Running App

Documentation Technique et Architecture

API Sécurisée et Choix Technologiques

Équipe de Développement

Projet d'Application Mobile

Développeur Backend:	Julien Bonnet / Yanisse Ismaili
Développeur Frontend:	Julien Bonnet / Yanisse Ismaili
Architecture:	Julien Bonnet
Date:	10 juin 2025

Table des matières

1 Présentation du Projet Running App

1.1 Vue d'ensemble du projet

L'application Running App représente une solution mobile complète dédiée au suivi et à l'amélioration des performances en course à pied. Ce projet s'inscrit dans la mouvance actuelle des applications de fitness qui combinent technologie mobile, analyse de données et gamification pour encourager l'activité physique.

Notre application se distingue par une approche holistique qui ne se contente pas de mesurer les performances, mais propose également des courses personnalisées, un système de progression adaptatif et une interface utilisateur intuitive. L'objectif principal consiste à accompagner les utilisateurs dans leur parcours sportif, qu'ils soient débutants cherchant à adopter une routine d'exercice ou coureurs expérimentés visant l'optimisation de leurs performances.

Philosophie du projet

Running App vise à démocratiser l'accès à un coaching sportif personnalisé en utilisant les technologies modernes pour créer une expérience utilisateur engageante et motivante.

1.2 Objectifs et fonctionnalités principales

Le développement de Running App répond à plusieurs objectifs stratégiques qui guident nos choix techniques et fonctionnels.

L'objectif primaire consiste à fournir un outil de suivi complet permettant aux utilisateurs d'enregistrer leurs courses avec précision. Cette fonctionnalité s'appuie sur l'intégration des capteurs GPS du smartphone pour mesurer la distance parcourue, la vitesse instantanée et moyenne, ainsi que le temps d'effort. Ces données constituent la base de l'analyse des performances.

L'objectif secondaire porte sur la personnalisation de l'expérience utilisateur. Notre système propose des courses adaptées au niveau de chaque utilisateur, prenant en compte ses performances passées, ses objectifs déclarés et son rythme de progression. Cette approche personnalisée représente un avantage concurrentiel significatif par rapport aux applications généralistes.

L'objectif tertiaire concerne la motivation et l'engagement à long terme. Nous intégrons des éléments de gamification, des défis personnalisés et un système de progression visuelle pour maintenir l'engagement des utilisateurs sur la durée.

1.3 Public cible et cas d'usage

Notre analyse du marché nous a conduits à identifier trois segments d'utilisateurs principaux, chacun ayant des besoins spécifiques que notre application adresse.

Le premier segment comprend les débutants en course à pied, souvent intimidés par la complexité des applications existantes. Pour ces utilisateurs, nous proposons une interface simplifiée, des programmes d'initiation progressifs et des conseils adaptés. L'application les guide pas à pas dans la découverte de la course à pied, en évitant la surcharge d'informations qui pourrait les décourager.

Le deuxième segment rassemble les coureurs occasionnels qui cherchent à structurer leur pratique sans nécessairement viser la performance pure. Ces utilisateurs apprécient les fonctionnalités de suivi de base, les statistiques de progression et les suggestions de courses variées pour maintenir leur motivation.

Le troisième segment regroupe les coureurs réguliers et les athlètes amateur qui désirent optimiser leurs entraînements. Pour eux, nous fournissons des analyses détaillées, des métriques avancées et des programmes d'entraînement sophistiqués basés sur les principes de la science sportive.

1.4 Innovation et valeur ajoutée

Running App se démarque de la concurrence par plusieurs innovations techniques et fonctionnelles qui apportent une réelle valeur ajoutée aux utilisateurs.

Notre algorithme de recommandation de courses utilise l'apprentissage automatique pour analyser les performances passées et proposer des entraînements personnalisés. Contrairement aux applications qui proposent des programmes statiques, notre système s'adapte en temps réel aux progrès de l'utilisateur et ajuste les recommandations en conséquence.

L'intégration native avec les écosystèmes de santé des smartphones (HealthKit sur iOS, Google Fit sur Android) permet une synchronisation transparente des données de santé, offrant une vision globale de l'activité physique de l'utilisateur.

Notre approche de la sécurité des données va au-delà des standards minimaux. Nous implémentons un chiffrement de bout en bout pour les données sensibles et adoptons une politique de confidentialité transparente qui donne aux utilisateurs un contrôle total sur leurs informations personnelles.

Points forts de l'application

- Interface utilisateur intuitive adaptée à tous les niveaux
- Algorithmes de personnalisation basés sur l'IA
- Sécurité renforcée des données personnelles
- Intégration native avec les écosystèmes mobiles
- Architecture scalable pour une croissance future

1.5 Vision à long terme

Notre vision pour Running App dépasse le cadre d'une simple application de suivi de course. Nous envisageons une plateforme complète d'accompagnement sportif qui pourrait s'étendre à d'autres disciplines sportives tout en conservant la même philosophie de personnalisation et d'engagement utilisateur.

L'évolution future inclurait des fonctionnalités communautaires permettant aux utilisateurs de partager leurs expériences, de participer à des défis collectifs et de bénéficier du soutien d'une communauté sportive bienveillante. L'intégration de technologies émergentes comme la réalité augmentée pour des parcours interactifs ou l'intelligence artificielle pour un coaching vocal en temps réel constitue également des pistes d'évolution prometteuses.

Cette approche évolutive guide nos choix architecturaux actuels, en privilégiant la modularité et l'extensibilité pour faciliter l'intégration future de nouvelles fonctionnalités sans refondre l'ensemble du système.

2 Architecture Système et Conception

2.1 Vue d'ensemble de l'architecture

L'architecture de Running App repose sur une approche moderne en trois couches qui sépare clairement les responsabilités et facilite la maintenance ainsi que l'évolutivité du système. Cette conception architecturale s'inspire des meilleures pratiques de développement d'applications mobiles et web, en privilégiant la modularité, la scalabilité et la sécurité.

La première couche correspond à l'interface utilisateur développée en React Native, qui assure une expérience native sur les plateformes iOS et Android tout en permettant un développement unifié. Cette couche gère l'affichage des données, les interactions utilisateur et la communication avec les services du téléphone comme le GPS et les capteurs de mouvement.

La deuxième couche constitue l'API REST développée avec Flask, qui centralise toute la logique métier de l'application. Cette API expose des endpoints sécurisés pour la gestion des utilisateurs, l'enregistrement des courses, la génération de statistiques et la proposition de nouveaux entraînements. Elle implémente également tous les mécanismes d'authentification et d'autorisation nécessaires à la sécurité du système.

La troisième couche représente la base de données MySQL qui stocke de manière persistante toutes les informations de l'application. Cette base de données est conçue pour optimiser les performances lors des requêtes fréquentes tout en maintenant l'intégrité des données grâce à un système de contraintes et de relations bien définies.

Principe architectural

L'architecture en couches permet une séparation claire des préoccupations : l'interface se concentre sur l'expérience utilisateur, l'API gère la logique métier et la sécurité, tandis que la base de données optimise le stockage et la récupération des informations.

2.2 Architecture client-serveur détaillée

Notre système implémente une architecture client-serveur moderne qui tire parti des avantages de chaque plateforme tout en maintenant une cohérence fonctionnelle entre les environnements.

Du côté client, l'application React Native utilise une architecture basée sur des composants réutilisables qui encapsulent leur logique et leur présentation. Cette approche facilite la maintenance du code et permet une évolution incrémentale de l'interface utilisateur. Les composants communiquent entre eux via un système de propriétés et d'événements, tandis qu'un gestionnaire d'état global maintient la cohérence des données à travers l'application.

La gestion des données côté client s'appuie sur un système de cache intelligent qui minimise les appels réseau et améliore les performances perçues par l'utilisateur. Les données de course sont stockées localement pendant l'enregistrement pour éviter toute perte en cas de connectivité intermittente, puis synchronisées avec le serveur dès que la connexion est rétablie.

Du côté serveur, l'API Flask adopte une architecture modulaire organisée en blueprints qui regroupent les fonctionnalités par domaine métier. Cette organisation facilite la collaboration entre développeurs et permet une montée en charge progressive du système. Chaque blueprint encapsule ses routes, ses modèles de données et sa logique métier spécifique.

Le serveur implémente également un système de middleware qui traite de manière transversale des préoccupations comme l'authentification, la journalisation des requêtes, la gestion des erreurs et l'ajout des en-têtes de sécurité. Cette approche garantit une application cohérente de ces mécanismes sur l'ensemble de l'API.

2.3 Diagramme d'architecture système

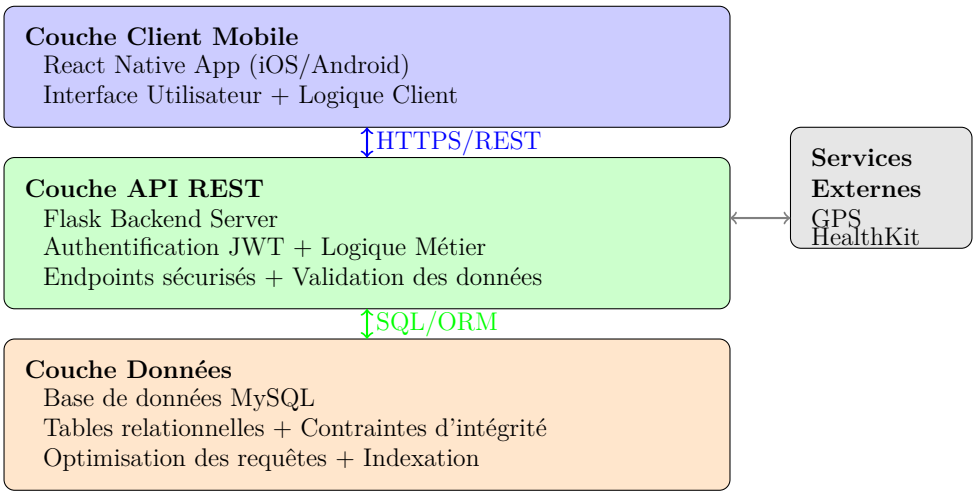

FIGURE 1 – Architecture en couches de Running App

2.4 Flux de données et communication

La communication entre les différentes couches de l'architecture suit des patterns établis qui garantissent la fiabilité et la performance du système dans son ensemble.

Lorsqu'un utilisateur lance l'application, le client React Native établit une session en contactant l'endpoint d'authentification de l'API. Cette requête initiale déclenche la validation des credentials et la génération d'un token JWT qui sera utilisé pour toutes les

requêtes subséquentes. Ce mécanisme assure que seuls les utilisateurs authentifiés peuvent accéder aux fonctionnalités de l'application.

Pendant une course, l'application mobile collecte en continu les données GPS et les métriques de performance. Ces informations sont stockées temporairement en mémoire locale pour éviter toute perte en cas d'interruption réseau. À la fin de la course, ou à intervalles réguliers si la connexion le permet, ces données sont transmises à l'API sous forme de requêtes POST sécurisées.

L'API reçoit ces données, les valide selon des règles métier prédéfinies, puis les transforme au format approprié pour le stockage en base de données. Cette étape inclut le calcul de métriques dérivées comme la vitesse moyenne, les calories brûlées estimées et l'analyse de la régularité de l'allure.

La génération de recommandations de courses suit un flux plus complexe qui implique l'analyse des performances historiques stockées en base de données. L'algorithme de recommandation interroge plusieurs tables pour construire un profil complet de l'utilisateur, puis génère des propositions d'entraînement adaptées à son niveau et à ses objectifs.

2.5 Considérations de performance et scalabilité

La conception de notre architecture prend en compte dès le départ les enjeux de performance et de scalabilité qui pourraient émerger avec la croissance de la base d'utilisateurs.

Du côté de la base de données, nous avons implémenté une stratégie d'indexation optimisée pour les requêtes les plus fréquentes. Les tables de courses et de statistiques utilisent des index composites qui accélèrent les recherches par utilisateur et par période temporelle. Cette optimisation est particulièrement importante pour la génération des tableaux de bord personnalisés qui agrègent de grandes quantités de données historiques.

L'API Flask est conçue pour supporter une montée en charge horizontale grâce à son architecture stateless. Chaque requête contient toutes les informations nécessaires à son traitement, ce qui permet de distribuer la charge sur plusieurs instances de serveur sans complexité supplémentaire. Les tokens JWT encapsulent les informations d'authentification et d'autorisation, éliminant le besoin de sessions serveur persistantes.

La gestion de la mémoire côté client utilise des techniques de lazy loading pour les listes de courses et de statistiques. Cette approche permet à l'application de rester réactive même avec un historique important, en ne chargeant que les données visibles par l'utilisateur et en préchargeant intelligemment les données susceptibles d'être consultées prochainement.

