|  |
| --- |
| API RESTful Runscape |

Table des matières

[1 Analyse préliminaire 4](#_Toc4070077)

[*1.1* Introduction 4](#_Toc4070078)

[1.2 Objectifs 4](#_Toc4070079)

[*1.3* Planification initiale 6](#_Toc4070080)

[2 Analyse 6](#_Toc4070081)

[2.1 Définition de l’audience 6](#_Toc4070082)

[2.2 Analyse Concurrentielle 7](#_Toc4070083)

[2.3 Analyse des architectures 8](#_Toc4070084)

[2.4 Étude de faisabilité 9](#_Toc4070085)

[2.5 Choix du matériel physique 10](#_Toc4070086)

[2.6 Utilisateurs 11](#_Toc4070087)

[2.7 Authentification 11](#_Toc4070088)

[2.8 Activités sportives 11](#_Toc4070089)

[2.9 Gestion des retours d'erreurs 11](#_Toc4070090)

[2.10 Interface web d’administration 12](#_Toc4070091)

[2.11 Modèle conceptuel des données 12](#_Toc4070092)

[2.12 Stratégie de test 13](#_Toc4070093)

[2.13 Risques techniques 13](#_Toc4070094)

[2.14 Budget initial 14](#_Toc4070095)

[3 Conception 14](#_Toc4070096)

[3.1 Analyse de l’environnement 14](#_Toc4070097)

[3.2 Utilisateurs 14](#_Toc4070098)

[3.3 Authentification 14](#_Toc4070099)

[3.4 Gestion d'erreurs 15](#_Toc4070100)

[3.5 Activités sportives 16](#_Toc4070101)

[3.6 Ressources de l'API 17](#_Toc4070102)

[3.7 Endpoints de l’API 17](#_Toc4070103)

[3.8 Version des logiciels 18](#_Toc4070104)

[3.9 Choix du moteur de base de données 18](#_Toc4070105)

[3.10 Modèle logique des données 18](#_Toc4070106)

[3.11 Interface web d'administration 21](#_Toc4070107)

[3.12 Diagrammes de flux 22](#_Toc4070108)

[3.13 Cas d’utilisation 24](#_Toc4070109)

[4 Réalisation 28](#_Toc4070110)

[4.1 Configuration serveur 28](#_Toc4070111)

[4.2 Dossier de réalisation 30](#_Toc4070112)

[4.3 Description des tests effectués 30](#_Toc4070113)

[*4.4* Erreurs restantes 32](#_Toc4070114)

[*4.5* Liste des documents fournis 32](#_Toc4070115)

[5 Conclusions 32](#_Toc4070116)

[6 Annexes 33](#_Toc4070117)

[*6.1* Résumé du rapport du TPI / version succincte de la documentation 33](#_Toc4070118)

[*6.2* Sources – Bibliographie 33](#_Toc4070119)

[*6.3* Journal de travail 33](#_Toc4070120)

[*6.4* Manuel d'Installation 33](#_Toc4070121)

[*6.5* Manuel d'Utilisation 33](#_Toc4070122)

[*6.6* Archives du projet 33](#_Toc4070123)

# Analyse préliminaire

## Introduction

Le projet consiste à réaliser une API Web permettant à une personne de se connecter, se déconnecter, gérer ses activités physiques (ajouter de nouvelles activités, les consulter, les modifier et les supprimer). Cette API ne possède pas d’interface graphique permettant à un utilisateur de communiquer plus simplement avec celle-ci.

Un deuxième projet sera entrepris par la suite pour permettre une interaction par interface graphique avec cette API, ce deuxième projet consistera à utiliser l’API dans une application mobile, des facteurs de perte de synchronisation seront à prendre en compte lors de la conception de cette API.

Plusieurs modules ont été réalisé, dans le cadre du CPNV, et me permettrons de mener à bien ce projet :

* ICT-100 (Distinguer, préparé et évaluer des données)
* ICT-104 (Implémenter un modèle de données)
* ICT-105 (Traiter une base de données SQL)
* ICT-306 (Réaliser un petit projet informatique)
* MA-07 (Linux)

## Objectifs

### Mettre en œuvre une base de données pour les activités sportives

L’API doit être en mesure de manipuler des données modélisant des activités sportives, mettant à disposition de l’utilisateur les informations suivantes :

* La date
* L’heure de départ
* Le type d’activités (vélo, natation, course à pied)
* La durée de l’effort
* La distance parcourue
* Le parcours (une représentation graphique sera attendue sur la partie cliente)
* Les dénivelés positifs et négatifs
* La vitesse moyenne

### Web Service en JSON

Le back-end devra avoir la capacité de dialoguer avec un composant client qui effectuera les actions suivantes :

* Enregistrer, modifier, supprimer une activité
* Consulter et analyser les activités sportives sur une période donnée (moyenne, nombre de kilomètres parcourus par sport)
* Importer des données en provenance de GPS (format .gpx considéré en priorité)
* S’interfacer[[1]](#footnote-1) avec au moins un acteur majeur de gestion d’activité tels que Garmin Connect, Tom tom

### Sécurisation du back-end

Le back-end devra implémenter les sécurités suivantes :

* N’être accessible que via le protocole HTTPS (self-signed certificate)
* Aucun mot de passe ne doit être présent dans la base de données ni dans les logs applicatifs[[2]](#footnote-2)
* Le code applicatif ne doit pas avoir la capacité de supprimer (ni DELETE, ni DROP) des données

### Authentification des utilisateurs

Le back-end devra implémenter les mécanismes d’authentification suivantes :

* Soit pouvoir se connecter à l’aide d’un service « 2-step verification »
* Soit s’appuyer sur les APIs de Facebook

## Planification initiale



# Analyse

## Définition de l’audience

Les personnes ciblées par l’application sont les personnes à la recherche d’une application leur permettant d’enregistrer les activités sportives qu’ils réalisent. L’application permet à n’importe quelle personne disposant d’un téléphone portable Android[[3]](#footnote-3). L’audience est donc composée de personne faisant du sport ou voulant commencer à en faire.

