### **Université Quisqueya**

# A1 : L’IDENTIFICATION DU PROJET

.

## A.1. Identification du document

|  |  |
| --- | --- |
| **Projet :** | FITNESS-APP |
| **Version du projet :** | 1.0 |
| **Version du document :** | 1 |
| **Sécurité du document :** | Confidentiel |
| **Date de création :** | 31/06/2024 |
| **Par :** | LAINÉ STÉPHANE |

## A.2 HISTORIQUE DES CHANGEMENTS

## **A.2. Historique des changements**

| **Qui?** | **Quand?** | **Quoi?** |
| --- | --- | --- |
| **LAINE STEPHANE** | 25 juillet 2024 | Création du document initial |
| **LAINE STEPHANE** | 1er août 2024 | Ajout de la section sur les configurations matérielles |
| **LAINE STEPHANE** | 5 août 2024 | choix de l’architecture réseau |
| **LAINE STEPHANE** | 6 août 2024 | ajout du choix de l’environnement dev |
| **LAINE STEPHANE** | 10 août 2024 | Finalisation du document et correction des fautes |

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

# Table des Matières

#### Introduction 1.1. Objectif 4 1.2. Portée

#### Spécifications Techniques 2.1. Environnement de Développement 7 2.1.1. Outils et Langages 2.2. Spécifications Matérielles 2.2.1. Serveurs 2.2.2. Infrastructure Réseau 2.2.3. Sécurité Réseau 2.2.4. Répartition de Charge et Haute Disponibilité 2.2.5. Surveillance et Maintenance 2.3. Configuration Logicielle 2.3.1. Backend 2.3.2. Frontend 2.3.3. CI/CD

#### Détails d'Implémentation 3.1. Structure du Projet 12 3.1.1. Frontend 3.1.2. Backend 3.2. Gestion de la Sécurité 3.2.1. Authentification 3.2.2. Autorisation 3.2.3. Sécurisation des API 3.3. Gestion des Erreurs 3.4. Scalabilité

#### Tests et Validation 4.1. Tests Unitaires 14 4.2. Tests d'Intégration 4.3. Tests End-to-End 4.4. Tests de Performance 4.5. Gestion des Bugs

#### Déploiement 5.1. Pré-requis 15 5.2. Docker 5.3. Push 5.4. Deploy

## 

## 

## 

## 

## **1. Introduction**

### **1.1. Objectif**

Ce document technique a pour but de fournir une description complète et détaillée des aspects techniques liés à la conception, au développement, et à la mise en œuvre du projet "Application de suivi FItness". Il s'adresse aux développeurs, ingénieurs, et autres parties prenantes techniques.

### **1.2. Portée**

Ce document couvre les spécifications matérielles et logicielles, les détails d'implémentation, les tests, les procédures de déploiement, et la documentation API.

## **2. Spécifications Techniques**

### **2.1. Environnement de Développement**

#### 2.1.1. Outils et Langages

### 

#### IDE/Éditeurs

* Visual Studio Code (VS Code) :  
  Visual Studio Code est un éditeur de code source développé par Microsoft, populaire pour son extensibilité, sa rapidité et sa prise en charge de nombreux langages de programmation. Il offre une multitude d'extensions pour améliorer la productivité, comme les intégrations avec Git, des plugins pour les frameworks JavaScript, et des outils pour le développement mobile avec Flutter.
* Android Studio (pour Flutter) :  
  Android Studio est l'IDE officiel de Google pour le développement d'applications Android. Il est particulièrement adapté pour le développement mobile avec Flutter, fournissant un support complet pour l'émulation, le débogage et le profilage des applications Flutter. Android Studio intègre également des outils comme le Dart Analyzer pour vérifier la qualité du code Dart.

#### Langages

* Dart (pour Flutter) :  
  Dart est un langage de programmation orienté objet développé par Google, utilisé principalement pour le développement d'applications front-end sur des plateformes mobiles et web avec Flutter. Dart est apprécié pour sa syntaxe simple et sa capacité à compiler en code natif, ce qui permet de créer des applications performantes.
* JavaScript/TypeScript (pour React.js, Node.js) :
  + JavaScript : Langage de programmation de base pour le développement web, JavaScript est utilisé pour la création de pages web interactives. Dans ce projet, JavaScript est utilisé à la fois côté client (pour React.js) et côté serveur (pour Node.js).
  + TypeScript : Un sur-ensemble de JavaScript qui ajoute des types statiques, facilitant la détection des erreurs lors de la compilation. TypeScript est utilisé pour des projets de plus grande envergure, où la gestion des types permet de maintenir un code propre et robuste.