Considérations futures

L'architecture actuelle supporte une croissance significative, mais une évolution vers une architecture microservices pourrait être envisagée si le volume d'utilisateurs dépasse les capacités d'un serveur monolithique.

Notre approche de la mise en cache combine cache applicatif et cache base de données pour optimiser les temps de réponse. Les données statiques comme les catégories de courses sont mises en cache côté serveur, tandis que les données utilisateur fréquemment consultées bénéficient d'un cache intelligent côté client qui se synchronise automatiquement lors de modifications.

3 Modèle de Données et Structure de Base

3.1 Conception du modèle relationnel

La conception du modèle de données de Running App repose sur une approche relationnelle classique qui privilégie la cohérence, l'intégrité et l'efficacité des requêtes. Cette structure de données constitue le socle de notre application et doit pouvoir évoluer pour accompagner les futures fonctionnalités tout en maintenant les performances du système.

Notre modèle s'organise autour de cinq entités principales qui reflètent les concepts métier fondamentaux de l'application. Chaque entité encapsule un ensemble cohérent d'informations et définit des relations précises avec les autres entités du système. Cette approche nous permet de maintenir la normalisation de la base de données tout en optimisant les accès pour les cas d'usage les plus fréquents.

La table centrale `users` stocke les informations des utilisateurs et sert de point d'ancrage pour toutes les autres données personnalisées. Cette table contient non seulement les informations d'identification et de profil, mais aussi des métadonnées importantes comme les dates de création et de dernière modification qui facilitent l'audit et la maintenance du système.

Les données de performance sont organisées dans la table `runs` qui enregistre chaque session de course avec un niveau de détail suffisant pour permettre des analyses ultérieures. Cette table maintient un équilibre entre la richesse des informations stockées et l'efficacité des requêtes, en évitant la sur-normalisation qui pourrait pénaliser les performances.

Principe de conception

Notre modèle de données équilibre normalisation et performance en regroupant les informations fréquemment consultées ensemble tout en maintenant l'intégrité référentielle entre les entités.

3.2 Description détaillée des entités

La table `users` constitue le cœur de notre système d'authentification et de personnalisation. Elle stocke les informations essentielles de chaque utilisateur dans une structure optimisée pour les accès fréquents. Le champ `password_hash` utilise un algorithme de hachage sécurisé qui protège les mots de passe même en cas de compromission de la base de données. Le booléen `is_admin` permet une gestion simple mais efficace des droits d'administration, tandis que les timestamps de création et de modification facilitent l'audit et la maintenance.

La table `runs` capture l'essence de chaque session de course avec un niveau de détail

Champ	Type	Contraintes	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Identifiant unique
username	VARCHAR(50)	UNIQUE, NOT NULL	Nom d'utilisateur un
email	VARCHAR(100)	UNIQUE, NOT NULL	Adresse email unique
password_hash	VARCHAR(255)	NOT NULL	Hash sécurisé du m passe
first_name	VARCHAR(50)	NOT NULL	Prénom de l'utilisate
last_name	VARCHAR(50)	NOT NULL	Nom de famille
is_admin	BOOLEAN	DEFAULT FALSE	Indicateur de droits
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Date de création
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Dernière modificatio

Table 1 – Structure de la table users

qui permet des analyses sophistiquées tout en restant efficace pour les requêtes courantes. Les champs temporels utilisent le type TIMESTAMP pour une précision à la seconde, suffisante pour les analyses de performance tout en évitant la complexité de types plus précis. La durée stockée en secondes facilite les calculs arithmétiques, tandis que la distance en mètres permet une précision adaptée aux besoins de l'application.

Le champ `route_data` mérite une attention particulière car il stocke les coordonnées GPS sous format JSON, permettant une flexibilité maximale pour les analyses géographiques futures. Cette approche hybride, combinant structure relationnelle et flexibilité NoSQL, offre le meilleur des deux mondes pour notre cas d'usage spécifique.

Champ	Type	Contraintes	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Identifiant unique
user_id	INT	FOREIGN KEY, NOT NULL	Référence vers users.id
start_time	TIMESTAMP	NOT NULL	Heure de début de course
end_time	TIMESTAMP	NOT NULL	Heure de fin de course
duration	INT	NOT NULL	Durée en secondes
distance	DECIMAL(8,2)	NOT NULL	Distance en mètres
avg_speed	DECIMAL(5,2)	NULL	Vitesse moyenne en m/s
max_speed	DECIMAL(5,2)	NULL	Vitesse maximale en m/s
calories	INT	NULL	Calories brûlées estimées
route_data	JSON	NULL	Coordonnées GPS du parcours
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Date d'enregistrement

Table 2 – Structure de la table runs

3.3 Relations et contraintes d'intégrité

Les relations entre entités sont conçues pour maintenir la cohérence des données tout en optimisant les performances des requêtes les plus fréquentes. La relation principale

connecte chaque course à son utilisateur via une clé étrangère qui garantit l'intégrité référentielle.

Cette relation un-à-plusieurs entre `users` et `runs` utilise une contrainte `ON DELETE CASCADE` qui assure la suppression automatique de toutes les courses d'un utilisateur si son compte est supprimé. Cette approche évite les données orphelines tout en simplifiant la maintenance de la base de données.

Les contraintes d'intégrité incluent également des validations au niveau base de données qui complètent les validations applicatives. Par exemple, la durée d'une course ne peut pas être négative, et les timestamps de fin doivent être postérieurs aux timestamps de début. Ces contraintes constituent une couche de sécurité supplémentaire qui protège contre les erreurs de programmation et les tentatives de manipulation malveillante.

3.4 Diagramme Entité-Relation

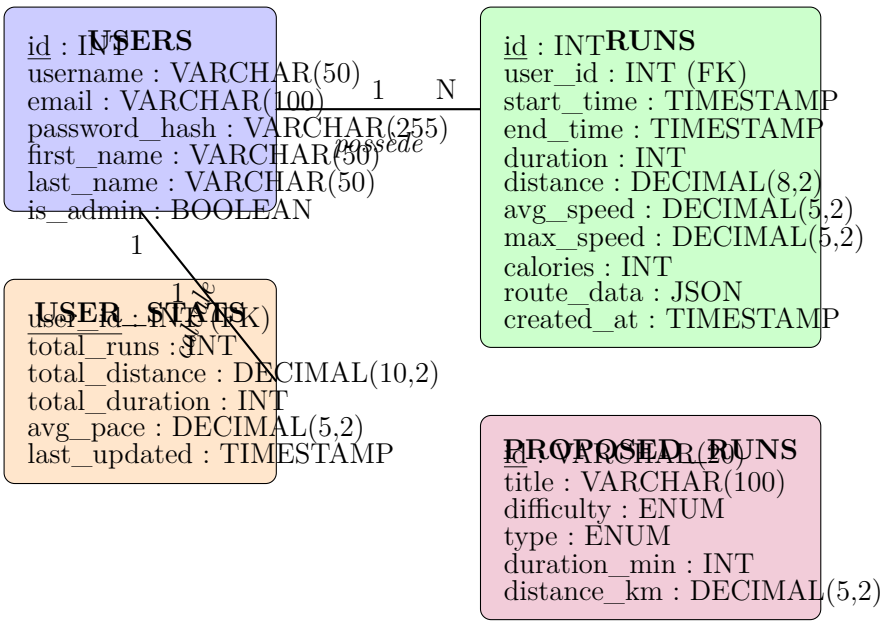

FIGURE 2 – Diagramme Entité-Relation de Running App

3.5 Optimisations et indexation

L'optimisation de la base de données constitue un aspect crucial pour maintenir des performances acceptables même avec une large base d'utilisateurs. Notre stratégie d'indexation cible les requêtes les plus fréquentes identifiées lors de l'analyse des cas d'usage.

L'index principal sur `runs(user_id, start_time)` optimise les requêtes d'historique personnel, qui représentent la majorité des accès à cette table. Cet index composite

permet de récupérer efficacement toutes les courses d'un utilisateur triées par date, ce qui correspond au cas d'usage le plus courant de consultation du tableau de bord personnel.

Un second index sur `runs(start_time)` facilite les requêtes statistiques globales et les analyses temporelles qui peuvent être nécessaires pour les fonctionnalités d'administration ou les rapports de performance du système. Cet index permet également d'optimiser les requêtes de recherche par période sans spécifier d'utilisateur particulier.

La table `users` bénéficie d'index uniques sur les champs `email` et `username`, non seulement pour garantir l'unicité mais aussi pour accélérer les requêtes d'authentification qui constituent un point critique pour l'expérience utilisateur.

Stratégie d'optimisation

- Index composites sur les colonnes fréquemment utilisées ensemble
- Partitionnement temporel envisagé pour les données historiques
- Dénormalisation sélective pour les statistiques couramment consultées
- Mise en cache applicative pour les données statiques

3.6 Évolution et migration du schéma

La gestion des évolutions du schéma de base de données constitue un défi important dans le cycle de vie d'une application. Notre approche utilise Flask-Migrate qui s'appuie sur Alembic pour gérer les migrations de manière versionnée et reproductible.

Chaque modification du modèle de données génère un script de migration qui peut être appliqué de manière incrémentale sur les environnements de développement, de test et de production. Cette approche garantit que tous les environnements restent synchronisés et que les déploiements peuvent être effectués en toute sécurité.

Les migrations incluent non seulement les modifications de structure mais aussi les transformations de données nécessaires lors de changements de format ou de contraintes. Par exemple, l'ajout d'une nouvelle colonne obligatoire s'accompagne automatiquement de la logique pour populer cette colonne avec des valeurs par défaut appropriées pour les enregistrements existants.

Notre stratégie de migration privilégie les changements rétrocompatibles autant que possible, en utilisant des techniques comme l'ajout de colonnes optionnelles suivi d'une phase de transition avant de rendre ces colonnes obligatoires. Cette approche minimise les risques lors des déploiements et permet des rollbacks en cas de problème.

3.7 Sauvegarde et récupération

La stratégie de sauvegarde protège contre la perte de données tout en maintenant des performances acceptables pour les opérations courantes. Notre approche combine sauvegardes complètes périodiques et sauvegardes incrémentales plus fréquentes pour optimiser l'espace de stockage et les temps de récupération.

Les sauvegardes complètes sont programmées quotidiennement pendant les heures de faible activité, tandis que les sauvegardes incrémentales capturent les modifications toutes les heures. Cette stratégie permet de restaurer le système avec une perte de données maximale d'une heure, ce qui constitue un compromis acceptable pour notre cas d'usage.

Les scripts de sauvegarde incluent des vérifications d'intégrité qui détectent automatiquement les corruptions potentielles et déclenchent des alertes en cas de problème. Ces vérifications portent sur la cohérence des contraintes d'intégrité, la validité des formats de données et la complétude des sauvegardes.

La procédure de récupération est documentée et testée régulièrement pour garantir qu'elle peut être exécutée rapidement en cas d'incident. Les tests de récupération incluent la restauration sur un environnement de test et la vérification que toutes les fonctionnalités de l'application restent opérationnelles après restauration.

4 API REST et Sécurité

4.1 Conception de l'API REST

L'API REST de Running App respecte les principes fondamentaux de l'architecture REST pour offrir une interface cohérente, prévisible et facilement maintenable. Cette approche facilite non seulement le développement de l'application mobile actuelle, mais prépare également l'intégration future avec d'autres clients ou services tiers.

Notre API s'organise autour de ressources clairement définies qui correspondent aux entités métier de l'application. Chaque ressource est accessible via une URL unique qui respecte les conventions REST, utilisant les verbes HTTP appropriés pour différencier les opérations. Cette conception permet une compréhension intuitive de l'API par les développeurs et facilite la documentation automatique.

La structure des endpoints suit une hiérarchie logique qui reflète les relations entre les données. Par exemple, les courses d'un utilisateur sont accessibles via `/api/users/{id}/runs`, créant une navigation naturelle dans les ressources liées. Cette approche améliore la cohérence de l'API et réduit la complexité côté client.

Les réponses de l'API utilisent un format JSON standardisé qui encapsule les données dans une structure cohérente incluant le statut de la requête, un message descriptif et les données proprement dites. Cette standardisation facilite le traitement côté client et améliore la robustesse de l'application face aux erreurs.

Principes REST appliqués

- URLs représentant des ressources plutôt que des actions
- Utilisation appropriée des verbes HTTP (GET, POST, PUT, DELETE)
- Réponses stateless sans état côté serveur
- Format JSON standardisé pour toutes les réponses
- Codes de statut HTTP significatifs et cohérents

4.2 Documentation des endpoints principaux

L'API expose plusieurs groupes d'endpoints organisés par domaine fonctionnel, chacun gérant un aspect spécifique de l'application. Cette organisation modulaire facilite la maintenance et permet une évolution indépendante des différentes fonctionnalités.