## Analyse Concurrentielle

### Définition des concurrents

Les concurrents qui ont été retenus pour cette analyse proposent tous des applications mobile Android et iOS, deux des trois concurrents possèdes plus de 10 millions de téléchargements de leur application sur le Play Store de Google, le troisième concurrent possède 1 million de téléchargement. Les applications comparées sont, par conséquent des applications utilisés par énormément de personnes et reconnues pour leur maitrise du métier.

Les éléments analysés ne prennent pas en compte l’interface graphique de l’application, il ne s’agit ici que de fonctionnalités brutes proposée par l’application.

Les 3 applications analysées sont :

* Garmin Connect
* Tom Tom Sports
* Adidas Runtastic

### Analyse des fonctionnalités proposées

Le tableau ci-dessous contient une liste non-exhaustive des fonctionnalités proposées par les applications analysées. Le but est de définir quelles fonctionnalités semblent indispensables et quelles fonctionnalités semblent superflues.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Garmin Connect | TomTom Sports | Adidas Runtastic |
| Changements unité (miles, mètres, kg, lb) | X | X | X |
| Distance parcourue | X | X | X |
| Ajout manuel d’activités | X |  | X |
| Ajout d’un type d’activité | X |  |  |
| Calories brulées pendant l’effort | X |  | X |
| Personnalisation de la longueur de foulée | X |  |  |
| Record personnels | X |  | X |
| Interactions sociales intégrées | X |  | X |
| Abonnement mensuel permettant d’ajouter des fonctionnalités |  |  | X |
| Visualisation du parcours effectué | X |  | X |
| Altitude et dénivelé de l’activité | X |  | X |
| Défis ou objectifs | X | X | X |
| Gestion d’équipements (chaussures, …) | X |  | X |
| Connection à des services tierces | X | X | X |
| Délégation de la connexion (ex : par Google) |  |  | X |

## Analyse des architectures

Plusieurs architectures d’API Web existent, les 2 principales étant le protocole SOAP et l’architecture REST.

|  |  |
| --- | --- |
|  | Avantages |
| REST | Supporte d’autres format que l’XML (JSON par exemple) |
| Le serveur n’enregistre pas l’état du client, ce qui permet de distribuer les charges sur plusieurs serveurs |
| Plus facile à intégrer que le SOAP |
| Plus utilisé que SOAP dans le contexte des Web Services |
| SOAP | Peut amener à une meilleure sécurité selon l’implémentation (signature par certificat XML par exemple) |
| Standardisé |
|  |
|  |

|  |  |
| --- | --- |
|  | Désavantages |
| REST | Architecture étant très stricte |
|  |
|  |
| SOAP | Ne supporte pas le JSON |
| Plus difficile à implémenter |
|  |

L’avantage d’utilisé l’architecture REST et de s’imposer un ensemble de contraintes pour faciliter l’évolutivité du système dans le cas où l’API venait à être distribuée sur plusieurs serveur par exemple.

Le SOAP n’étant plus autant utilisé que le REST et ne permettant pas l’’utilisation du JSON, j’ai choisi l’architecture RESTful pour mon API.

## Étude de faisabilité

### Faisabilité système

Ayant déjà réalisé l’installation d’un serveur NodeJS et d’une base de données MySQL sur un Raspberry Pi en dehors de ce projet, aucun problème ne devrait survenir lors d’une nouvelle installation.

### Faisabilité logicielle

Des modules NodeJS seront utilisés lors de la réalisation du Web Service, voici une liste de plusieurs modules qui seront sûrement utilisés :

* fs (accès au système de fichier)
* http (serveur HTTP)
* https (serveur HTTPS)
* router (gestion des routes URLs)
* crypto (hash et chiffrement léger)
* bcrypt (hash et chiffrement important)
* url (gestion des URLs)
* querystring (gestion des querystring dans les URLs)
* validator (gestion des chaînes de charactères)
* util (utilitaires)
* xml2js (conversion des XML en JSON)

Ces modules sont, soit des modules de bases, soit des modules très connus dans l’écosystème NodeJS, il est donc très facile de se documenter sur ces modules. Aucun problème de faisabilité de devrait être rencontré lors de la réalisation du Web Service.

## Choix du matériel physique

Les environnements de déploiement évalués sont les suivants :

* Serveur physique
* Machine virtuelle
* Raspberry Pi
* Solution Cloud

### Serveur physique

|  |  |
| --- | --- |
| Avantages | Désavantages |
| Composants dédiés | Coûts élevé |
| Évolutivité du serveur | Extension de la puissance de calcul |
| Performances | Administration et entretien nécessaire |
| Administration et entretien nécessaire |  |

### Machine virtuelle

|  |  |
| --- | --- |
| Avantages | Désavantages |
| Possibilité de faire des snapshots | Composants virtuels |
| Réplication des machines très rapide | Performances |
| Extension de la puissance de calcul simple | Administration et entretien nécessaire |
| Administration et entretien nécessaire |  |

### Raspberry Pi

|  |  |
| --- | --- |
| Avantages | Désavantages |
| Peu onéreux | Puissance de calcul limitée |
| Composants dédiés | Stockage limité |
| Déjà disponible | Évolutivité du serveur limité |
| Administration et entretien nécessaire | Administration et entretien nécessaire |
| Consommation électrique |  |

### Solution Cloud

|  |  |
| --- | --- |
| Avantages | Désavantages |
| Aucun entretien serveur-s nécessaire | Gestion des serveur déléguées à une entité tierce |
| Extension de la puissance de calcul simple | Coûts |
| Administration et entretien nécessaire |  |

L'hébergement choisi et le Raspberry Pi, car celui-ci est déjà disponible et permet d'administrer soi-même le serveur, ce qui permet d'avoir une très grande liberté lors du développement et de la mise en place de l'architecture. Le serveur physique n'a pas été choisi car les coûts de celui-ci auraient été trop élevés.