#### Frameworks

* Frontend Mobile : Flutter :  
  Flutter est un framework open-source développé par Google pour créer des applications nativement compilées pour mobile, web, et bureau à partir d'une seule base de code. Flutter utilise Dart comme langage de programmation, et permet de créer des interfaces utilisateur attrayantes et performantes grâce à son moteur de rendu intégré.
* Frontend Web : React.js ou Angular :
  + React.js : Une bibliothèque JavaScript open-source utilisée pour construire des interfaces utilisateur en composants. Développé par Facebook, React permet de créer des applications web dynamiques et réactives grâce à son approche déclarative et sa gestion efficace de l'état via le Virtual DOM.
  + Angular : Un framework JavaScript open-source développé par Google, Angular est une solution complète pour le développement de SPAs (Single Page Applications). Il offre des outils robustes pour la gestion de l'architecture des applications, le routing, et la gestion de l'état.
* Backend : Node.js avec Express.js :  
  Node.js est une plateforme qui permet d'exécuter du code JavaScript côté serveur. Utilisé pour développer des applications web évolutives, il est connu pour sa capacité à gérer un grand nombre de connexions simultanées avec un faible temps de réponse.
  + Express.js : Un framework web pour Node.js, Express simplifie la création d'APIs robustes et performantes en fournissant une structure minimaliste pour gérer les routes, les middlewares et les requêtes HTTP.

#### Bases de Données

* Firestore :  
  Firestore est une base de données NoSQL en temps réel, offerte par Firebase de Google. Elle est utilisée pour stocker et synchroniser des données entre les utilisateurs et les appareils en temps réel. Firestore est particulièrement adaptée pour les applications nécessitant une synchronisation instantanée des données.
* MongoDB :  
  MongoDB est une base de données NoSQL orientée documents, qui stocke les données au format BSON (une version binaire de JSON). MongoDB est utilisé pour sa flexibilité, sa scalabilité horizontale et sa capacité à gérer de grandes quantités de données non structurées.

#### Contrôle de Version

* Git, hébergé sur GitHub :  
  Git est un système de contrôle de version distribué utilisé pour suivre les modifications du code source tout au long du cycle de développement. Il permet de collaborer efficacement, de gérer différentes versions du projet, et de suivre l'historique des modifications. Le code source du projet est hébergé sur GitHub, une plateforme de partage et de collaboration de code, qui offre également des outils de gestion de projet tels que les issues et les pull requests.

#### CI/CD (Intégration Continue / Déploiement Continu)

* GitHub Actions :  
  GitHub Actions est une fonctionnalité intégrée de GitHub qui permet de créer des workflows d'intégration continue et de déploiement continu directement dans le dépôt GitHub. Il offre une automatisation flexible pour les tests, les builds, et les déploiements en réponse à divers événements dans le dépôt.
* CircleCI :  
  CircleCI est une plateforme CI/CD hébergée qui permet aux développeurs d'automatiser les tests et les déploiements de leurs applications. CircleCI s'intègre facilement avec GitHub et offre des pipelines rapides et configurables pour un déploiement efficace.

#### Tests

* Jest (tests unitaires) :  
  Jest est un framework de test JavaScript conçu par Facebook, principalement utilisé pour les tests unitaires et les tests de composants. Jest est facile à configurer et offre des fonctionnalités avancées comme les mocks, les tests asynchrones, et la surveillance de la couverture du code.
* capacité à déboguer les tests en temps réel.