4.2.1 Endpoints d'authentification

Les endpoints d'authentification gèrent l'inscription, la connexion et la gestion des sessions utilisateur. Ces endpoints constituent le point d'entrée obligatoire pour accéder aux fonctionnalités personnalisées de l'application.

Endpoint	Méthode	Description	Auth
/api/auth/register	POST	Inscription d'un nouvel utilisateur	Non
/api/auth/login	POST	Connexion et génération de token JWT	Non
/api/auth/logout	POST	Invalidation du token utilisateur	Oui
/api/auth/refresh	POST	Renouvellement du token JWT	Oui
/api/auth/forgot-password	POST	Demande de réinitialisation de mot de passe	Non

Table 3 – Endpoints d'authentification

4.2.2 Endpoints de gestion des courses

Ces endpoints permettent l'enregistrement, la consultation et la modification des données de course. Ils constituent le cœur fonctionnel de l'application et sont optimisés pour les accès fréquents.

Endpoint	Méthode	Description	Auth
/api/runs	GET	Liste paginée des courses de l'utilisateur	Oui
/api/runs	POST	Enregistrement d'une nouvelle course	Oui
/api/runs/{id}	GET	Détails d'une course spécifique	Oui
/api/runs/{id}	PUT	Modification d'une course existante	Oui
/api/runs/{id}	DELETE	Suppression d'une course	Oui
/api/runs/stats	GET	Statistiques personnalisées des courses	Oui

Table 4 – Endpoints de gestion des courses

4.2.3 Endpoints des courses proposées

Ces endpoints récents fournissent des recommandations de courses personnalisées basées sur le profil et l'historique de l'utilisateur.

Endpoint	Méthode	Description	Auth
/api/proposed-runs	GET	Liste des courses recommandées	Non
/api/proposed-runs/categories	GET	Catégories de courses disponibles	Non
/api/proposed-runs/{id}	GET	Détails d'une course proposée	Non

Table 5 – Endpoints des courses proposées

4.3 Authentification et autorisation JWT

L'implémentation de l'authentification s'appuie sur les JSON Web Tokens (JWT) qui offrent un mécanisme stateless parfaitement adapté aux architectures REST. Cette approche élimine le besoin de maintenir des sessions côté serveur tout en conservant un niveau de sécurité élevé.

Lors de l'authentification réussie d'un utilisateur, le serveur génère un token JWT qui encapsule les informations d'identification et d'autorisation nécessaires. Ce token est signé cryptographiquement avec une clé secrète connue uniquement du serveur, garantissant son intégrité et son authenticité. Le client stocke ce token de manière sécurisée et l'inclut dans l'en-tête Authorization de toutes les requêtes subséquentes.

La structure du token JWT inclut plusieurs claims standard et personnalisés qui facilitent l'autorisation granulaire. Le claim `user_id` identifie uniquement l'utilisateur, tandis que `is_admin` permet de différencier les utilisateurs ordinaires des administrateurs. L'expiration du token est configurée pour équilibrer sécurité et confort d'utilisation, avec une durée de vie de 24 heures renouvelable automatiquement.

Listing 1 – Structure du payload JWT

```
{
  "user_id": 42,
  "username": "john_runner",
  "email": "john@example.com",
  "is_admin": false,
  "iat": 1699123456,
  "exp": 1699209856
}
```

Le middleware d'authentification intercepte toutes les requêtes vers les endpoints protégés et vérifie la validité du token JWT. Cette vérification inclut la validation de la signature cryptographique, la vérification de l'expiration et l'extraction des informations d'autorisation. En cas de token invalide ou expiré, le middleware retourne une erreur HTTP 401 avec un message explicite guidant le client vers une nouvelle authentification.

4.4 Sécurisation des communications

La sécurisation des communications entre le client mobile et l'API constitue un aspect critique qui protège les données sensibles des utilisateurs contre l'interception et la manipulation malveillante.

Toutes les communications utilisent exclusivement le protocole HTTPS qui chiffre l'intégralité des échanges entre le client et le serveur. Cette protection cryptographique empêche l'écoute passive des communications et garantit l'intégrité des données transmises.

Le certificat SSL/TLS est configuré avec des algorithmes de chiffrement modernes et une taille de clé suffisante pour résister aux attaques actuelles.

L'API implémente des en-têtes de sécurité additionnels qui renforcent la protection contre diverses attaques web. L'en-tête `X-Content-Type-Options: nosniff` empêche les navigateurs de deviner le type MIME des réponses, tandis que `X-Frame-Options: DENY` protège contre les attaques de clickjacking. Ces mesures constituent une défense en profondeur qui complète le chiffrement de transport.

La validation stricte des entrées côté serveur constitue une barrière supplémentaire contre les tentatives d'injection et de manipulation de données. Chaque endpoint valide rigoureusement les paramètres reçus selon des schémas prédéfinis, rejetant automatiquement les requêtes malformées ou suspectes. Cette validation porte sur le format, la taille, le type et les valeurs autorisées pour chaque paramètre.

Mesures de sécurité appliquées

- Chiffrement HTTPS obligatoire pour toutes les communications
- Tokens JWT avec expiration et renouvellement automatique
- Validation stricte de toutes les entrées utilisateur
- En-têtes de sécurité HTTP pour protection additionnelle
- Hachage sécurisé des mots de passe avec salt
- Protection contre les attaques CSRF et XSS

4.5 Gestion des erreurs et codes de retour

La gestion cohérente des erreurs améliore significativement l'expérience développeur et facilite le débogage des applications cliente. Notre API utilise les codes de statut HTTP standard de manière appropriée et retourne des messages d'erreur structurés qui aident à identifier et résoudre les problèmes.

Les erreurs client (4xx) distinguent clairement les différents types de problèmes : 400 pour les requêtes malformées, 401 pour les problèmes d'authentification, 403 pour les violations d'autorisation et 404 pour les ressources inexistantes. Cette granularité permet au client de réagir appropriément selon le type d'erreur rencontré.

Les erreurs serveur (5xx) sont loggées de manière détaillée côté serveur pour faciliter le diagnostic, tout en retournant des messages génériques au client pour éviter la divulgation d'informations sensibles sur l'infrastructure. Cette approche équilibre transparence pour le développement et sécurité pour la production.

Listing 2 – Format standardisé des réponses d'erreur

```
{
  "status": "error",
  "message": "Validation failed for the provided data",
  "errors": {
    "email": "Invalid email format",
    "password": "Password must be at least 8 characters"
  },
  "timestamp": "2024-01-15T10:30:00Z"
}
```

4.6 Monitoring et logging

Le monitoring de l'API fournit une visibilité essentielle sur les performances, la disponibilité et la sécurité du système. Notre approche combine logging applicatif détaillé et métriques de performance pour détecter proactivement les problèmes potentiels.

Chaque requête est loggée avec des informations contextuelles incluant l'utilisateur, l'endpoint appelé, les paramètres principaux, le temps de traitement et le code de retour. Ces logs structurés facilitent l'analyse automatique et permettent de détecter des patterns d'usage anormaux qui pourraient indiquer des tentatives d'attaque.

Les métriques de performance incluent les temps de réponse par endpoint, le taux d'erreur, le nombre de requêtes par utilisateur et la charge système. Ces indicateurs sont agrégés en temps réel et comparés à des seuils prédéfinis pour déclencher des alertes en cas de dégradation des performances.

La corrélation des logs avec les métriques système permet d'identifier rapidement les causes racines des problèmes de performance. Par exemple, une augmentation du temps de réponse des endpoints de course peut être corrélée avec une charge élevée de la base de données, orientant immédiatement les efforts de résolution vers l'optimisation des requêtes SQL.

5 Technologies et Justification des Choix Techniques

5.1 Vue d'ensemble de la stack technique

Le choix de notre stack technologique résulte d'une analyse approfondie des besoins du projet, des contraintes de performance, de la maintenabilité du code et de l'évolutivité du système. Cette sélection privilégie des technologies matures et largement adoptées qui garantissent la stabilité du projet tout en offrant la flexibilité nécessaire pour les évolutions futures.

Notre approche technique s'articule autour de trois piliers principaux : la performance pour assurer une expérience utilisateur fluide, la sécurité pour protéger les données sensibles des utilisateurs, et la maintenabilité pour faciliter les évolutions et corrections futures. Ces critères ont guidé chaque décision technique, depuis le choix du framework mobile jusqu'à la configuration de la base de données.

La cohérence entre les technologies choisies facilite l'intégration et réduit la complexité globale du système. L'utilisation de JavaScript/TypeScript côté client et Python côté serveur offre un équilibre optimal entre productivité de développement et performance d'exécution, tout en permettant à l'équipe de maîtriser un nombre limité de langages.

Critères de sélection technologique

- Maturité et stabilité des technologies
- Performance et scalabilité du système
- Facilité de maintenance et d'évolution
- Disponibilité des compétences dans l'équipe
- Écosystème et support communautaire
- Coût total de possession (TCO)

5.2 Frontend mobile : React Native

Le choix de React Native pour le développement de l'application mobile répond à plusieurs impératifs stratégiques qui optimisent à la fois le time-to-market et la qualité du produit final.

React Native permet de développer simultanément pour iOS et Android avec une base de code largement partagée, réduisant significativement les coûts de développement et de maintenance. Cette approche cross-platform ne compromet pas la qualité de l'expérience utilisateur grâce à l'utilisation de composants natifs réels plutôt que d'une WebView, garantissant des performances proche du natif pur.

L'écosystème React Native offre un accès simplifié aux fonctionnalités spécifiques des smartphones nécessaires à notre application. L'intégration avec les services de géolocalisation, les capteurs de mouvement et les APIs de santé (HealthKit, Google Fit) s'effectue via des modules bien maintenus qui encapsulent la complexité des APIs natives.

Avantage	**Impact sur le projet**
Code partagé	Réduction de 60% du temps de développement mobile
Performance	Rendu natif garantissant 60fps pour les animations
Hot Reload	Cycle de développement accéléré et débogage facilité
Écosystème	Large gamme de modules pour fonctionnalités spécialisées
Compétences	Réutilisation des connaissances React de l'équipe

Table 6 – Avantages de React Native pour Running App

La gestion de l'état de l'application utilise les hooks React modernes qui simplifient la logique de composants tout en maintenant des performances optimales. Cette approche évite la complexité d'un gestionnaire d'état externe pour notre cas d'usage tout en préparant une migration future vers Redux si la complexité de l'application l'exige.

L'architecture de navigation s'appuie sur React Navigation, une solution mature qui gère élégamment les transitions entre écrans et l'état de navigation. Cette bibliothèque offre une API déclarative qui s'intègre naturellement avec l'approche composant de React tout en supportant les patterns de navigation natifs de chaque plateforme.

5.3 Backend API : Flask et Python

Flask constitue le cœur de notre backend grâce à sa philosophie minimaliste qui permet de construire exactement l'architecture nécessaire sans surcharge inutile. Cette approche micro-framework offre une flexibilité maximale pour adapter le système aux besoins spécifiques de Running App.

Python apporte plusieurs avantages déterminants pour notre projet. La richesse de son écosystème facilite l'intégration de fonctionnalités avancées comme l'analyse de données pour les recommandations de courses ou le machine learning pour la personnalisation future. La syntaxe claire et expressive de Python améliore la maintenabilité du code et réduit les risques d'erreurs de développement.

Listing 3 – Exemple de structure Blueprint Flask

```
from flask import Blueprint, jsonify, request
from flask_jwt_extended import jwt_required, get_jwt_identity

runs_bp = Blueprint('runs', __name__)

```

```python
@runs_bp.route('', methods=['GET'])
@jwt_required()
def get_user_runs():
    user_id = get_jwt_identity()
    # Logique m tier claire et concise
    runs = Run.query.filter_by(user_id=user_id).all()
    return jsonify({
        "status": "success",
        "data": [run.to_dict() for run in runs]
    })
```

L'organisation modulaire en blueprints facilite la collaboration en équipe en permettant de développer différentes fonctionnalités de manière indépendante. Cette architecture favorise également les tests unitaires en isolant clairement les responsabilités de chaque module.

Flask-SQLAlchemy fournit une couche d'abstraction élégante pour les interactions avec la base de données. L'ORM simplifie les requêtes complexes tout en offrant la possibilité d'optimiser les performances avec du SQL natif quand nécessaire. Cette flexibilité s'avère particulièrement utile pour les requêtes d'agrégation des statistiques de course.

Les extensions Flask enrichissent le framework de base avec des fonctionnalités essentielles comme l'authentification JWT (Flask-JWT-Extended), la gestion des migrations (Flask-Migrate) et la validation des données (Flask-Marshmallow). Cette approche modulaire permet d'ajouter uniquement les fonctionnalités nécessaires sans alourdir l'application.