## Utilisateurs

L'application a besoin d'utilisateurs car elle doit avoir la capacité de déterminer à quel sportif appartiennent quelles activités. Les utilisateurs permettront aussi de déterminer quels sportifs sont administrateurs de l'application et quels sportifs n'ont pas les droits d'administrations.

## Authentification

Une authentification est nécessaire car il faut être en mesure de déterminer à quel sportif appartiennent les activités qu'il a réalisé.

L'authentification permet aussi de définir quels sportifs sont administrateur et lesquels sont de simple utilisateur de l'application.

Une authentification permettrait aussi, par la suite, d'ajouter des fonctionnalités de réseau social, comme par exemple, le partage des activités d'un sportif, l'ajout de contacts ou une fonctionnalité "se souvenir de mes identifiants".

## Activités sportives

Le cahier des charges demande l'intégration d'une manipulation d'activités sportives au sein de l'application. Cette fonctionnalité permettra à un sportif d'enregistrer, de consulter, de modifier et de supprimer les activités qu'il a réalisé sur l'application.

Seul un sportif connecté peut gérer les activités liées à son compte, aucun autre sportif ne doit avoir la possibilité de créer, lire, modifier ou supprimer les activités d'un autre.

Le sportif aura la possibilité d'importer soit un GPX[[4]](#footnote-4), soit entrer manuellement son activité sportive. L'intérêt pour le sportif d'importer un GPX est de ne pas avoir à entrer tous les champs manuellement, est donc de perdre du temps.

## Gestion des retours d'erreurs

Un sportif ne doit pas avoir la possibilité d'être informé sur le matériel et les logiciels utilisés pour faire fonctionner l'application. Les erreurs retournées au sportif doivent être compréhensible.

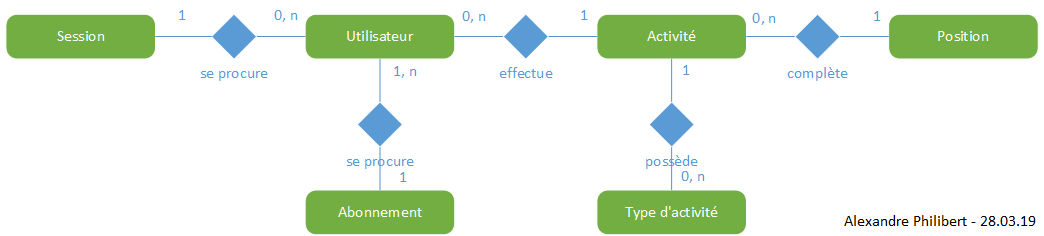
La gestion des erreurs permettra aussi de limiter les informations que peuvent collecter des personnes malintentionnées.

## Interface web d’administration

### Connexion

L’interface web d’administration doit permettre à un administrateur de l’application de gérer la liste des sports ainsi que les sportifs, cette gestion doit être accessible uniquement par un administrateur, c’est pourquoi une connexion est nécessaire.

## Modèle conceptuel des données



### Utilisateur

Un utilisateur est unique, il peut s’abonner plusieurs fois et peut réaliser plusieurs activités sportives (Un sportif créer une nouvelle activité à chaque entrainement).

### Abonnement

Plusieurs abonnements peuvent être procurés par les utilisateurs, un abonnement à une durée limitée et appartient à un seul utilisateur (Un abonnement est créé lorsqu’un utilisateur renouvelle son abonnement).

### Activité

Une activité est unique, un seul utilisateur peut effectuer une activité. Une activité possède un type d’activité qui peut être possédée par d’autres activités. (Un sportif peut consulter toutes les activités de type ‘course à pied’.

### Type d’activité

Un type d’activité est possédé par les activités. Un type d’activité est unique. (Un sportif veut pouvoir choisir le type d’activité parmi une liste de possibilité).

### Identifiants unique naturel

|  |  |
| --- | --- |
| Entité | Champs |
| Utilisateur | Email |
| Activité | Heure de début  Heure de fin  Fichier GPX |
| Type d’activité | Nom |

## Stratégie de test

### Étendue des tests

Les tests seront effectués sur tous les endpoints de l’API. Les tests seront réalisés à l’aide du logiciel Postman.

### Testeurs

* Alexandre Philibert : [alexandre.philibert@cpnv.ch](mailto:alexandre.philibert@cpnv.ch)

### Type de tests

* Unitaire
* Intégration
* Fonctionnels

### Liste des tests

* Vérification du fonctionnement des use cases (décrit ci-dessus)
* Vérification du fonctionnement des scénarios (décrit ci-dessus)

### Données de test à prévoir

Aucunes données de test ne sont prévues

## Budget initial

Aucun budget n’est alloué à la réalisation de ce projet.

# Conception

## Analyse de l’environnement

J’ai choisi de travailler avec le logiciel Notepad++, j’utilise l’auto complétion proposée par Notepad++ et les plugins pour d’autres fonctionnalités comme le FTP.

Lorsque je rencontre des problèmes, je peux utiliser la console de développement de Chrome pour accéder à un débuggeur NodeJS.

## Utilisateurs

Les utilisateurs s'authentifierons par e-mail et mot de passe, l'email permet d'avoir un identifiant unique. Il ne sera pas possible de créer plusieurs comptes avec la même adresse e-mail.

## Authentification

### Authentification par nom et mot de passe

L'authentification par nom et mot de passe et la manière la plus simple d'authentifier une personne sur un web service. Cette méthode à l'avantage de ne pas avoir à stocker de session sur le serveur. Il y a néanmoins un coût important à chaque requête car le serveur doit vérifier la correspondance des informations d'authentification avec celles présentes dans la base de données.