### **2.2. Spécifications Matérielles**

#### **2.2.1. Serveurs**

* **Type** :  
  Les serveurs virtuels privés (VPS) ou les services managés tels qu'AWS EC2 et Google Cloud Compute Engine sont choisis pour leur capacité à fournir une infrastructure scalable, permettant un ajustement des ressources selon la demande de l'application.
* **CPU** :  
  Les serveurs doivent disposer d'au moins 2 vCPUs (virtual Central Processing Units) pour gérer efficacement les requêtes utilisateur, les processus de traitement de données, et les opérations backend sans compromettre les performances.
* **RAM** :  
  Une mémoire vive (RAM) de 8 GB est recommandée pour assurer une exécution fluide des services backend, incluant la gestion des connexions simultanées, le traitement des requêtes API, et le fonctionnement des bases de données en mémoire.
* **Stockage** :  
  Un disque de stockage SSD (Solid-State Drive) d'une capacité minimale de 100 GB est nécessaire. Le stockage SSD est choisi pour sa rapidité et sa fiabilité, garantissant des temps de lecture/écriture optimaux pour les opérations de base de données et le stockage des fichiers statiques.
* **Système d'Exploitation** :  
  Linux, et plus précisément Ubuntu 20.04 LTS, est recommandé pour son support à long terme, sa stabilité, et sa large adoption dans les environnements de production. Ubuntu offre une grande compatibilité avec les outils de développement modernes et un écosystème de sécurité robuste, essentiel pour le déploiement d'applications web.

#### **2.2.2. Infrastructure Réseau**

L'infrastructure réseau est un élément fondamental pour assurer la connectivité, la sécurité et la performance de l'application de suivi de fitness. Une architecture réseau robuste et scalable est conçue pour répondre aux besoins de l'application, en garantissant une haute disponibilité et une sécurité renforcée.

#### 2.2.1. Topologie Réseau

L'infrastructure réseau est structurée selon une architecture multi-tier, séparant les différentes couches fonctionnelles de l'application pour optimiser la gestion des ressources et renforcer la sécurité.

* Couches de l'Infrastructure :
  + Frontend : Hébergement des fichiers statiques (HTML, CSS, JavaScript) sur des serveurs de contenu, accessibles via HTTPS pour garantir la sécurité des communications.
  + Backend : Serveurs d'application responsables du traitement des requêtes, de la logique métier, et de l'interaction avec les bases de données. Cette couche est protégée par des pare-feu pour restreindre l'accès uniquement aux points d'entrée autorisés.
  + Base de Données : Les bases de données sont déployées sur des serveurs dédiés ou via des services managés, avec des communications chiffrées entre le backend et la couche de données pour protéger les informations sensibles.

#### 2.2.2. Sécurité Réseau

La sécurité est intégrée à chaque niveau de l'infrastructure pour protéger les données et assurer l'intégrité du système.

* Chiffrement : L'ensemble des communications réseau utilise HTTPS avec TLS pour sécuriser les échanges de données entre les utilisateurs et les serveurs. Les certificats SSL/TLS sont gérés de manière automatisée par des solutions comme Let's Encrypt.
* Pare-feu et Contrôles d'Accès : Des pare-feu sont configurés pour filtrer le trafic, avec des groupes de sécurité définissant les règles d'accès aux ressources critiques comme les serveurs d'application et les bases de données.
* Accès Administratif : L'accès aux serveurs de production est restreint via un VPN et des Bastion Hosts, qui servent de passerelles sécurisées pour les connexions SSH, limitant ainsi les points d'entrée potentiels pour les attaques.

#### 2.2.3. Répartition de Charge et Haute Disponibilité

Pour garantir la disponibilité et la performance, des solutions de répartition de charge et de scalabilité automatique sont mises en place.

* Load Balancing : Utilisation de load balancers pour distribuer le trafic entre plusieurs instances de backend, assurant une réponse rapide aux requêtes des utilisateurs et une tolérance aux pannes.
* Auto-scaling : Les serveurs sont configurés pour auto-scaling, avec des instances ajoutées ou supprimées automatiquement en fonction des charges de travail, assurant ainsi une performance constante.
* Réplication des Données : Les bases de données sont répliquées sur plusieurs serveurs pour assurer la redondance et la tolérance aux pannes, avec des mécanismes de failover pour minimiser les interruptions en cas de défaillance.

#### 2.2.4. Surveillance et Maintenance

La surveillance continue de l'infrastructure est essentielle pour maintenir la performance et détecter les problèmes potentiels.