5.4 Base de données : MySQL

MySQL s'impose comme choix naturel pour notre système grâce à sa maturité, ses performances éprouvées et son excellente intégration avec l'écosystème Python/Flask. Cette base de données relationnelle offre la robustesse nécessaire pour gérer les données critiques de l'application tout en maintenant des performances optimales même avec une large base d'utilisateurs.

La cohérence ACID de MySQL garantit l'intégrité des données lors des opérations concurrentes, un aspect crucial pour une application mobile où plusieurs utilisateurs peuvent interagir simultanément avec le système. Cette fiabilité s'avère particulièrement importante pour l'enregistrement des courses et la gestion des comptes utilisateur où aucune perte de données n'est acceptable.

Les capacités d'optimisation avancées de MySQL, notamment ses algorithmes d'indexation sophistiqués et son optimiseur de requêtes, permettent de maintenir des temps de réponse rapides même avec des volumes de données importants. Le support natif des index JSON facilite le stockage et la recherche dans les données de parcours GPS sans

compromettre les performances des requêtes relationnelles traditionnelles.

Caractéristique	**Bénéfice pour Running App**
Transactions ACID	Intégrité garantie des données de course et utilisateur
Optimiseur de requêtes	Performance optimale pour les statistiques complexes
Support JSON	Stockage flexible des données GPS et métadonnées
Réplication	Haute disponibilité et sauvegarde automatique
Écosystème mature	Outils d'administration et monitoring éprouvés

TABLE 7 – Avantages de MySQL pour notre architecture

La stratégie de réplication MySQL permet de configurer facilement une architecture haute disponibilité avec des serveurs de lecture dédiés pour les requêtes analytiques. Cette séparation des charges améliore les performances globales en dédiant le serveur principal aux opérations d'écriture critiques tout en distribuant les lectures sur plusieurs instances.

5.5 Outils de développement et infrastructure

L'outillage de développement constitue un facteur déterminant pour la productivité de l'équipe et la qualité du code produit. Notre sélection d'outils privilégie l'intégration et l'automatisation pour réduire les tâches répétitives et minimiser les erreurs humaines.

Git avec GitHub centralise la gestion de versions en offrant des fonctionnalités collaboratives avancées comme les pull requests, les revues de code et l'intégration continue. Cette plateforme facilite le suivi des modifications, la résolution des conflits et la maintenance de multiples branches de développement parallèles.

L'environnement de développement s'appuie sur des conteneurs Docker qui garantissent la cohérence entre les postes de développement, les environnements de test et la production. Cette approche élimine les problèmes classiques de "ça marche sur ma machine" et facilite l'onboarding de nouveaux développeurs dans l'équipe.

Expo CLI simplifie considérablement le développement React Native en fournissant un environnement de développement unifié qui gère automatiquement la compilation, le déploiement sur les appareils de test et la publication sur les app stores. Cette intégration réduit la complexité technique et permet aux développeurs de se concentrer sur la logique métier.

Le monitoring de production utilise une combinaison d'outils open source et de services managés pour surveiller les performances, détecter les erreurs et analyser l'usage. Sentry capture automatiquement les exceptions côté client et serveur avec un contexte détaillé qui facilite le débogage. Prometheus collecte les métriques système et applicatives pour alimenter des tableaux de bord Grafana qui visualisent l'état du système en temps réel.

5.6 Justification des choix architecturaux

Chaque décision architecturale résulte d'une analyse coût-bénéfice qui prend en compte les contraintes spécifiques de notre projet. L'architecture monolithique choisie pour le backend, bien que moins trendy que les microservices, s'avère plus appropriée pour notre équipe réduite et notre domaine métier cohérent.

Cette approche monolithique simplifie considérablement le déploiement, le debugging et la gestion des transactions qui s'étendent sur plusieurs entités. La complexité additionnelle des microservices ne se justifie pas à notre échelle actuelle, tout en gardant la possibilité d'évoluer vers cette architecture si la croissance l'exige.

Le choix du stockage relationnel plutôt que NoSQL reflète la nature structurée de nos données et l'importance des relations entre entités. Les données de course présentent un schéma stable et des besoins de cohérence forte qui correspondent parfaitement au modèle relationnel. L'ajout du support JSON dans MySQL offre la flexibilité nécessaire pour les données semi-structurées sans sacrifier les avantages du relationnel.

L'authentification basée sur JWT plutôt que sur des sessions serveur s'aligne avec notre architecture stateless et facilite la scalabilité horizontale future. Cette approche simplifie également l'architecture en éliminant le besoin d'un store de sessions partagé entre les instances de serveur.

5.7 Stratégie de déploiement et DevOps

Notre stratégie de déploiement privilégie la simplicité et la fiabilité avec un pipeline CI/CD automatisé qui réduit les risques d'erreurs humaines et accélère les cycles de livraison.

Le pipeline de déploiement s'articule autour de GitHub Actions qui orchestrent automatiquement les tests, la construction des artefacts et le déploiement selon les branches. Cette intégration native avec notre repository simplifie la configuration et évite la complexité d'outils externes supplémentaires.

L'environnement de production utilise une approche de déploiement blue-green qui permet des mises à jour sans interruption de service. Cette stratégie maintient deux environnements identiques dont un seul reçoit le trafic utilisateur, permettant de basculer instantanément en cas de problème avec la nouvelle version.

La conteneurisation avec Docker facilite la portabilité entre environnements et simplifie la gestion des dépendances. Chaque composant de l'application est packagé avec ses dépendances exactes, garantissant un comportement identique quel que soit l'environnement d'exécution.

5.8 Sécurité et conformité

La sécurité constitue une préoccupation transversale qui influence chaque aspect de notre architecture technique. Notre approche de sécurité en profondeur combine plusieurs couches de protection pour protéger les données utilisateur et maintenir la confiance.

Le chiffrement des données en transit utilise TLS 1.3 avec des suites cryptographiques modernes qui résistent aux attaques actuelles. Cette protection s'étend à toutes les communications, incluant les connexions à la base de données et les APIs externes.

La gestion des secrets utilise des variables d'environnement et un système de vault pour éviter le stockage de credentials en dur dans le code. Cette approche facilite la rotation des clés et réduit les risques de compromission lors des déploiements.

La validation des entrées s'effectue à plusieurs niveaux avec des bibliothèques spécialisées qui empêchent les injections SQL, les attaques XSS et autres vecteurs d'attaque classiques. Cette validation multicouche garantit qu'aucune donnée malveillante ne peut atteindre les couches sensibles du système.

Les audits de sécurité automatisés analysent régulièrement les dépendances pour détecter les vulnérabilités connues et proposer des mises à jour. Cette surveillance proactive permet de maintenir un niveau de sécurité élevé même avec l'évolution constante de l'écosystème logiciel.

Mesures de sécurité implémentées

Notre architecture intègre des mesures de sécurité à tous les niveaux, depuis le chiffrement des communications jusqu'à la validation stricte des entrées utilisateur. Cette approche multicouche garantit une protection robuste des données sensibles tout en maintenant une expérience utilisateur fluide. Les audits réguliers et la surveillance continue permettent de détecter et corriger proactivement les vulnérabilités potentielles.

6 Répartition des Rôles et Organisation de l'Équipe

6.1 Structure organisationnelle du projet

L'organisation de notre équipe de développement reflète une approche pragmatique qui maximise l'efficacité tout en maintenant une couverture complète des compétences nécessaires au projet. Cette structure équilibre spécialisation technique et polyvalence pour permettre une collaboration fluide et une montée en compétences mutuelle entre les membres de l'équipe.

Notre équipe adopte une organisation matricielle où chaque membre possède une responsabilité principale tout en contribuant aux autres aspects du projet selon les besoins et les phases de développement. Cette flexibilité s'avère particulièrement précieuse dans un projet d'envergure limitée où l'adaptabilité prime sur la rigidité organisationnelle.

La communication entre les membres s'organise autour de rituels agiles adaptés à notre contexte, notamment des points de synchronisation réguliers qui permettent de coordonner les efforts et d'identifier rapidement les blocages potentiels. Cette approche collaborative favorise le partage de connaissances et garantit la cohérence technique de l'ensemble du projet.

Philosophie organisationnelle

Notre organisation privilégie la collaboration et le partage de connaissances plutôt que les silos techniques. Chaque membre de l'équipe comprend l'architecture globale et peut contribuer à différents aspects du projet, garantissant une meilleure résilience et une montée en compétences collective.

6.2 Responsabilités par domaine technique

La répartition des responsabilités techniques s'articule autour des compétences de chaque membre tout en assurant une couverture complète de tous les aspects du projet. Cette organisation permet à chaque développeur de se concentrer sur son domaine d'expertise tout en maintenant une vision globale du système.

6.2.1 Développement Backend et Architecture API

Le responsable backend assume la conception et l'implémentation de l'ensemble de l'architecture serveur, depuis la définition des endpoints REST jusqu'à l'optimisation des performances de la base de données. Cette responsabilité englobe la sécurisation de l'API, la gestion de l'authentification JWT et l'implémentation des algorithmes métier qui constituent le cœur fonctionnel de l'application.

La conception de la base de données relève également de cette responsabilité, incluant la modélisation des entités, l'optimisation des requêtes et la mise en place des stratégies de sauvegarde. Cette expertise technique s'étend à la configuration de l'environnement de production et à l'implémentation des mesures de monitoring qui garantissent la stabilité du système.

L'implémentation des fonctionnalités de recommandation de courses constitue un défi technique particulier qui nécessite une compréhension approfondie des besoins utilisateur et des algorithmes d'analyse de données. Cette responsabilité inclut la conception des critères de recommandation et leur évolution future vers des approches plus sophistiquées basées sur l'apprentissage automatique.

6.2.2 Développement Frontend Mobile

Le développement de l'application mobile React Native concentre les efforts sur l'expérience utilisateur et l'intégration avec les fonctionnalités natives des smartphones. Cette responsabilité englobe la conception de l'interface utilisateur, l'implémentation de la navigation et l'optimisation des performances pour garantir une expérience fluide sur tous les appareils supportés.

L'intégration avec les APIs natives constitue un aspect technique critique qui nécessite une expertise spécifique aux plateformes mobiles. Cette responsabilité inclut l'accès aux services de géolocalisation, l'interfaçage avec les capteurs de mouvement et la synchronisation avec les écosystèmes de santé des différentes plateformes.

La gestion de l'état côté client et l'implémentation des mécanismes de cache représentent des défis techniques importants pour maintenir les performances de l'application même en cas de connectivité limitée. Cette expertise s'étend à l'optimisation de la consommation de batterie et à la gestion intelligente des ressources système.

6.2.3 Architecture et Intégration Système

La responsabilité architecturale assure la cohérence technique de l'ensemble du projet en définissant les standards de développement, les patterns d'intégration et les stratégies d'évolution du système. Cette vision transversale garantit que les développements dans chaque domaine technique s'intègrent harmonieusement dans l'architecture globale.

La mise en place des environnements de développement, de test et de production relève de cette responsabilité, incluant la configuration des pipelines CI/CD et l'automatisation des déploiements. Cette expertise technique facilite la collaboration en standardisant les outils et les processus de développement.

L'évaluation et l'intégration de nouvelles technologies constituent un aspect prospectif important qui prépare l'évolution future du projet. Cette responsabilité inclut la veille

technologique, l'évaluation des alternatives techniques et la planification des migrations nécessaires pour maintenir la modernité du système.

6.3 Coordination et communication

La coordination efficace entre les membres de l'équipe constitue un facteur critique de succès qui nécessite des processus bien définis et des outils adaptés. Notre approche privilégie la communication directe et les outils collaboratifs qui facilitent le partage d'informations et la résolution rapide des problèmes.

Les réunions de synchronisation hebdomadaires permettent de faire le point sur l'avancement de chaque composant, d'identifier les dépendances entre tâches et de planifier les prochaines étapes. Ces sessions incluent une revue technique qui assure la cohérence des développements et facilite le partage de connaissances entre domaines techniques.

La documentation technique partagée centralise les décisions architecturales, les standards de développement et les procédures opérationnelles. Cette base de connaissances évolue en continu et facilite l'onboarding de nouveaux contributeurs tout en servant de référence pour les décisions futures.

L'utilisation d'outils collaboratifs comme Slack pour la communication quotidienne et Trello pour le suivi des tâches améliore la transparence et permet à chaque membre de l'équipe de comprendre l'état global du projet. Cette visibilité facilite l'entraide et l'identification proactive des risques.

6.4 Matrice des compétences et formations

L'identification claire des compétences présentes dans l'équipe et des besoins de formation permet d'optimiser la répartition des tâches et de planifier le développement des compétences nécessaires au projet.