### Authentification par OAuth

L'OAuth est un protocole d'authentification et d'identification très utilisé. Le principal avantage de l'OAuth est la délégation de l'authentification à un parti tierce. L'OAuth permet d'authentifier un utilisateur sur une application avec des identifiants d'une autre application, sans que les identifiants ne quittent l'application à laquelle ils appartiennent.

Cette fonctionnalité n'est pas nécessaire à la réalisation de ce projet, l'implémentation de l'OAuth ajouterais des fonctionnalités non demandées au projet, il ne sera donc pas utilisé dans celui-ci.

### Authentification par tokens

Une implémentation d'authentification stateless ne permet pas de stocker des sessions sur les serveurs. Cette approche permet de très facilement étendre le nombre de nœuds de calculs sans subir des pertes de performances et des problèmes de synchronisation des sessions.

L'une des seules méthodes permettant une authentification stateless est l'authentification par tokens. Un standard a été créer pour gérer ces tokens, les JSON Web Tokens (JWT).

Un des désavantages d'une authentification entièrement stateless est qu'il n'est pas possible de révoquer un token d'authentification.

Dans le cas où une révocation des tokens doit être implémenté, l'authentification ne pourra plus rester stateless, un stockage des tokens sera nécessaire.

Les JWT permettent d'attribuer des scopes de droits à un utilisateur, ce qui permet de limiter l'accès d'un utilisateur à la modification de ses propres activités uniquement.

Il serait préférable d’implémenter une authentification par token standard, la caractéristique stateless de l’API sera donc perdue, mais l’authentification sur un navigateur web sera grandement améliorer. Le JWT n’étant pas un choix correct pour l’authentification par navigateur web.

Une authentification par token de session sera aussi implémentée. Ce token prendra la forme d'une chaîne de caractères aléatoires ayant une longueur de 32 caractères.

## Gestion d'erreurs

### Structure de retour

La structure de retour de l’api prendra la forme suivante :

|  |
| --- |
| {  "data": [  [Object object]  ],  "error": {    }  } |
|  |

Cette structure permet de très facilement voire si un problème est survenu lors du traitement de la requête. Elle possède aussi l’avantage de ne pas changer de forme selon l’état de la requête (dans tous les cas les objets ‘data’ et ‘error’ seront présents).

Avec cette structure de retour il est aussi possible de retourner des informations tout en ayant soulevé une erreur sur le serveur. Ce qui permet de tout de même continuer l’exécution de l’application.

Cette structure permet de n’avoir qu’une seule erreur retournée par le serveur, il est tout à fait envisageable, dans la suite du projet, de modifier cette structure pour permettre le retour de plusieurs erreurs simultanément.

### Gestionnaire de structure de retour

Pour assurer l’unicité des structures de retour, il serait préférable d’ajouter un gestionnaire de structure de retour. Il sera donc possible, avec ce gestionnaire, de vérifier pour chaque retour utilisateur, la validité du format de retour.

### Classes d’erreurs

Plusieurs classes d’erreurs vont être ajouté à l’API, celle-ci permettent de préciser quel type d’erreur a été soulevé et quel traitement doit être effectué. Les classes d’erreurs présentes dans l’API sont les suivantes :

|  |  |
| --- | --- |
| MysqlError | Cette classe d’erreur doit être utilisée lorsqu’un problème d’exécution se produit sur la base de données. Elle permet de clairement différencier les erreurs de base de données et de ne pas les remontées à l’utilisateur final. |
| UnauthorizedError | Cette classe d’erreur doit être utilisé lorsqu’un utilisateur essaie d’accéder à une ressource protégée dont il ne dispose pas les droits nécessaires. |
| MissingFieldError | Cette classe d’erreur doit être utilisé dès le moment ou un champ nécessaire à l’exécution d’une fonction n’est pas rempli |
| InvalidFormatError | Cette classe d’erreur doit être utilisé lorsqu’un champ ne contient pas le bon format de données, par exemple un email sans ‘@’, ou encore une chaîne de charactères à la place d’un entier |

## Activités sportives

La gestion des activités sportives nécessite la création d'une nouvelle entité dans la base de données car un utilisateur doit pouvoir enregistrer de multiples activités sportives. Une nouvelle entité permet de simplement différencier les ressources que l'application peut gérer.

L'utilisateur doit avoir la possibilité d'importer un fichier GPX et le sauvegarder en tant qu'activité sportive. Le fichier sera parser sur le serveur pour calculer certains champs tel que la vitesse moyenne et stocker les positions GPS dans la table des postions.

Lors de la création d’une activité, certains champs doivent être calculer, tel que la vitesse moyenne par exemple. Ces calculs devraient se réaliser après le retour du message de réussite de création de l’activité à l’utilisateur, car ces calcules peuvent prendre du temps, et qu’ils ne sont pas nécessaire directement à l’utilisateur.

Plusieurs méthodes sont envisageables pour gérer les calculs réalisés après le retour à l’utilisateur :

* Lancer un nouveau processus NodeJS
* Lancer un nouveau thread NodeJS
* Exécuter une tache de fond après une certaine durée
* Exécuter les calculs à la suite du retour (directement dans la méthode)

Étant limité par le temps, l'option d'exécuter les calculs directement à la suite du retour utilisateur a été choisi. Cette option comporte néanmoins des désavantages assez conséquents. Il n'est pas possible, avec l'architecture actuelle, de faire un retour à l'utilisateur suite à l'exécution des calculs. Il serait possible, dans la suite du développement, d'implémenter un système de notifications à l'utilisateur.

### Positions GPS

Les positions GPS sont une entité de la base de données, ils seront traités en tant qu’attribut de la classe d’activités au sein de l’API, car l’existence d’une postions ne fait pas de sens si elle n’appartient pas à une activité sportive.