* Monitoring : Des outils comme New Relic ou Datadog surveillent la performance du réseau et des applications en temps réel, permettant une détection précoce des problèmes et une optimisation continue.
* Logs et Audits : Tous les événements réseau sont journalisés pour permettre des audits et un suivi en cas d'incidents. Les logs sont analysés en continu pour identifier les anomalies.
* Alertes Proactives : Des systèmes d'alerte sont configurés pour notifier l'équipe en temps réel en cas de problème, permettant une réponse rapide et réduisant ainsi les temps d'arrêt.

### **2.3. Configuration Logicielle**

#### **2.3.1. Backend**

* Node.js : Version 16.x
* Express.js : Version 4.x
* Firebase : Intégré via SDK
* MongoDB : Version 4.x ou plus, hébergé sur MongoDB Atlas ou sur une instance dédiée

#### **2.3.2. Frontend**

* Flutter : Version 3.x
* React.js : Version 18.x ou Angular 14.x
* Redux (ou NgRx pour Angular) : pour la gestion de l'état
* Firebase SDK : Pour l'authentification et la gestion en temps réel

#### **2.3.3. CI/CD**

* GitHub Actions : Pour l'intégration continue
* Docker : Pour la création d'images de conteneurs
* Jenkins : Pour le déploiement continu (si utilisé)

## **3. Détails d'Implémentation**

### **3.1. Structure du Projet**

#### **3.1.1. Frontend**

* **/lib (Flutter)** : Contient les fichiers source pour les widgets, les routes, les services.
* **/src (React.js)** : Contient les composants, les services API, les reducers (pour Redux).
* **/assets** : Images, fichiers de styles, etc.

#### **3.1.2. Backend**

* **/src/controllers** : Contient les contrôleurs pour gérer les requêtes API.
* **/src/models** : Contient les modèles de données (Mongoose pour MongoDB).
* **/src/routes** : Définit les routes API.
* **/src/services** : Contient la logique métier et les interactions avec les bases de données.

### **3.2. Gestion de la Sécurité**

#### **3.2.1. Authentification**

* **Firebase Authentication** : Utilisé pour gérer l'authentification des utilisateurs via OAuth (Google, Facebook, etc.).

#### **3.2.2. Autorisation**

* **JWT (JSON Web Tokens)** : Utilisé pour gérer l'autorisation des requêtes API. Les tokens sont générés lors de la connexion et utilisés pour vérifier l'accès aux ressources protégées.

#### **3.2.3. Sécurisation des API**

* **HTTPS** : Toutes les communications entre le frontend et le backend sont sécurisées via HTTPS.
* **Rate Limiting** : Implémenté sur les API pour prévenir les abus et les attaques par déni de service (DDoS).

### **3.3. Gestion des Erreurs**

* **Middleware d'Erreur** : Express.js utilise un middleware global pour capturer et gérer les erreurs, renvoyant des réponses structurées avec des codes HTTP appropriés.
* Logs : Les erreurs sont enregistrées via un service de logging (comme Winston) pour faciliter le suivi et la résolution des problèmes.

### **3.4. Scalabilité**

* Microservices : Le backend peut être modulé en microservices si la charge utilisateur devient trop importante pour une seule application monolithique.
* Cache : Utilisation de Redis pour le caching des requêtes lourdes et la gestion des sessions.
* Auto-scaling : Configuré au niveau des serveurs cloud pour ajuster les ressources en fonction de la demande.

## **4. Tests et Validation**

### **4.1. Tests Unitaires**

* **Jest** : Utilisé pour tester les composants JavaScript/TypeScript côté frontend et les fonctions du backend.
* **Flutter Test** : Utilisé pour tester les widgets et les composants dans l'application mobile.
* **Supertest** :  
  Supertest est utilisé pour vérifier l'intégrité et la fonctionnalité des endpoints de l'API. En simulant des requêtes HTTP, cet outil permet de s'assurer que les endpoints interagissent correctement avec la base de données et d'autres services, tels que l'authentification et la gestion des sessions. Les tests d'intégration avec Supertest garantissent que les différentes parties du backend fonctionnent ensemble de manière harmonieuse, détectant les erreurs qui pourraient survenir lorsque les modules individuels sont combinés.