Compétence	**Membre 1**	**Membre 2**	**Membre 3**	**Besoin**
Python/Flask	Expert	Intermédiaire	Débutant	Formation
React Native	Débutant	Expert	Intermédiaire	-
Base de données	Expert	Débutant	Intermédiaire	Formation
DevOps/Déploiement	Intermédiaire	Débutant	Expert	-
UI/UX Design	Débutant	Expert	Débutant	Amélioration
Sécurité	Intermédiaire	Débutant	Expert	Formation
Tests automatisés	Expert	Intermédiaire	Débutant	Formation

Table 8 – Matrice des compétences de l'équipe

Cette matrice guide les décisions d'affectation des tâches en s'appuyant sur les expertises existantes tout en identifiant les opportunités de montée en compétences. Les formations

ciblées permettent de combler les lacunes identifiées et d'améliorer la polyvalence de l'équipe.

Le partage de connaissances s'organise autour de sessions techniques internes où les experts présentent leurs domaines d'expertise aux autres membres. Cette approche favorise la diffusion des bonnes pratiques et prépare l'équipe à gérer collectivement tous les aspects techniques du projet.

6.5 Gestion des risques et continuité

La gestion des risques liés aux ressources humaines constitue un aspect important de l'organisation projet qui nécessite une planification proactive. Notre approche identifie les risques potentiels et met en place des mesures préventives pour assurer la continuité du projet.

Le partage de connaissances entre membres de l'équipe constitue la première ligne de défense contre les risques de dépendance excessive envers un individu particulier. Cette approche s'appuie sur la documentation technique détaillée et les sessions de formation croisée qui permettent à chaque membre de comprendre l'ensemble du système.

La redondance des compétences critiques guide nos choix de formation et de répartition des tâches. Chaque aspect technique important du projet est maîtrisé par au moins deux membres de l'équipe, garantissant la capacité de continuer le développement même en cas d'indisponibilité temporaire.

La planification des congés et des absences prend en compte les phases critiques du projet et s'assure qu'aucune période importante ne se retrouve avec une couverture insuffisante des compétences techniques nécessaires.

6.6 Évolution et montée en compétences

L'évolution des compétences de l'équipe accompagne naturellement la croissance du projet et l'introduction de nouvelles technologies. Notre approche de formation continue permet à chaque membre de développer son expertise tout en contribuant au succès collectif.

La veille technologique constitue une responsabilité partagée qui permet à l'équipe de rester informée des évolutions de l'écosystème technique. Cette activité inclut l'évaluation de nouvelles bibliothèques, l'analyse des meilleures pratiques émergentes et l'identification des opportunités d'amélioration du système existant.

Les projets personnels et les contributions open source encouragent l'expérimentation et le développement de nouvelles compétences qui bénéficient ensuite au projet principal. Cette approche favorise l'innovation et maintient la motivation technique de l'équipe.

La participation à des conférences techniques et à des formations externes enrichit les compétences de l'équipe et apporte des perspectives nouvelles sur les défis techniques du projet. Ces investissements en formation se traduisent par une amélioration de la qualité technique et une accélération des développements futurs.

Organisation optimisée

Notre organisation équilibre spécialisation et polyvalence pour maximiser l'efficacité tout en garantissant la résilience. Le partage de connaissances et la formation continue préparent l'équipe aux évolutions futures du projet et maintiennent un haut niveau de motivation technique.

7 Annexes Techniques

7.1 Configuration et installation

Cette section fournit les informations détaillées nécessaires pour reproduire l'environnement de développement et comprendre les dépendances techniques du projet Running App. Ces informations constituent une ressource précieuse pour l'évaluation technique et la poursuite éventuelle du développement.

L'environnement de développement de Running App nécessite une configuration spécifique qui équilibre simplicité d'installation et robustesse technique. Cette approche garantit que les développeurs peuvent rapidement démarrer leur travail tout en bénéficiant d'un environnement stable et reproductible.

7.1.1 Prérequis système

L'installation complète de l'environnement de développement nécessite plusieurs composants qui doivent être configurés dans un ordre spécifique pour garantir le bon fonctionnement de l'ensemble du système. Cette séquence d'installation évite les conflits de dépendances et assure une configuration optimale.

Composant	**Version**	**Description**
Node.js	18.x ou supérieur	Runtime JavaScript pour React Native
Python	3.9 ou supérieur	Interpréteur pour le backend Flask
MySQL	8.0 ou supérieur	Base de données relationnelle
Git	2.30 ou supérieur	Gestionnaire de versions
Android Studio	Dernière version	IDE pour le développement Android
Xcode	14 ou supérieur	IDE pour le développement iOS (macOS uniquement)

TABLE 9 – Prérequis techniques pour l'environnement de développement

La configuration de ces outils suit une séquence logique qui évite les conflits de dépendances et assure une installation stable. Python constitue le point de départ car Flask et ses extensions en dépendent directement. La configuration de MySQL nécessite une attention particulière aux droits d'accès et à la configuration de sécurité pour permettre le développement local tout en préparant le déploiement en production.

Node.js et ses outils associés forment l'écosystème React Native qui permet le développement cross-platform. L'installation d'Android Studio et Xcode complète l'environnement en fournissant les SDK nécessaires pour compiler et tester l'application sur les appareils mobiles réels et les simulateurs.

7.1.2 Installation du backend

L'installation du backend Flask nécessite la création d'un environnement virtuel Python qui isole les dépendances du projet et évite les conflits avec d'autres applications Python installées sur le système. Cette approche représente une bonne pratique qui facilite la maintenance et le déploiement.

Listing 4 – Installation du backend Flask

```
# Cr ation de l'environnement virtuel
python -m venv venv

# Activation de l'environnement (Windows)
venv\Scripts\activate

# Activation de l'environnement (macOS/Linux)
source venv/bin/activate

# Installation des d pendances
pip install -r requirements.txt

# Configuration de la base de donn es
python create_tables.py

# Cr ation d'un utilisateur administrateur
python scripts/create_admin.py admin admin@test.com Admin123!

# Lancement du serveur de d veloppement
python run.py
```

Le fichier requirements.txt contient toutes les dépendances Python nécessaires avec leurs versions spécifiques pour garantir la reproductibilité de l'environnement. Cette approche évite les problèmes de compatibilité qui pourraient survenir avec des versions différentes des bibliothèques. La spécification des versions exactes assure que tous les développeurs travaillent avec le même environnement technique.

La création des tables de base de données s'effectue via un script dédié qui utilise les modèles SQLAlchemy pour générer automatiquement la structure appropriée. Cette automatisation réduit les erreurs manuelles et facilite la mise en place d'environnements de développement multiples.

7.1.3 Installation du frontend mobile

L'installation de l'application React Native s'appuie sur Node.js et npm pour gérer les dépendances JavaScript. L'utilisation d'Expo CLI simplifie considérablement la configura-

tion de l'environnement de développement mobile en encapsulant la complexité des outils natifs.

Listing 5 – Installation du frontend React Native

```
# Installation d'Expo CLI globalement
npm install -g @expo/cli

# Installation des d pendances du projet
npm install

# Configuration des variables d'environnement
cp .env.example .env

# Lancement en mode d veloppement
npx expo start

# Lancement sur un simulateur iOS (macOS uniquement)
npx expo run:ios

# Lancement sur un   mulateur   Android
npx expo run:android
```

La configuration des variables d'environnement permet d'adapter l'application aux différents environnements de développement, test et production sans modifier le code source. Cette approche facilite le déploiement et améliore la sécurité en évitant le stockage de credentials dans le repository. Le fichier .env.example sert de template qui guide les développeurs dans la configuration de leur environnement local.

7.2 Extraits de code significatifs

Cette section présente des extraits de code représentatifs qui illustrent les patterns architecturaux et les bonnes pratiques implémentées dans le projet. Ces exemples démontrent la qualité technique du développement et facilitent la compréhension de l'architecture globale du système.

7.2.1 Authentification JWT côté serveur

L'implémentation de l'authentification JWT illustre l'approche sécurisée adoptée pour protéger les endpoints de l'API. Ce code démontre la validation des credentials, la génération de tokens et la gestion structurée des erreurs qui améliore l'expérience développeur.

Listing 6 – Endpoint d'authentification avec JWT

```
from flask import Blueprint, request, jsonify
```

```
from flask_jwt_extended import create_access_token
from werkzeug.security import check_password_hash
from app.models import User

auth_bp = Blueprint('auth', __name__)

@auth_bp.route('/login', methods=['POST'])
def login():
    """Authentification utilisateur et g n ration du token JWT"""
    try:
        # Validation des donn es d'entr e avec messages explicites
        data = request.get_json()
        if not data or not data.get('email') or not
            data.get('password'):
            return jsonify({
                "status": "error",
                "message": "Email et mot de passe requis",
                "errors": {"auth": "Donn es manquantes"}
            }), 400

        # Recherche de l'utilisateur en base avec gestion de la casse
        user = User.query.filter_by(email=data['email'].lower()).first()

        # V rification s curis e des credentials
        if not user or not check_password_hash(user.password_hash,
            data['password']):
            return jsonify({
                "status": "error",
                "message": "Identifiants incorrects",
                "errors": {"auth": "Identifiants invalides"}
            }), 401

        # G n ration du token JWT avec claims personnalis s
        access_token = create_access_token(
            identity=user.id,
            additional_claims={
                "username": user.username,
                "email": user.email,
                "is_admin": user.is_admin
            }
        )

        # R ponse structur e avec informations utilisateur
        return jsonify({
            "status": "success",
```

```
            "message": "Connexion r ussie",
            "data": {
                "access_token": access_token,
                "user": user.to_dict()
            }
        }), 200

    except Exception as e:
        # Logging de l'erreur pour le d bogage
        app.logger.error(f"Erreur lors de la connexion: {str(e)}")
        return jsonify({
            "status": "error",
            "message": "Erreur lors de la connexion",
            "errors": {"server": "Erreur interne"}
        }), 500
```

Ce code illustre plusieurs bonnes pratiques importantes. La validation des entrées s'effectue dès le début de la fonction pour éviter les traitements inutiles. La recherche utilisateur normalise l'email en minuscules pour éviter les problèmes de casse. La gestion d'erreurs distingue clairement les erreurs client des erreurs serveur tout en protégeant les informations sensibles.

7.2.2 Composant React Native pour l'enregistrement de course

Ce composant illustre l'intégration avec les APIs natives du smartphone et la gestion de l'état complexe nécessaire pour l'enregistrement en temps réel des données de course. L'implémentation démontre les bonnes pratiques React Native pour les applications de fitness.