## Ressources de l'API

|  |  |
| --- | --- |
| Ressource | Utilisation |
| User | Stocke les données d’un utilisateur |
| Token | Créer un token d'authentification |
| Activity | Stocke une activité sportive réalisé par un sportif |
| Subscription | Stocke l’abonnement d’un utilisateur (date de début, date de fin, …) |
| Position | Stocke une position GPS à un temps donné |

## Endpoints de l’API

|  |  |  |
| --- | --- | --- |
| Fonction | Endpoints | Verbe HTTP |
| Création compte | /user | POST |
| Modification compte | /user/{userid} | PUT |
| Authentification | /token | GET |
| Création d’activité | /user/{userid}/activity/ | POST |
| Lecture d’activité | activity/{activityid} | GET |
| /user/{userid}/activity/{activityid} |
| Modification d’activité | activity/{activityid} | PUT |
| /user/{userid}/activity/{activityid} |
| Suppression d’activité | activity/{activityid} | DELETE |
| /user/{userid}/activity/{activityid} |
| Modification d’une position | /position/{positionid} | PUT |
| Suppression d’une position | /position/{positionid} | DELETE |
| Création type d’activité | /activity-type/ | POST |
| Lecture types d’activité | /activity-type/ | GET |
| Suppression type d’activité | /activity-type/{typeid} | DELETE |
| Modification type d’activité | /activity-type/{typeid} | PUT |
| Ajout abonnement | /subscription/ | POST |
| Modification abonnement | /subscription/{subid} | PUT |
| Lecture abonnement | /user/{userid}/subscription | GET |

## Version des logiciels

|  |  |
| --- | --- |
| NodeJS | v8.11.1 |
| npm | 1.4.21 |
| MySQL | 10.1.37-MariaDB-0+deb9u1 |
| Raspbian | stretch 9.4 |
| git | git version 2.11.0 |

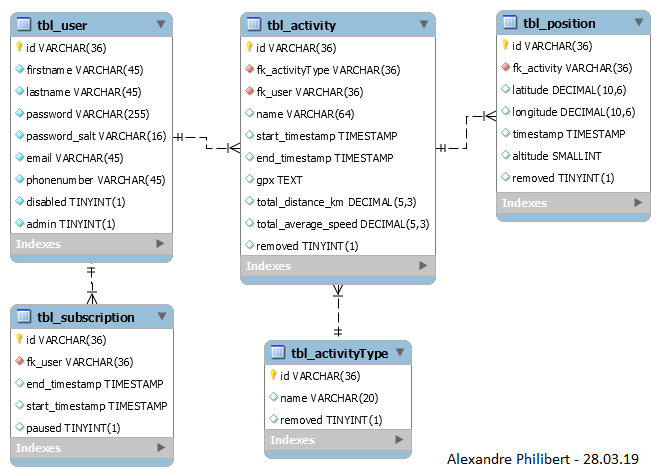
## Choix du moteur de base de données

Le moteur de base de données utilisé est InnoDB, car il est le moteur de base de données par défaut fournit avec MySQL. Ce moteur remplis toutes les contraintes imposées par le projet et l'architecture choisie (Clé primaire, clé étrangère, transactions, types de champs).

Un moteur relationnel a été choisi car je ne maitrise pas de moteur non-relationnel tels que MongoDB.

## Modèle logique des données

Le modèle présenté dans cette section n’est pas définitif, il permet d’avoir une idée globale de l’architecture de la base de données. Des modifications peuvent être apportées lorsque l’architecture ne s’avère plus adaptée.



Vous trouverez, ci-dessous, une description des champs pouvant contenir des informations spéciales. D’autres champs ne sont pas détaillés car le contenu de ceux-ci est basique.

### Généralités

Les identifiants (clé primaire et clé étrangères) sont en VARCHAR(36) car le choix de stocker des UUIDs a été fait. Il permet de générer des identifiants différents sur des systèmes distribués et de ne pas avoir de problème de collision d'identifiants.

### User

|  |  |
| --- | --- |
| Champs | Utilisation |
| password | Stocke le hash du mot de passe |
| disabled | Définit si un compte utilisateur est activé ou non |
| admin | Définit si un compte utilisateur est administrateur ou non |

### Activity

|  |  |
| --- | --- |
| Champs | Utilisation |
| start\_timestamp | Timestamp de début d’activité |
| end\_timestamp | Timestmap de fin d’activité |
| removed | Définit si une activité a été supprimée |

### Position

|  |  |
| --- | --- |
| Champs | Utilisation |
| latitude | Stocke la latitude de l’enregistrement de position avec une précision d’environ 10cm |
| longitude | Stocke la longitude de l’enregistrement de position avec une précision d’environ 10cm |
| altitude | Stocke l’altitude de l’enregistrement de positon avec comme point de référence le niveau de la mer |
| removed | Définit si un enregistrement a été supprimer |

### Subscription

|  |  |
| --- | --- |
| Champs | Utilisation |
| end\_timestamp | Timestamp de fin d’abonnement |
| start\_timestamp | Timestamp de début d’abonnement |
| paused | Définit si l’abonnement est mis en pause |

### ActivityType

|  |  |
| --- | --- |
| Champs | Utilisation |
| name | Nom du type d’activité |
| removed | Définit si le type d’activité a été supprimer |