### **4.3. Tests End-to-End**

* **Cypress** :  
  Cypress est l'outil de choix pour réaliser des tests end-to-end (E2E) sur l'application. Ces tests couvrent l'intégralité du flux utilisateur, depuis la connexion à l'application, l'enregistrement d'une séance d'entraînement, jusqu'à la visualisation des résultats. Cypress permet de simuler les interactions réelles des utilisateurs avec l'application, tout en vérifiant que toutes les fonctionnalités fonctionnent correctement à travers les différents navigateurs et environnements. Les tests E2E avec Cypress assurent que l'application offre une expérience utilisateur cohérente et sans faille.

### **4.4. Tests de Performance**

* **Apache JMeter** :  
  Apache JMeter est utilisé pour évaluer la performance de l'API en simulant des charges élevées. Cet outil permet de tester la capacité de l'API à gérer un grand nombre de requêtes simultanées, en mesurant les temps de réponse et l'endurance sous pression. Grâce à JMeter, il est possible d'identifier les goulots d'étranglement et de s'assurer que l'infrastructure backend est suffisamment robuste pour supporter une montée en charge, garantissant ainsi une performance optimale même en conditions de haute demande.

### **4.5. Gestion des Bugs**

* **Issue Tracker** :  
  Le suivi et la gestion des bugs signalés sont effectués via GitHub Issues. Cet outil intégré à GitHub permet de créer, organiser et suivre les problèmes rencontrés tout au long du cycle de vie du projet. Chaque bug ou amélioration est enregistré sous forme d'issue, avec la possibilité de prioriser, de commenter, et d'assigner les tâches aux membres de l'équipe. GitHub Issues offre une vue claire de l'avancement des résolutions, facilitant ainsi la collaboration et la transparence dans le processus de développement.
* Reporting :  
  La mise en place de rapports automatisés des bugs est assurée par des outils comme Sentry. Sentry est utilisé pour capturer les erreurs en production et fournir des rapports détaillés, incluant les traces de pile, les contextes utilisateur, et les conditions du système au moment de l'erreur. Cela permet de détecter et de corriger rapidement les problèmes critiques, minimisant ainsi l'impact sur les utilisateurs finaux. Les notifications en temps réel et les tableaux de bord de Sentry aident l'équipe de développement à surveiller la santé de l'application en continu et à réagir de manière proactive aux incidents.

## **5. Déploiement**

## **5.1. Pré-requis**

Exécution des scripts de build pour préparer le backend en vue du déploiement. Cette étape inclut la compilation du code, la génération des fichiers nécessaires, et la vérification que toutes les dépendances sont correctement installées. L'objectif est de produire une version stable et prête à être empaquetée dans une image Docker.

* Docker :  
  Construction de l'image Docker pour le backend en utilisant un fichier Dockerfile. Cette image contiendra tout ce qui est nécessaire pour exécuter l'application, y compris le code source, les bibliothèques, et les dépendances système. Le processus de construction de l'image est optimisé pour créer une image légère et sécurisée.
* Push :  
  Poussée de l'image Docker vers un registre de conteneurs, tel que GitHub Container Registry ou Docker Hub. Ce registre sert de dépôt centralisé où les images sont stockées et accessibles pour le déploiement sur différents environnements.
* Deploy :  
  Déploiement de l'image Docker sur les serveurs via un orchestrateur de conteneurs comme Kubernetes. Kubernetes gère le déploiement, la mise à l'échelle, et la gestion des conteneurs en production, assurant ainsi la haute disponibilité et la résilience du backend.

### 5.2.2. Frontend

* Build :  
  Exécution des scripts de build pour compiler et optimiser le frontend. Cette étape inclut la minification des fichiers CSS et JavaScript, la génération des fichiers HTML, et la préparation des assets pour une livraison rapide et efficace via le réseau.
* Deploy :  
  Déploiement des fichiers statiques résultants sur un CDN (Content Delivery Network) ou un service de hosting comme Netlify ou AWS S3. Ces services distribuent les fichiers à travers un réseau mondial de serveurs, réduisant ainsi les temps de chargement et améliorant l'expérience utilisateur.