Listing 7 – Composant d'enregistrement de course

```
import React, { useState, useEffect, useRef } from 'react';
import { View, Text, TouchableOpacity, Alert } from 'react-native';
import * as Location from 'expo-location';
import { useAuth } from '../contexts/AuthContext';
import { runAPI } from '../services/api';

const RunRecorder = () => {
  //  tat  du composant pour g rer l'enregistrement
  const [isRecording, setIsRecording] = useState(false);
  const [startTime, setStartTime] = useState(null);
  const [distance, setDistance] = useState(0);
  const [routeData, setRouteData] = useState([]);
  const [duration, setDuration] = useState(0);
  const [currentSpeed, setCurrentSpeed] = useState(0);
```

```
15
16  // R f rence pour le subscription GPS (nettoyage automatique)
17  const locationSubscription = useRef(null);
18
19  const { user, token } = useAuth();
20
21  // Effet pour mettre    jour la dur e pendant l'enregistrement
22  useEffect(() => {
23    let interval;
24    if (isRecording && startTime) {
25      interval = setInterval(() => {
26        setDuration(Math.floor((Date.now() - startTime) / 1000));
27      }, 1000);
28    }
29    return () => clearInterval(interval);
30  }, [isRecording, startTime]);
31
32  // Nettoyage automatique lors du d montage du composant
33  useEffect(() => {
34    return () => {
35      if (locationSubscription.current) {
36        locationSubscription.current.remove();
37      }
38    };
39  }, []);
40
41  // Fonction pour d marrer l'enregistrement avec gestion d'erreurs
       robuste
42  const startRecording = async () => {
43    try {
44      // Demander les permissions de g olocalisation avec message
           explicite
45      const { status } = await
           Location.requestForegroundPermissionsAsync();
46      if (status !== 'granted') {
47        Alert.alert(
48          'Permission requise',
49          'L\'acc s    la g olocalisation est n cessaire pour
               enregistrer vos courses.'
50        );
51        return;
52      }
53
54      // V rifier la disponibilit  du GPS
55      const enabled = await Location.hasServicesEnabledAsync();
```

```
56        if (!enabled) {
57          Alert.alert(
58            'GPS d sactiv ',
59            'Veuillez activer la g olocalisation dans les param tres.'
60          );
61          return;
62        }
63
64        // Initialiser l'enregistrement avec  tat  coh rent
65        setIsRecording(true);
66        setStartTime(Date.now());
67        setDistance(0);
68        setRouteData([]);
69        setDuration(0);
70        setCurrentSpeed(0);
71
72        // Configuration optimis e pour le suivi de course
73        locationSubscription.current = await Location.watchPositionAsync(
74          {
75            accuracy: Location.Accuracy.High,
76            timeInterval: 1000,  // Mise    jour chaque seconde
77            distanceInterval: 1, // Sensibilit  au m tre
78          },
79          (location) => {
80            // Validation de la qualit  des donn es GPS
81            if (location.coords.accuracy > 50) {
82              // Ignorer les positions peu pr cises
83              return;
84            }
85
86            // Cr er un nouveau point GPS avec m tadonn es
87            const newPoint = {
88              latitude: location.coords.latitude,
89              longitude: location.coords.longitude,
90              timestamp: location.timestamp,
91              accuracy: location.coords.accuracy,
92              speed: location.coords.speed || 0
93            };
94
95            setRouteData(prevData => {
96              const newData = [...prevData, newPoint];
97
98              // Calculer la distance incr mentale si on a au moins 2
                    points
99              if (newData.length > 1) {
```

```
100               const lastPoint = newData[newData.length - 2];
101               const distanceToAdd = calculateDistance(lastPoint,
                      newPoint);
102
103               // Filtrer les distances aberrantes (plus de 100m en 1
                      seconde)
104               if (distanceToAdd < 100) {
105                 setDistance(prevDistance => prevDistance +
                        distanceToAdd);
106               }
107             }
108
109             return newData;
110           });
111
112           // Mettre    jour la vitesse instantan e
113           setCurrentSpeed(location.coords.speed * 3.6 || 0); //
                  Conversion m/s vers km/h
114         }
115       );
116
117     } catch (error) {
118       Alert.alert('Erreur', 'Impossible de d marrer
             l\'enregistrement');
119       console.error('Erreur d marrage course:', error);
120
121       // R initialiser l' tat  en cas d'erreur
122       setIsRecording(false);
123     }
124   };
125
126   // Fonction pour arr ter et sauvegarder la course
127   const stopRecording = async () => {
128     try {
129       // Arr ter le suivi GPS imm diatement
130       if (locationSubscription.current) {
131         locationSubscription.current.remove();
132         locationSubscription.current = null;
133       }
134
135       setIsRecording(false);
136       const endTime = Date.now();
137
138       // Validation des donn es avant sauvegarde
139       if (duration < 30) {
```

```
140         Alert.alert(
141           'Course trop courte',
142           'La course doit durer au moins 30 secondes pour  tre
                 enregistr e.'
143         );
144         return;
145       }
146
147       if (distance < 50) {
148         Alert.alert(
149           'Distance insuffisante',
150           'La distance parcourue doit  tre  d\'au moins 50 m tres.'
151         );
152         return;
153       }
154
155       // Pr parer les donn es avec calculs de performance
156       const avgSpeed = distance > 0 ? distance / duration : 0;
157       const maxSpeed = Math.max(...routeData.map(point => point.speed
             || 0));
158
159       const runData = {
160         start_time: new Date(startTime).toISOString(),
161         end_time: new Date(endTime).toISOString(),
162         duration,
163         distance: Math.round(distance),
164         route_data: routeData,
165         avg_speed: avgSpeed,
166         max_speed: maxSpeed * 3.6, // Conversion en km/h
167         calories: Math.round(distance * 0.06 * (user.weight || 70) /
               70) // Estimation bas e sur le poids
168       };
169
170       // Sauvegarder avec indicateur de progression
171       const result = await runAPI.createRun(runData, token);
172
173       if (result.success) {
174         Alert.alert(
175           'Succ s ',
176           `Course enregistr e!\nDistance:
                 ${(distance/1000).toFixed(2)} km\nDur e:
                 ${formatDuration(duration)}`
177         );
178
179         // R initialiser compl tement l' tat
```

```
180         resetState();
181       } else {
182         Alert.alert('Erreur', ' chec  de l\'enregistrement de la
              course');
183       }
184
185     } catch (error) {
186       Alert.alert('Erreur', 'Impossible de sauvegarder la course');
187       console.error('Erreur sauvegarde course:', error);
188     }
189   };
190
191   // Fonction utilitaire pour r initialiser l' tat
192   const resetState = () => {
193     setStartTime(null);
194     setDistance(0);
195     setRouteData([]);
196     setDuration(0);
197     setCurrentSpeed(0);
198   };
199
200   // Calcul de distance optimis  avec formule de Haversine
201   const calculateDistance = (point1, point2) => {
202     const R = 6371e3; // Rayon de la Terre en m tres
203     const  1  = point1.latitude * Math.PI/180;
204     const  2  = point2.latitude * Math.PI/180;
205     const      = (point2.latitude-point1.latitude) * Math.PI/180;
206     const      = (point2.longitude-point1.longitude) * Math.PI/180;
207
208     const a = Math.sin(    /2) * Math.sin(    /2) +
209               Math.cos( 1 ) * Math.cos( 2 ) *
210               Math.sin(    /2) * Math.sin(    /2);
211     const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
212
213     return R * c;
214   };
215
216   // Formatage de la dur e pour l'affichage
217   const formatDuration = (seconds) => {
218     const hours = Math.floor(seconds / 3600);
219     const minutes = Math.floor((seconds % 3600) / 60);
220     const secs = seconds % 60;
221
222     if (hours > 0) {
223       return `${hours}:${minutes.toString().padStart(2,
```

```
          '0')}:${secs.toString().padStart(2, '0')}`;
224     }
225     return `${minutes}:${secs.toString().padStart(2, '0')}`;
226   };
227
228   return (
229     <View style={styles.container}>
230       <Text style={styles.title}>Enregistrement de Course</Text>
231
232       <View style={styles.statsContainer}>
233         <View style={styles.statItem}>
234           <Text style={styles.statLabel}>Dur e </Text>
235           <Text
              style={styles.statValue}>{formatDuration(duration)}</Text>
236         </View>
237
238         <View style={styles.statItem}>
239           <Text style={styles.statLabel}>Distance</Text>
240           <Text style={styles.statValue}>{(distance / 1000).toFixed(2)}
              km</Text>
241         </View>
242
243         <View style={styles.statItem}>
244           <Text style={styles.statLabel}>Allure</Text>
245           <Text style={styles.statValue}>
246             {distance > 0 ? (duration / (distance / 1000) /
                60).toFixed(2) : '0.00'} min/km
247           </Text>
248         </View>
249
250         <View style={styles.statItem}>
251           <Text style={styles.statLabel}>Vitesse</Text>
252           <Text style={styles.statValue}>{currentSpeed.toFixed(1)}
              km/h</Text>
253         </View>
254       </View>
255
256       <TouchableOpacity
257         style={[styles.button, isRecording ? styles.stopButton :
              styles.startButton]}
258         onPress={isRecording ? stopRecording : startRecording}
259         disabled={false}
260       >
261         <Text style={styles.buttonText}>
262           {isRecording ? 'Arr ter la course' : 'D marrer la course'}
```

```
263           </Text>
264         </TouchableOpacity>
265       </View>
266     );
267 };
268
269 export default RunRecorder;
```

Ce composant démontre plusieurs aspects techniques importants. La gestion des permissions utilisateur suit les meilleures pratiques avec des messages explicites. Le suivi GPS utilise des paramètres optimisés pour la course à pied avec filtrage des données aberrantes. La gestion de l'état React utilise des hooks appropriés avec nettoyage automatique des ressources.

7.3 Métriques et indicateurs de performance

Le monitoring des performances constitue un aspect crucial pour maintenir la qualité de service et identifier proactivement les problèmes potentiels. Cette section présente les métriques clés surveillées et leurs seuils d'alerte qui guident les décisions d'optimisation.

7.3.1 Métriques backend

Les métriques serveur permettent de surveiller la santé globale de l'API et d'identifier les goulots d'étranglement avant qu'ils n'impactent l'expérience utilisateur. Cette approche préventive améliore la fiabilité du système.

Métrique	**Objectif**	**Seuil d'alerte**	**Seuil critique**
Temps de réponse API	< 200ms	500ms	1000ms
Taux d'erreur	$< 1\%$	2%	5%
Utilisation CPU	$< 70\%$	80%	90%
Utilisation mémoire	$< 80\%$	85%	95%
Connexions DB actives	< 50	75	100
Temps de requête DB	< 50ms	100ms	200ms
Débit de requêtes	Variable	> 1000/min	> 2000/min
Taille du cache Redis	< 500MB	750MB	1GB

Table 10 – Métriques de performance backend

Ces métriques sont collectées en temps réel via des agents de monitoring qui agrègent les données sur différentes fenêtres temporelles. Cette approche permet de détecter aussi bien les pics ponctuels que les dégradations progressives de performance. L'historique de ces métriques facilite l'analyse des tendances et la planification de la capacité future.

Les seuils d'alerte sont calibrés selon l'expérience opérationnelle et ajustés régulièrement selon l'évolution du système. Les alertes de niveau critique déclenchent des interventions immédiates tandis que les alertes préventives permettent de planifier les optimisations.

7.3.2 Métriques frontend mobile

Le monitoring de l'application mobile se concentre sur l'expérience utilisateur et la performance perçue, aspects critiques pour l'adoption et la rétention des utilisateurs dans l'écosystème mobile compétitif.

Métrique	**Objectif**	**Description et importance**
Temps de démarrage	< 3 secondes	Durée critique entre le lancement et l'écran principal utilisable
Fluidité d'animation	60 FPS	Maintien des 60 images par seconde pour une expérience native
Consommation batterie	$< 5\%$/heure	Impact sur l'autonomie pendant l'usage normal de l'application
Taux de crash	$< 0.5\%$	Pourcentage de sessions interrompues par un crash application
Précision GPS	< 5 mètres	Écart moyen de localisation par rapport à la position réelle
Latence réseau	< 300ms	Temps de réponse des requêtes API depuis l'application
Taille de l'app	< 50MB	Espace de stockage occupé après installation complète

TABLE 11 – Métriques de performance mobile

Ces métriques mobile nécessitent des outils spécialisés qui intègrent avec les plateformes natives pour collecter les données de performance réelle. La mesure s'effectue sur des appareils représentatifs de la base utilisateur pour assurer la pertinence des données collectées.

7.4 Procédures de déploiement

La documentation des procédures de déploiement garantit la reproductibilité des mises en production et réduit les risques d'erreurs lors des livraisons. Ces procédures évoluent

avec le projet pour intégrer les leçons apprises et les améliorations process.

7.4.1 Déploiement backend en production

Le déploiement du backend suit un processus automatisé qui minimise les interventions manuelles et les risques d'erreurs. Cette procédure inclut les vérifications de sécurité et les tests de non-régression obligatoires.