## Interface web d'administration

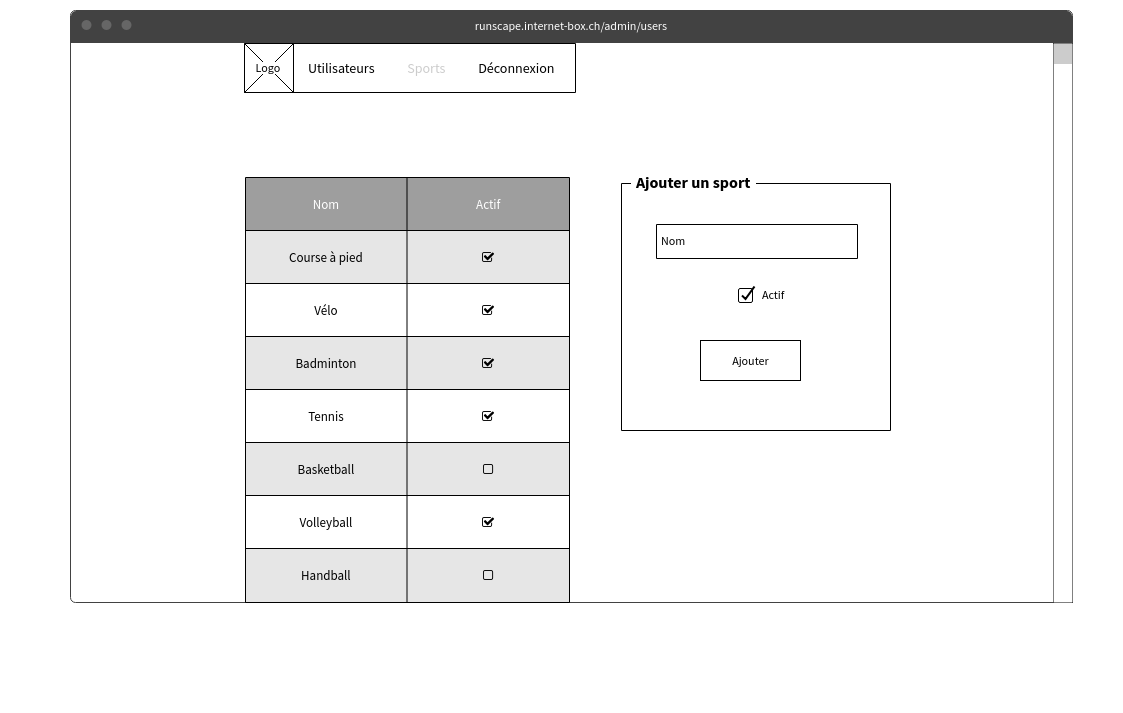
### Connexion à l’interface web

La connexion à l’interface web utilisera l’identifiant unique naturel d’un utilisateur (email). L’utilisateur devra aussi fournir un mot de passe. Des appels à l’API seront effectué sur le endpoint d’obtention du token.

### Gestion des activités sportives

Une page de gestion des activités sportives permettra de consulter, d'ajouter, de modifier et de désactiver les activités sportives. Une vérification du token sera effectuée avant que la page ne charge pour vérifier que la personne souhaitant consulter la page soit bien un administrateur.

### Wireframes



## Diagrammes de flux









## Cas d’utilisation

### Diagramme des cas d’utilisation



### Cas d’utilisation visiteur

|  |  |  |
| --- | --- | --- |
| En tant que visiteur | Je veux créer un compte | Dans le but de devenir un membre |

|  |  |  |
| --- | --- | --- |
| Déroulement | Étape | Description |
| 1 | Visiteur : Soumet une requête POST à "/user" |
| 2 | API : Valide la requête |
| 3 | API : retourne une réponse |
| Extensions | 2a | Format d'email invalide  API : Retourne une erreur de format d'email invalide |
| 2b | Format de mot de passe invalide  API : Retourne une erreur de format de mot de passe |
| 2c | Champs manquant dans la requête  API : Retourne une erreur de champs manquant |

|  |  |  |
| --- | --- | --- |
| En tant que visiteur | Je veux m'authentifier | Dans le but d'accéder au contenu nécessitant une authentification |

|  |  |  |
| --- | --- | --- |
| Déroulement | Étape | Description |
| 1 | Visiteur : Soumet une requête POST à "/token" |
| 2 | API : Valide la requête |
| 3 | API : retourne une réponse avec un token |
| Extensions | 2a | Aucun email ne correspond dans la base de données  API : Retourne une erreur de connexion |
| 2b | Le mot de passe n'est pas valide  API : Retourne une erreur de connexion |
| 2c | Champs manquant dans la requête  API : Retourne une erreur de champs manquant |

### Cas d'utilisation membre

|  |  |  |
| --- | --- | --- |
| En tant que membre | Je veux enregistrer un GPX | Dans le but d'analyser mon activité sportive |

|  |  |  |
| --- | --- | --- |
| Déroulement | Étape | Description |
| 1 | Membre : Soumet une requête POST à "user/{userid}/activity" |
| 2 | API : Valide la requête |
| 3 | API : Valide l'autorisation de création de la ressource |
| 4 | API : enregistre les informations dans la base de données |
| 5 | API : Retourne l'état de l'enregistrement au membre |
| Extensions | 2a | Autorisation non valide  API : Retourne une erreur d'autorisation |
| 2b | Format de GPX invalide  API : Retourne une erreur de format GPX invalide |
| 3a | Format de date de début invalide  API : Retourne une erreur de format de date |
| 3b | Format de date de fin invalide  API : Retourne une erreur de format de date |
| 3c | ID de type d'activité inexistant  API : Retourne une erreur de format d'activité |
| 3d | Champs fichier GPX vide, autres champs vides  API : Retourne une erreur de champs manquant |

|  |  |  |
| --- | --- | --- |
| En tant que membre | Je veux enregistrer une activité sportive sans GPX | Dans le but d'analyser mon activité sportive |