### 5.3. Monitoring et Maintenance

* Monitoring :  
  Mise en place d'un système de surveillance de la performance du système en utilisant des services comme New Relic ou Datadog. Ces outils fournissent une vue détaillée des métriques critiques (temps de réponse, utilisation des ressources, etc.) et déclenchent des alertes en cas de problème. Cela permet de réagir rapidement aux incidents et d'assurer la stabilité du système.
* Mises à Jour :  
  Les mises à jour du système suivent un processus de déploiement continu (CI/CD) pour minimiser les interruptions de service. Chaque mise à jour est d'abord testée dans un environnement de staging avant d'être progressivement déployée en production, assurant ainsi une transition fluide et sans risque pour les utilisateurs finaux.

## **6. Documentation API**

### **6.1. Description Générale**

L'API RESTful fournit des endpoints pour interagir avec les données du système, y compris les utilisateurs, les séances d'entraînement, et les objectifs. Cette API est conçue pour faciliter la gestion des routines de fitness des utilisateurs, y compris la création, la mise à jour, et le suivi des objectifs de fitness, ainsi que l'accès aux données d'exercices.

L'API utilise ExerciseDB, une base de données d'exercices, pour enrichir l'expérience des utilisateurs en leur fournissant un large éventail d'exercices avec des descriptions détaillées, des instructions et des images. ExerciseDB est intégré à l'API pour permettre la recherche d'exercices spécifiques et la création de routines d'entraînement personnalisées basées sur les besoins et les préférences des utilisateurs.

### **6.2. Endpoints Principaux**

* **POST /users/register**
  + **Description** : Créer un nouveau compte utilisateur.
  + **Payload** : { "email": "string", "password": "string" }
  + **Réponses** :
    - 201 Created : Utilisateur créé avec succès.
    - 400 Bad Request : Données de requête mal formées.
* **POST /users/login**
  + **Description** : Authentifier un utilisateur.
  + **Payload** : { "email": "string", "password": "string" }
  + **Réponses** :
    - 200 OK : Authentification réussie, renvoie un token JWT.
    - 401 Unauthorized : Échec de l'authentification.
* **GET /workouts**
  + **Description** : Récupérer les séances d'entraînement de l'utilisateur authentifié.
  + **Headers** : Authorization: Bearer <JWT>
  + **Réponses** :
    - 200 OK : Renvoie une liste de séances d'entraînement.
    - 401 Unauthorized : Token JWT manquant ou invalide.

### **6.3. Gestion des Erreurs API**

L'API renvoie des erreurs structurées au format JSON avec un code HTTP et un message descriptif.

* **Exemple** :  
  json  
  Copy code  
  {
* "error": {
* "code": 400,
* "message": "Invalid request payload"
* }
* }

## **7. Synthèse**

Le projet de suivi de fitness que nous avons conçu et développé repose sur une architecture robuste et moderne, utilisant des technologies de pointe pour offrir une expérience utilisateur fluide et performante. De la sélection rigoureuse des outils et langages, à l'intégration continue et au déploiement automatisé, chaque étape a été pensée pour garantir un système sécurisé, évolutif et facilement maintenable.

Le backend, construit avec Node.js et Express.js, et encapsulé dans des conteneurs Docker, assure une gestion efficace des données et des performances optimales grâce à l'orchestration via Kubernetes. Le frontend, développé avec Flutter pour le mobile et React.js ou Angular pour le web, propose une interface intuitive et réactive, permettant aux utilisateurs de suivre leurs objectifs de fitness avec facilité.

L'intégration de ExerciseDB enrichit l'application en offrant un vaste répertoire d'exercices, ce qui permet aux utilisateurs de créer des routines personnalisées basées sur des informations fiables. La surveillance continue, assurée par des outils tels que New Relic et Datadog, ainsi que la mise en œuvre de pratiques de mise à jour via CI/CD, garantissent que le système reste performant et sécurisé tout au long de son cycle de vie.

En résumé, ce document technique sert de guide détaillé pour comprendre l'architecture, les processus et les technologies qui soutiennent ce projet de suivi de fitness. Il offre une base solide pour le développement futur, l'amélioration continue et l'expansion potentielle des fonctionnalités, en s'assurant que l'application répondra aux besoins des utilisateurs tout en restant à la pointe de l'innovation technologique.