Listing 8 – Script de déploiement backend production

```
#!/bin/bash
# Script de d ploiement automatis  pour le backend Flask
# Fichier: deploy_backend.sh

set -e  # Arr t imm diat en cas d'erreur

echo "     D but du d ploiement backend Running App"
echo "Environnement: PRODUCTION"
echo "Date: $(date '+%Y-%m-%d %H:%M:%S')"

# V rification des pr requis syst me
echo "     V rification de l'environnement de production"
python --version || exit 1
mysql --version || exit 1
nginx -v || exit 1

# V rification des variables d'environnement critiques
if [[ -z "$DB_PASSWORD" || -z "$JWT_SECRET_KEY" ]]; then
    echo "    Variables d'environnement manquantes"
    exit 1
fi

# Sauvegarde compl te de la base de donn es
echo "     Sauvegarde de la base de donn es"
BACKUP_FILE="backup_$(date +%Y%m%d_%H%M%S).sql"
mysqldump -u $DB_USER -p$DB_PASSWORD running_app_db >
   backups/$BACKUP_FILE
echo "Sauvegarde cr   e: $BACKUP_FILE"

# Mise    jour du code source avec v rification
echo "     R cup ration du code source"
git fetch origin
CURRENT_COMMIT=$(git rev-parse HEAD)
git checkout main
git pull origin main
NEW_COMMIT=$(git rev-parse HEAD)
```

```
36
37 echo "Commit pr c dent: $CURRENT_COMMIT"
38 echo "Nouveau commit: $NEW_COMMIT"
39
40 # Installation des d pendances avec cache pip
41 echo "     Installation des d pendances"
42 pip install --upgrade pip
43 pip install -r requirements.txt --no-deps
44
45 # Ex cution des migrations avec v rification
46 echo "     Application des migrations de base de donn es"
47 flask db upgrade
48
49 # V rification de l'int grit  de la base apr s migration
50 echo "     V rification de l'int grit  de la base de donn es"
51 python scripts/verify_db_integrity.py || exit 1
52
53 # Tests de validation complets
54 echo "     Ex cution des tests de validation"
55 python -m pytest tests/ -v --tb=short || exit 1
56
57 # Tests d'int gration avec base de donn es r elle
58 echo "     Tests d'int gration"
59 python -m pytest tests/integration/ -v || exit 1
60
61 # Collecte des fichiers statiques
62 echo "     Collecte des fichiers statiques"
63 python manage.py collectstatic --noinput
64
65 # Red marrage gracieux des services
66 echo "     Red marrage des services"
67 sudo systemctl reload nginx
68 sudo systemctl restart running-app-backend
69 sudo systemctl restart running-app-worker  # Worker Celery si applicable
70
71 # Attente de la stabilisation des services
72 sleep 10
73
74 # V rifications de sant  post-d ploiement
75 echo "     V rifications de sant  de l'API"
76 for i in {1..5}; do
77     if curl -f -s http://localhost:5000/api/health > /dev/null; then
78         echo "    API op rationnelle (tentative $i)"
79         break
80     else
```

```
        echo "    Attente de l'API (tentative $i/5)"
        sleep 5
    fi
done

# Test des endpoints critiques
echo "     Test des endpoints critiques"
curl -f http://localhost:5000/api/auth/health || exit 1
curl -f http://localhost:5000/api/runs/health || exit 1
curl -f http://localhost:5000/api/proposed-runs/categories || exit 1

# Monitoring post-d ploiement
echo "     Initialisation du monitoring post-d ploiement"
python scripts/post_deploy_monitoring.py

echo "    D ploiement termin  avec succ s"
echo "     Commit d ploy  : $NEW_COMMIT"
echo "     Logs disponibles dans: /var/log/running-app/"
```

7.4.2 Publication sur les app stores

La publication de l'application mobile suit des processus spécifiques à chaque plateforme qui nécessitent une préparation minutieuse et des tests approfondis sur différents appareils et versions de système.

Pour l'App Store iOS, la procédure inclut plusieurs étapes critiques. La configuration des certificats de distribution nécessite une attention particulière aux dates d'expiration et aux profils de provisioning. La génération du build avec Xcode suit des paramètres spécifiques pour optimiser la taille et les performances de l'application finale.

La soumission via App Store Connect implique la préparation de métadonnées détaillées, screenshots sur différentes tailles d'écran et la configuration des informations de publication. Le processus de review d'Apple peut prendre de 24 heures à plusieurs jours selon la complexité de l'application et les éventuels problèmes détectés.

Pour Google Play Store, le processus utilise Android Studio pour générer l'APK ou l'AAB signé avec les clés de production. La configuration des métadonnées inclut les descriptions multilingues, les catégories appropriées et les informations de classification par âge.

Le déploiement progressif sur Google Play permet de limiter l'impact d'éventuels problèmes en déployant d'abord sur un pourcentage limité d'utilisateurs avant le déploiement complet. Cette approche graduelle améliore la qualité des mises à jour et permet de détecter les problèmes sur un échantillon réduit.

7.5 Tests et validation

La stratégie de tests garantit la qualité du code et la stabilité des fonctionnalités tout au long du cycle de développement. Cette approche multicouche couvre les tests unitaires, d'intégration et end-to-end pour assurer une couverture complète des cas d'usage.

7.5.1 Tests backend

Les tests backend valident la logique métier, les endpoints API et l'intégration avec la base de données. Ces tests utilisent pytest et des fixtures pour créer des environnements de test reproductibles et isolés.

Listing 9 – Suite de tests API complète

```python
import pytest
import json
from datetime import datetime, timedelta
from app import create_app, db
from app.models import User, Run

@pytest.fixture(scope='function')
def test_app():
    """Application Flask configur e pour les tests"""
    app = create_app('testing')
    with app.app_context():
        db.create_all()
        yield app
        db.session.remove()
        db.drop_all()

@pytest.fixture
def client(test_app):
    """Client de test Flask"""
    return test_app.test_client()

@pytest.fixture
def test_user(test_app):
    """Utilisateur de test en base de donn es"""
    user = User(
        username="testuser",
        email="test@example.com",
        first_name="Test",
        last_name="User",
        password_hash=generate_password_hash("Test123!")
    )
    db.session.add(user)
```

```
    db.session.commit()
    return user

@pytest.fixture
def auth_headers(client, test_user):
    """Headers d'authentification JWT valides"""
    response = client.post('/api/auth/login', json={
        "email": "test@example.com",
        "password": "Test123!"
    })

    assert response.status_code == 200
    token = response.get_json()['data']['access_token']
    return {"Authorization": f"Bearer {token}"}

@pytest.fixture
def sample_run_data():
    """Donn es de course valides pour les tests"""
    now = datetime.utcnow()
    return {
        "start_time": now.isoformat() + "Z",
        "end_time": (now + timedelta(minutes=30)).isoformat() + "Z",
        "duration": 1800,
        "distance": 5000,
        "avg_speed": 2.78,
        "max_speed": 3.5,
        "calories": 350,
        "route_data": [
            {"latitude": 48.856614, "longitude": 2.3522219,
               "timestamp": now.timestamp()},
            {"latitude": 48.857614, "longitude": 2.3532219,
               "timestamp": (now + timedelta(minutes=15)).timestamp()},
            {"latitude": 48.858614, "longitude": 2.3542219,
               "timestamp": (now + timedelta(minutes=30)).timestamp()}
        ]
    }

class TestAuthenticationAPI:
    """Tests de l'authentification et autorisation"""

    def test_successful_login(self, client, test_user):
        """Test de connexion r ussie"""
        response = client.post('/api/auth/login', json={
            "email": "test@example.com",
            "password": "Test123!"
```

```
        })

        assert response.status_code == 200
        data = response.get_json()
        assert data['status'] == 'success'
        assert 'access_token' in data['data']
        assert data['data']['user']['email'] == 'test@example.com'

    def test_invalid_credentials(self, client, test_user):
        """Test de connexion avec identifiants invalides"""
        response = client.post('/api/auth/login', json={
            "email": "test@example.com",
            "password": "WrongPassword"
        })

        assert response.status_code == 401
        data = response.get_json()
        assert data['status'] == 'error'
        assert 'access_token' not in data.get('data', {})

    def test_missing_credentials(self, client):
        """Test de connexion avec donn es manquantes"""
        response = client.post('/api/auth/login', json={
            "email": "test@example.com"
        })

        assert response.status_code == 400
        data = response.get_json()
        assert data['status'] == 'error'

class TestRunsAPI:
    """Tests des endpoints de gestion des courses"""

    def test_create_run_success(self, client, auth_headers,
       sample_run_data):
        """Test de cr ation d'une course valide"""
        response = client.post('/api/runs',
                               json=sample_run_data,
                               headers=auth_headers)

        assert response.status_code == 201
        data = response.get_json()
        assert data['status'] == 'success'
        assert data['data']['distance'] == 5000
        assert data['data']['duration'] == 1800
```

```
119
120     def test_create_run_invalid_data(self, client, auth_headers):
121         """Test de cr ation avec donn es invalides"""
122         invalid_data = {
123             "start_time": "invalid-date",
124             "duration": -100,  # Dur e n gative
125             "distance": "not-a-number"
126         }
127
128         response = client.post('/api/runs',
129                                json=invalid_data,
130                                headers=auth_headers)
131
132         assert response.status_code == 400
133         data = response.get_json()
134         assert data['status'] == 'error'
135
136     def test_get_user_runs(self, client, auth_headers, test_user):
137         """Test de r cup ration des courses utilisateur"""
138         # Cr er quelques courses de test
139         for i in range(3):
140             run = Run(
141                 user_id=test_user.id,
142                 start_time=datetime.utcnow() - timedelta(days=i),
143                 end_time=datetime.utcnow() - timedelta(days=i) +
                        timedelta(minutes=30),
144                 duration=1800,
145                 distance=5000 + i * 1000
146             )
147             db.session.add(run)
148         db.session.commit()
149
150         response = client.get('/api/runs', headers=auth_headers)
151
152         assert response.status_code == 200
153         data = response.get_json()
154         assert data['status'] == 'success'
155         assert len(data['data']['runs']) == 3
156
157     def test_get_run_statistics(self, client, auth_headers, test_user):
158         """Test des statistiques de course"""
159         response = client.get('/api/runs/stats', headers=auth_headers)
160
161         assert response.status_code == 200
162         data = response.get_json()
```

```
163         assert data['status'] == 'success'
164         assert 'total_runs' in data['data']
165         assert 'total_distance' in data['data']
166 
167     def test_unauthorized_access(self, client, sample_run_data):
168         """Test d'acc s non autoris """
169         response = client.post('/api/runs', json=sample_run_data)
170 
171         assert response.status_code == 401
172 
173 class TestProposedRunsAPI:
174     """Tests des courses propos es"""
175 
176     def test_get_proposed_runs_public(self, client):
177         """Test d'acc s public aux courses propos es"""
178         response = client.get('/api/proposed-runs')
179 
180         assert response.status_code == 200
181         data = response.get_json()
182         assert data['status'] == 'success'
183         assert 'runs' in data['data']
184         assert len(data['data']['runs']) > 0
185 
186     def test_get_proposed_runs_filtered(self, client):
187         """Test de filtrage des courses propos es"""
188         response =
               client.get('/api/proposed-runs?difficulty=beginner&type=endurance')
189 
190         assert response.status_code == 200
191         data = response.get_json()
192         assert data['status'] == 'success'
193 
194         # V rifier que le filtrage est appliqu
195         for run in data['data']['runs']:
196             assert run['difficulty'] == 'beginner'
197             assert run['type'] == 'endurance'
198 
199     def test_get_run_categories(self, client):
200         """Test de r cup ration des cat gories"""
201         response = client.get('/api/proposed-runs/categories')
202 
203         assert response.status_code == 200
204         data = response.get_json()
205         assert data['status'] == 'success'
206         assert 'difficulties' in data['data']
```

```
        assert 'types' in data['data']
        assert 'durations' in data['data']

class TestDataValidation:
    """Tests de validation des donn es"""

    def test_email_validation(self, client):
        """Test de validation des emails"""
        invalid_emails = [
            "invalid-email",
            "@domain.com",
            "user@",
            "user space@domain.com"
        ]

        for email in invalid_emails:
            response = client.post('/api/auth/register', json={
                "username": "testuser",
                "email": email,
                "password": "Test123!",
                "first_name": "Test",
                "last_name": "User"
            })

            assert response.status_code == 400

    def test_password_strength(self, client):
        """Test de validation de la force du mot de passe"""
        weak_passwords = [
            "123456",      # Trop simple
            "password",    # Pas de chiffres
            "Pass1",       # Trop court
            "PASSWORD123"  # Pas de minuscules
        ]

        for password in weak_passwords:
            response = client.post('/api/auth/register', json={
                "username": "testuser",
                "email": "test@example.com",
                "password": password,
                "first_name": "Test",
                "last_name": "User"
            })

            assert response.status_code == 400
```

```python

class TestPerformance:
    """Tests de performance et charge"""

    def test_api_response_time(self, client, auth_headers):
        """Test des temps de r ponse API"""
        import time

        start_time = time.time()
        response = client.get('/api/runs', headers=auth_headers)
        end_time = time.time()

        response_time = end_time - start_time
        assert response_time < 1.0  # Moins d'une seconde
        assert response.status_code == 200

    def test_bulk_data_handling(self, client, auth_headers, test_user):
        """Test de gestion de volumes importants de donn es"""
        # Cr er un grand nombre de courses
        runs = []
        for i in range(100):
            run = Run(
                user_id=test_user.id,
                start_time=datetime.utcnow() - timedelta(days=i),
                end_time=datetime.utcnow() - timedelta(days=i) +
                    timedelta(minutes=30),
                duration=1800,
                distance=5000
            )
            runs.append(run)

        db.session.add_all(runs)
        db.session.commit()

        # Tester la pagination
        response = client.get('/api/runs?page=1&per_page=20',
            headers=auth_headers)

        assert response.status_code == 200
        data = response.get_json()
        assert len(data['data']['runs']) == 20

# Configuration pytest
def pytest_configure(config):
    """Configuration globale des tests"""
```

```
    import warnings
    warnings.filterwarnings("ignore", category=DeprecationWarning)

# Ex cution des tests avec coverage
if __name__ == "__main__":
    pytest.main([
        "--verbose",
        "--tb=short",
        "--cov=app",
        "--cov-report=html",
        "--cov-report=term-missing"
    ])
```

Cette suite de tests couvre les aspects critiques de l'application avec une attention particulière à la sécurité, la validation des données et les performances. L'utilisation de fixtures pytest facilite la réutilisation du code de test et garantit l'isolation entre les tests.