|  |  |  |
| --- | --- | --- |
| Déroulement | Étape | Description |
| 1 | Membre : Soumet une requête POST à "user/{userid}/activity" |
| 2 | API : Valide la requête |
| 3 | API : Valide l'autorisation de création de la ressource |
| 4 | API : enregistre les informations dans la base de données |
| 5 | API : Retourne l'état de l'enregistrement au membre |
| Extensions | 2a | Autorisation non valide  API : Retourne une erreur d'autorisation |
| 3a | Format de date de début invalide  API : Retourne une erreur de format de date |
| 3b | Format de date de fin invalide  API : Retourne une erreur de format de date |
| 3c | ID de type d'activité inexistant  API : Retourne une erreur de format d'activité |
| 3d | Champs fichier GPX vide, autres champs vides  API : Retourne une erreur de champs manquant |

|  |  |  |
| --- | --- | --- |
| En tant que membre | Je veux consulter une activité sportive | Dans le but de voir mon résultat |

|  |  |  |
| --- | --- | --- |
| Déroulement | Étape | Description |
| 1 | Membre : Soumet une requête GET à "/activity/{activityid}" |
| 2 | API : Valide la requête |
| 3 | API : Valide l'autorisation d'accès à la ressource |
| 4 | API : Sélectionne l'activité sportive demandé par l'utilisateur dans la base de données |
| 5 | API : Retourne l'activité au membre |
| Extensions | 2a | Format de la requête invalide  API : Retourne une erreur de format de requête |
| 3a | Autorisation non valide  API : Retourne une erreur d'autorisation |
| 4a | L'activité n'existe pas  API : Retourne une information mentionnant l'inexistence de l'activité |
| 4b | L'utilisateur n'existe pas  API : Retourne une information mentionnant l'inexistence du membre |

|  |  |  |
| --- | --- | --- |
| En tant que membre | Je veux mettre à jour une activité sportive | Dans le but de rectifier une erreur que j'aurais faite lors de la création |

|  |  |  |
| --- | --- | --- |
| Déroulement | Étape | Description |
| 1 | Membre : Soumet une requête PUT à "/activity/{activityid}" |
| 2 | API : Valide la requête |
| 3 | API : Valide l'autorisation d'accès à la ressource |
| 4 | API : Mets à jour l'activité sportive dans la base de données |
| 5 | API : Retourne l'activité au membre |
| Extensions | 2a | Format de la requête invalide  API : Retourne une erreur de format de requête |
| 3a | Autorisation non valide  API : Retourne une erreur d'autorisation |
| 4a | L'activité n'existe pas  API : Retourne une information mentionnant l'inexistence de l'activité |
| 4b | L'utilisateur n'existe pas  API : Retourne une information mentionnant l'inexistence du membre |

|  |  |  |
| --- | --- | --- |
| En tant que membre | Je veux consulter la liste de tous les types d'activité | Dans le but de choisir le bon type d'activité pour l'activité que je créer |

|  |  |  |
| --- | --- | --- |
| Déroulement | Étape | Description |
| 1 | Membre : Soumet une requête GET à "/activity-type" |
| 2 | API : Valide la requête |
| 3 | API : Sélectionne tous les types d'activités |
| 4 | API : Retourne les types d'activités |
| Extensions | 2a | Format de la requête invalide  API : Retourne une erreur de format de requête |

### Cas d'utilisation administrateur

|  |  |  |
| --- | --- | --- |
| En tant qu'administrateur | Je veux ajouter un type d'activité | Dans le but de créer des activités d'un type encore inexistant |

|  |  |  |
| --- | --- | --- |
| Déroulement | Étape | Description |
| 1 | Administrateur : Soumet une requête POST à "/activity-type}" |
| 2 | API : Valide la requête |
| 3 | API : Valide l'autorisation d'accès à la ressource |
| 4 | API : Créer le nouveau type d'activité |
| 5 | API : Retourne l'identifiant du type d'activité créer |
| Extensions | 2a | Format de la requête invalide  API : Retourne une erreur de format de requête |
| 3a | Token non valide  API : Retourne une erreur d'autorisation |
| 3b | L'utilisateur n'est pas un administrateur  API: Retourne une erreur d'autorisation |
| 4a | La base de données ne répond pas  API: Retourne une erreur de serveur |

|  |  |  |
| --- | --- | --- |
| En tant qu'administrateur | Je veux mettre à jour un type d'activité | Dans le but de rectifier une erreur que j'aurais faite lors de la création |

|  |  |  |
| --- | --- | --- |
| Déroulement | Étape | Description |
| 1 | Administrateur : Soumet une requête PUT à "/activity-type}" |
| 2 | API : Valide la requête |
| 3 | API : Valide l'autorisation d'accès à la ressource |
| 4 | API : Mets à jour le type d'activité |
| 5 | API : Retourne l'état de la mise à jour (réussi ou non) |
| Extensions | 2a | Format de la requête invalide  API : Retourne une erreur de format de requête |
| 3a | Token non valide  API : Retourne une erreur d'autorisation |
| 3b | L'utilisateur n'est pas un administrateur  API: Retourne une erreur d'autorisation |
| 4a | La base de données ne répond pas  API: Retourne une erreur de serveur |

|  |  |  |
| --- | --- | --- |
| En tant qu'administrateur | Je veux supprimer un type d'activité | Dans le but de ne plus proposé ce type d'activité sur l'application |

|  |  |  |
| --- | --- | --- |
| Déroulement | Étape | Description |
| 1 | Administrateur : Soumet une requête DELETE à "/activity-type/{activityTypeId}" |
| 2 | API : Valide la requête |
| 3 | API : Valide l'autorisation d'accès à la ressource |
| 4 | API : Supprime le type d'activité |
| 5 | API : Retourne l'état de la suppression (réussi ou non) |
| Extensions | 2a | Format de la requête invalide  API : Retourne une erreur de format de requête |
| 2b | Identifiant du type d'activité n'existe pas dans la base de données  API : Retourne une erreur spécifiant que l'identifiant n'existe pas dans la base de données |
| 3a | Token non valide  API : Retourne une erreur d'autorisation |
| 3b | L'utilisateur n'est pas un administrateur  API: Retourne une erreur d'autorisation |
|  | 4a | La base de données ne répond pas  API: Retourne une erreur de serveur |

