll = get_collection("users")
    pwd_hash = security.pwd_context.hash(os.getenv("ADMIN_PASSWORD"))
    coll.update_one({"username": os.getenv("ADMIN_USERNAME")},
                    {"$set": {..., "password_hash": pwd_hash, "role": "admin"},
                     "$setOnInsert": {"created_at": now()}}, upsert=True)

if __name__ == "__main__":
    test_connection()   # ping Mongo
    ensure_indexes()    # création/MAJ idempotente des index
    seed_referentials(force="--force" in sys.argv)
```

10.8 Bonnes pratiques & sécurité

Quelques bonnes pratiques doivent être respectées pour maintenir la cohérence des données et la sécurité de l'application.

- **Idempotence** : relancer le script ne casse pas les index existants si la configuration n'a pas changé.
- **Collation** : utilisez des collations cohérentes pour les unicités *case-insensitive* (ex. utilisateurs).
- **Indexes géo & texte** : un seul index **texte** par collection ; vérifiez la présence des `2dsphere` pour les requêtes cartographiques.
- **Quotas & coûts** : la multiplication d'index a un **coût d'écriture** ; validez les index réellement utiles via vos workloads.
- **Secrets** : stockez l'URI Mongo et le mot de passe admin via variables d'environnement (jamais en clair).

10.9 Vérification après exécution

Une fois le seeding effectué, il est important de vérifier la présence effective des index attendus. Checklist rapide (via mongosh) :

```
use <your_db>
// Exemples
db.users.getIndexes()
db.caches.getIndexes()
db.challenges.getIndexes()
```

Vous devez retrouver les index listés en **6.6**.

10.10 Tests associés au seeding

Des tests unitaires viennent compléter le dispositif afin de s'assurer de la bonne accessibilité de la base et du respect des contraintes.

Un test Pytest permet de vérifier que l'environnement backend accède bien à MongoDB :

```
# backend/tests/test_connectivity.py
from app.db.mongodb import client as mg_client