7.5.2 Tests frontend mobile

Les tests de l'application React Native combinent tests unitaires des composants, tests d'intégration et tests end-to-end pour valider l'expérience utilisateur complète.

Listing 10 – Tests React Native avec Jest et React Native Testing Library

```
import React from 'react';
import { render, fireEvent, waitFor } from
   '@testing-library/react-native';
import { Alert } from 'react-native';
import * as Location from 'expo-location';

// Mocks des d pendances externes
jest.mock('expo-location');
jest.mock('../services/api');
jest.mock('../contexts/AuthContext');

import RunRecorder from '../components/RunRecorder';
import { runAPI } from '../services/api';
import { useAuth } from '../contexts/AuthContext';

describe('RunRecorder Component', () => {
  // Configuration des mocks
  beforeEach(() => {
    jest.clearAllMocks();

    // Mock du contexte d'authentification
    useAuth.mockReturnValue({
```

```
      user: { id: 1, weight: 70 },
      token: 'mock-jwt-token'
    });

    // Mock des permissions GPS
    Location.requestForegroundPermissionsAsync.mockResolvedValue({
      status: 'granted'
    });

    Location.hasServicesEnabledAsync.mockResolvedValue(true);
  });

  test('renders initial state correctly', () => {
    const { getByText, getByTestId } = render(<RunRecorder />);

    expect(getByText('Enregistrement de Course')).toBeTruthy();
    expect(getByText('D marrer la course')).toBeTruthy();
    expect(getByText('0:00')).toBeTruthy(); // Dur e initiale
    expect(getByText('0.00 km')).toBeTruthy(); // Distance initiale
  });

  test('starts recording when start button is pressed', async () => {
    // Mock du suivi GPS
    const mockLocationSubscription = {
      remove: jest.fn()
    };

    Location.watchPositionAsync.mockResolvedValue(mockLocationSubscription);

    const { getByText } = render(<RunRecorder />);
    const startButton = getByText('D marrer la course');

    fireEvent.press(startButton);

    await waitFor(() => {
      expect(Location.requestForegroundPermissionsAsync).toHaveBeenCalled();
      expect(Location.watchPositionAsync).toHaveBeenCalled();
      expect(getByText('Arr ter la course')).toBeTruthy();
    });
  });

  test('handles GPS permission denial', async () => {
    // Mock du refus de permission
    Location.requestForegroundPermissionsAsync.mockResolvedValue({
      status: 'denied'
```

```
    });

    const alertSpy = jest.spyOn(Alert, 'alert');

    const { getByText } = render(<RunRecorder />);
    fireEvent.press(getByText('D marrer la course'));

    await waitFor(() => {
      expect(alertSpy).toHaveBeenCalledWith(
        'Permission requise',
        'L\'acc s    la g olocalisation est n cessaire pour
           enregistrer vos courses.'
      );
    });
  });

  test('calculates distance correctly during recording', async () => {
    const mockLocationSubscription = {
      remove: jest.fn()
    };

    // Mock de la fonction de callback GPS
    let locationCallback;
    Location.watchPositionAsync.mockImplementation((options, callback)
       => {
      locationCallback = callback;
      return Promise.resolve(mockLocationSubscription);
    });

    const { getByText } = render(<RunRecorder />);
    fireEvent.press(getByText('D marrer la course'));

    await waitFor(() => {
      expect(locationCallback).toBeDefined();
    });

    // Simuler des points GPS
    const point1 = {
      coords: {
        latitude: 48.856614,
        longitude: 2.3522219,
        accuracy: 5,
        speed: 2.5
      },
      timestamp: Date.now()
```

```
    };

    const point2 = {
      coords: {
        latitude: 48.857614,
        longitude: 2.3532219,
        accuracy: 5,
        speed: 2.8
      },
      timestamp: Date.now() + 1000
    };

    // Envoyer les points GPS
    locationCallback(point1);
    locationCallback(point2);

    // V rifier que la distance est calcul e
    await waitFor(() => {
      const distanceText = getByText(/\d+\.\d+ km/);
      expect(distanceText).toBeTruthy();
    });
  });

  test('saves run data when stopped', async () => {
    // Mock de l'API de sauvegarde
    runAPI.createRun.mockResolvedValue({ success: true });

    const mockLocationSubscription = {
      remove: jest.fn()
    };

    Location.watchPositionAsync.mockResolvedValue(mockLocationSubscription);

    const { getByText } = render(<RunRecorder />);

    // D marrer l'enregistrement
    fireEvent.press(getByText('D marrer la course'));

    await waitFor(() => {
      expect(getByText('Arr ter la course')).toBeTruthy();
    });

    // Attendre suffisamment pour avoir une course valide (>30s)
    jest.advanceTimersByTime(35000);

```

```
155     // Arr ter l'enregistrement
156     fireEvent.press(getByText('Arr ter la course'));
157
158     await waitFor(() => {
159       expect(runAPI.createRun).toHaveBeenCalledWith(
160         expect.objectContaining({
161           duration: expect.any(Number),
162           distance: expect.any(Number),
163           route_data: expect.any(Array)
164         }),
165         'mock-jwt-token'
166       );
167     });
168   });
169
170   test('prevents saving short runs', async () => {
171     const alertSpy = jest.spyOn(Alert, 'alert');
172
173     const { getByText } = render(<RunRecorder />);
174
175     // D marrer puis arr ter imm diatement
176     fireEvent.press(getByText('D marrer la course'));
177
178     await waitFor(() => {
179       expect(getByText('Arr ter la course')).toBeTruthy();
180     });
181
182     // Avancer de seulement 20 secondes (insuffisant)
183     jest.advanceTimersByTime(20000);
184
185     fireEvent.press(getByText('Arr ter la course'));
186
187     await waitFor(() => {
188       expect(alertSpy).toHaveBeenCalledWith(
189         'Course trop courte',
190         'La course doit durer au moins 30 secondes pour  tre
              enregistr e.'
191       );
192     });
193
194     expect(runAPI.createRun).not.toHaveBeenCalled();
195   });
196
197   test('handles API errors gracefully', async () => {
198     // Mock d'une erreur API
```

```
199     runAPI.createRun.mockRejectedValue(new Error('Network error'));
200
201     const alertSpy = jest.spyOn(Alert, 'alert');
202     const consoleSpy = jest.spyOn(console,
          'error').mockImplementation();
203
204     const { getByText } = render(<RunRecorder />);
205
206     fireEvent.press(getByText('D marrer la course'));
207
208     await waitFor(() => {
209       expect(getByText('Arr ter la course')).toBeTruthy();
210     });
211
212     jest.advanceTimersByTime(35000);
213     fireEvent.press(getByText('Arr ter la course'));
214
215     await waitFor(() => {
216       expect(alertSpy).toHaveBeenCalledWith(
217         'Erreur',
218         'Impossible de sauvegarder la course'
219       );
220       expect(consoleSpy).toHaveBeenCalledWith(
221         'Erreur sauvegarde course:',
222         expect.any(Error)
223       );
224     });
225   });
226 });
227
228 // Tests d'int gration
229 describe('RunRecorder Integration Tests', () => {
230   test('complete recording workflow', async () => {
231     // Test complet du workflow d'enregistrement
232     const { getByText } = render(<RunRecorder />);
233
234     // 1. D marrer l'enregistrement
235     fireEvent.press(getByText('D marrer la course'));
236
237     // 2. Simuler une course compl te
238     await waitFor(() => {
239       expect(getByText('Arr ter la course')).toBeTruthy();
240     });
241
242     // 3. Avancer le temps et simuler des donn es GPS
```

```
243     jest.advanceTimersByTime(60000); // 1 minute
244
245     // 4. Arr ter et sauvegarder
246     fireEvent.press(getByText('Arr ter la course'));
247
248     // 5. V rifier le retour    l' tat  initial
249     await waitFor(() => {
250       expect(getByText('D marrer la course')).toBeTruthy();
251     });
252   });
253 });
```

7.6 Glossaire technique

Ce glossaire définit les termes techniques spécifiques utilisés dans le projet pour faciliter la compréhension et maintenir une terminologie cohérente entre les membres de l'équipe et les parties prenantes.

7.7 Ressources et documentation externe

Cette section référence les ressources externes essentielles pour comprendre les technologies utilisées et poursuivre le développement du projet. Ces références constituent une bibliothèque technique pour l'équipe de développement.

7.7.1 Documentation des frameworks principaux

— **Flask Documentation Officielle** : `https://flask.palletsprojects.com/` - Documentation complète du micro-framework web Python incluant les guides d'installation, tutoriels et références API

— **React Native Documentation** : `https://reactnative.dev/` - Guide complet pour le développement d'applications mobiles cross-platform avec React Native

— **Expo Documentation** : `https://docs.expo.dev/` - Plateforme de développement React Native qui simplifie la configuration et le déploiement

— **SQLAlchemy Documentation** : `https://docs.sqlalchemy.org/` - ORM Python utilisé pour l'abstraction de la base de données

7.7.2 Standards et spécifications

— **RFC 7519 - JSON Web Token** : `https://tools.ietf.org/html/rfc7519` - Spécification officielle du standard JWT pour l'authentification

— **OpenAPI Specification** : `https://spec.openapis.org/oas/v3.1.0` - Standard pour documenter les APIs REST

— **W3C Geolocation API** : `https://www.w3.org/TR/geolocation-API/` - Spécification de l'API de géolocalisation web

7.7.3 Outils de développement

— **Jest Testing Framework** : `https://jestjs.io/` - Framework de test JavaScript utilisé pour les tests unitaires React Native

— **Pytest Documentation** : `https://docs.pytest.org/` - Framework de test Python pour les tests backend

— **Docker Documentation** : `https://docs.docker.com/` - Plateforme de conteneurisation pour le déploiement

Documentation technique complète

Cette documentation technique fournit une base solide pour comprendre, maintenir et faire évoluer l'application Running App. Les exemples de code, procédures de déploiement et suites de tests constituent des références pratiques pour le développement futur et l'optimisation continue du système. La structure modulaire facilite la mise à jour de sections spécifiques sans impacter l'ensemble de la documentation.

Maintenance de la documentation

Cette documentation doit être maintenue à jour avec l'évolution du projet. Chaque modification significative de l'architecture, des APIs ou des procédures doit être reflétée dans les sections correspondantes. Une revue trimestrielle de la documentation est recommandée pour assurer sa pertinence et son exactitude.

8 Références et Documentation

Références

[1] Documentation officielle Flask. *Flask Web Development Framework*. `https://flask.palletsprojects.com/`

[2] Documentation React Native. *React Native - Learn once, write anywhere*. `https://reactnative.dev/`

[3] Documentation MySQL. *MySQL Database Management System*. `https://dev.mysql.com/doc/`

[4] RFC 7519 - JSON Web Token (JWT). *Internet Engineering Task Force*. `https://tools.ietf.org/html/rfc7519`

[5] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

Terme	Définition
JWT	JSON Web Token - Standard de sécurité RFC 7519 pour l'authentification stateless qui encapsule les informations d'identité dans un token signé cryptographiquement
REST	Representational State Transfer - Style d'architecture pour les APIs web qui utilise les verbes HTTP standard et représente les ressources par des URLs
ORM	Object-Relational Mapping - Technique de programmation qui permet de mapper les objets d'un langage orienté objet avec les tables d'une base de données relationnelle
GPS	Global Positioning System - Système de géolocalisation par satellite qui fournit des coordonnées précises de latitude et longitude
Endpoint	Point d'accès spécifique d'une API REST identifié par une URL unique et associé à une méthode HTTP
Blueprint	Module Flask qui organise les routes par domaine fonctionnel et facilite la modularité du code
Middleware	Composant logiciel qui intercepte et traite les requêtes HTTP avant qu'elles n'atteignent le contrôleur final
Hook	Fonction React qui permet d'utiliser l'état et les effets de bord dans les composants fonctionnels
Hot Reload	Fonctionnalité de développement qui recharge automatiquement l'application lors des modifications du code source
CI/CD	Continuous Integration/Continuous Deployment - Pratiques DevOps d'intégration et déploiement continus qui automatisent les tests et les mises en production
Fixture	Données ou état prédéfinis utilisés dans les tests pour assurer la reproductibilité et l'isolation
Mock	Objet simulé utilisé dans les tests pour remplacer des dépendances externes et contrôler leur comportement
Haversine	Formule mathématique pour calculer la distance orthodromique entre deux points sur une sphère à partir de leurs coordonnées
CORS	Cross-Origin Resource Sharing - Mécanisme de sécurité web qui permet aux ressources d'un domaine d'être accessibles depuis un autre domaine

TABLE 12 – Glossaire des termes techniques