def test_backend_can_access_mongo():
    dbs = mg_client.list_database_names()
    assert isinstance(dbs, list)
```

- Vérifie que la connexion fonctionne et qu'une liste de bases est renvoyée.
- Peut être lancé seul pour diagnostiquer un problème de connexion.
- Intégré dans la suite Pytest pour automatiser le contrôle lors du CI/CD.

11 Veille — sécurité & technologies

Pendant le développement de *GeoChallenge Tracker*, une veille continue a été menée sur la sécurité et les technologies utilisées.
Cette démarche s'est concentrée à la fois sur les vulnérabilités spécifiques à **FastAPI** et **MongoDB**, et sur des menaces plus larges liées aux **dépendances logicielles** (chaîne d'approvisionnement, packages compromis).
L'objectif est double : anticiper les risques et garantir la robustesse du projet dans la durée.

11.1 Cas concrets récents

11.1.1 Compromission de paquets npm (attaque de la chaîne d'approvisionnement)

En septembre 2025, plusieurs paquets très utilisés sur **npm** (*debug*, *chalk*, *ansi-styles*) ont été compromis à la suite d'un compte mainteneur piraté via une attaque de phishing.
Les versions infectées injectaient du code malveillant visant notamment le vol de fonds dans des portefeuilles crypto.
Cet incident, considéré comme l'une des plus grosses attaques sur npm, illustre la **fragilité de la chaîne d'approvisionnement** : même des dépendances réputées sûres peuvent devenir un vecteur d'attaque.
Cela rappelle l'importance d'un suivi attentif des versions et de ne dépendre que de packages essentiels web†source .

11.1.2 Vulnérabilité FastAPI Guard (CVE-2025-46814)

Une vulnérabilité a été identifiée dans la librairie *FastAPI Guard*, un middleware utilisé pour ajouter des contrôles de sécurité (restriction d'IP, logging, etc.) dans FastAPI.
Elle permettait d'injecter une valeur dans l'en-tête `X-Forwarded-For` et de contourner certains contrôles d'accès basés sur l'adresse IP.
Les journaux pouvaient être falsifiés et des clients non autorisés pouvaient passer pour des utilisateurs légitimes.
Le problème a été corrigé rapidement, mais cet événement démontre qu'un projet ne doit pas se contenter des protections par défaut : il faut surveiller activement les **dépendances tierces** et limiter leur usage au strict nécessaire web†source .

11.1.3 Incident de sécurité MongoDB (décembre 2023)

En décembre 2023, MongoDB a révélé un incident de sécurité impliquant un accès non autorisé à ses systèmes internes après une campagne de phishing.
Des métadonnées clients (noms, adresses e-mail, numéros de téléphone) ont été exposées, même si les données hébergées sur MongoDB Atlas n'ont pas été compromises.
Cet épisode illustre que même les grands acteurs du cloud peuvent être vulnérables à des attaques ciblées, et qu'une vigilance accrue est nécessaire sur les **comptes administratifs** et les **mécanismes d'authentification** web†source .

11.2 Implications pour *GeoChallenge Tracker*

Ces événements démontrent que les menaces sont multiples et peuvent provenir aussi bien des **dépendances** que des **services managés**.
Dans ce projet :
- seules les dépendances strictement nécessaires sont retenues,
- les versions des bibliothèques sont suivies et mises à jour régulièrement,
- des bonnes pratiques de sécurité sont appliquées (validation stricte des entrées, séparation des rôles, tokens JWT avec durées distinctes, index sécurisés en base),
- une attention particulière est portée à la configuration de MongoDB pour éviter tout accès non contrôlé.

La veille se poursuivra dans la durée, notamment via le suivi des CVE concernant FastAPI, Starlette, MongoDB, ainsi que les bibliothèques Python utilisées dans le projet.

11.3 Sources

- JFrog – Compromission de paquets npm (septembre 2025)
- NVD – Vulnérabilité FastAPI Guard (CVE-2025-46814)
- MongoDB – Incident de sécurité (décembre 2023)

- Snyk – Vulnérabilités FastAPI
- Article Medium – Vulnérabilité critique Starlette / FastAPI

Chaîne YouTube recommandée :
- Fireship – API Security in 100 Seconds (vidéo concise présentant les principaux risques et bonnes pratiques pour sécuriser des APIs modernes).

12 Lexique du géocaching (notions utiles au projet)

Afin de faciliter la lecture du dossier par des lecteurs non familiers du géocaching, je vous propose ci-après un petit lexique. Il rassemble uniquement les notions indispensables à la compréhension du projet *GeoChallenge Tracker*, en donnant une définition claire des termes utilisés dans le document.

- **Cache** : conteneur physique dissimulé par un joueur, localisé par ses coordonnées GPS et listé sur une plateforme de géocaching.
- **Cache trouvée** : cache qui a été effectivement localisée et loguée comme trouvée par un joueur. Ces données constituent la base de calcul pour évaluer la progression dans les challenges.
- **Géocacheur** : joueur pratiquant le géocaching, qui recherche des caches posées par d'autres et en publie éventuellement lui-même.
- **Owner / Poseur** : géocacheur ayant créé et publié une cache. Son rôle est de la maintenir en bon état et de gérer son descriptif en ligne.
- **Found it** : log positif indiquant qu'un joueur a trouvé la cache.
- **DNF (Did Not Find)** : log indiquant qu'un joueur a cherché la cache mais ne l'a pas trouvée.
- **Challenge Cache** : type particulier de cache qui ne peut être loguée comme trouvée que si le joueur a fait un *found it* **ET** a rempli certaines conditions prédéfinies (ex. avoir trouvé 100 caches d'un type donné).
- **Challenge (au sens du projet)** : défi géocaching, correspondant aux conditions fixées par une *challenge cache*. Dans *GeoChallenge Tracker*, les challenges sont importés depuis les caches détectées comme "challenge", puis suivis individuellement par chaque joueur.
- **Task (tâche de challenge)** : sous-condition définie par l'utilisateur pour un challenge donné, permettant de préciser la manière dont il souhaite interpréter ou suivre ce défi.
- **Progression** : enregistrement des étapes franchies par un joueur dans la réalisation d'un challenge, calculée à partir des caches trouvées et des tâches associées.
- **Target (cible)** : cache identifiée par l'application comme potentiellement utile pour compléter un challenge en cours, en fonction des conditions définies dans ses tâches.
- **D/T (Difficulty / Terrain)** : double cotation officielle indiquant la difficulté intellectuelle (D) et la difficulté physique/terrain (T) d'une cache. Ces valeurs peuvent être utilisées comme conditions dans les challenges.
- **Type (de cache)** : il existe différentes catégories de caches nécessitant des actions différentes : traditionnelles, mystères, virtuelles…
- **Taille (de cache)** : indique la taille physique de la cache, allant de micro à large.
- **Attribut (de cache)** : étiquette descriptive rattachée à une cache (ex. disponible la nuit, accessible aux enfants). Dans le projet, certains attributs servent de critères pour les tâches de challenge.
- **Marker clustering** : regroupement de points.
- **Grammaire AST** : représentation d'expressions ou conditions sous forme d'arbre syntaxique abstrait, permettant leur interprétation par le logiciel.