# Réalisation

## Configuration serveur

### Utilisateurs Raspbian

|  |  |  |  |
| --- | --- | --- | --- |
| Nom d'utilisateur | Fonction | Accès SSH | Accès sudo |
| pi | Utilisateur par défaut de Raspbian, il n’a pas été supprimé car des fonctionnalités de Raspbian ont besoin de cet utilisateur. | Non | Oui |
| alexandre | Administrateur du serveur | Oui | Oui |

### Paquets installés

Plusieurs paquets ont été ajoutés, en plus de la configuration par défaut de Raspbian, vous trouverez la liste des paquets ajoutés et mis à jour ci-dessous :

|  |  |
| --- | --- |
| Nom du paquet | Version |
| openssh-server | OpenSSH\_7.4p1 Raspbian-10+deb9u5, OpenSSL 1.0.2q 20 Nov 2018 |
| ufw | 0.35 |
| fail2ban | Fail2Ban v0.9.6 |
| nodejs | v8.11.1 |
| npm | 1.4.21 |
| certbot | certbot 0.28.0 |

### Configuration SSH

Le fichier de configuration SSH se trouve à l'emplacement "/etc/ssh/sshd\_config", les commandes modifiées sont :

|  |
| --- |
| ChallengeResponseAuthentication no  PasswordAuthentication no  DenyUsers pi  UsePAM no |

### Configuration firewall

Le firewall utilisé sur le serveur est ufw, par sa simplicité de configuration, voici la liste des ports ouverts :

|  |
| --- |
| To Action From  -- ------ ----  443 ALLOW Anywhere  80 ALLOW Anywhere  22 ALLOW Anywhere  443 (v6) ALLOW Anywhere (v6)  80 (v6) ALLOW Anywhere (v6)  22 (v6) ALLOW Anywhere (v6) |

Seulement les ports par défauts des protocoles HTTP, HTTPS et SSH sont autorisé en version IPv4 et IPv6.

### Configuration fail2ban

Un outil de prévention contre les intrusions à été installé sur le serveur, il permet de bannir les adresses IP après un nombre de tentatives de connexion trop importantes sur le port SSH par exemple. La configuration de fail2ban à été laissé par défaut, seulement la ligne suivantes du fichier "/etc/fail2ban/jail.local" sous la section [sshd] a été modifiée :

|  |
| --- |
| maxretry = 10 |

### Installation certificat SSL

L’installation du certificat SSL à été faite à l’aide de certbot. Vous trouverez ci-dessous la procédure d’installation du certificat :

|  |
| --- |
| sudo certbot certonly --manual |

Cette commande permet l’installation manuel du certificat, certbot n’ayant pas d’utilitaire d’aide pour NodeJS.

|  |
| --- |
| Please enter in your domain name(s) (comma and/or space separated) (Enter 'c' to cancel): runscape.internet-box.ch |

Entrer le nom de domaine à certifier. Il faudra par la suite rendre disponible sur le serveur une chaîne de charactère à une URL spécifique du serveur avant de continuer le script.

Une fois la page mis à disposition, reprendre l’exécution du script. Le fichier de clé privée et le certificat seront générés au chemin suivant :

|  |
| --- |
| /etc/letsencrypt/live/runscape.internet-box.ch/privkey.pem  /etc/letsencrypt/live/runscape.internet-box.ch/cert.pem |

### Création d'un utilisateur MYSQL applicatif

Pour créer un nouvel utilisateur MSQL, connectez-vous à MYSQL sur le serveur :

|  |
| --- |
| sudo mysql |

Exécutez la directive SQL suivante (en remplaçant "username et "password") :

|  |
| --- |
| GRANT ALL PRIVILEGES ON runscape.\* TO 'username'@'localhost' IDENTIFIED BY 'password'; |

### Démarrage de l'application

Plusieurs manières sont à disposition pour démarrer l'application, le plus simple étant la commande suivante :

|  |
| --- |
| sudo node app.js |

Il est aussi possible de lancer le serveur à l'aide du module 'forever' :

|  |
| --- |
| sudo forever –w –l err.log app.js |

La directive **–w** permettant de relancer le serveur lors de modifications de fichiers et la directive **–l** permettant de définir un fichier de log des erreurs.

## Description des tests effectués

|  |  |  |
| --- | --- | --- |
| En tant que visiteur | Je veux créer un compte | Dans le but de devenir un membre |

|  |  |  |  |
| --- | --- | --- | --- |
| Déroulement | Étape | Description | Résultat |
| 1 | Visiteur : Soumet une requête POST à "/user" |  |
| 2 | API : Valide la requête |  |
| 3 | API : retourne une réponse |  |
| Extensions | 2a | Format d'email invalide  API : Retourne une erreur de format d'email invalide | Réussi |
| 2b | Format de mot de passe invalide  API : Retourne une erreur de format de mot de passe | Réussi |
| 2c | Champs manquant dans la requête  API : Retourne une erreur de champs manquant | Réussi |

|  |  |  |
| --- | --- | --- |
| En tant que visiteur | Je veux m'authentifier | Dans le but d'accéder au contenu nécessitant une authentification |

|  |  |  |  |
| --- | --- | --- | --- |
| Déroulement | Étape | Description | Résultat |
| 1 | Visiteur : Soumet une requête POST à "/token" |  |
| 2 | API : Valide la requête |  |
| 3 | API : retourne une réponse avec un token |  |
| Extensions | 2a | Aucun email ne correspond dans la base de données  API : Retourne une erreur de connexion | Réussi |
| 2b | Le mot de passe n'est pas valide  API : Retourne une erreur de connexion | Réussi |
| 2c | Champs manquant dans la requête  API : Retourne une erreur de champs manquant | Réussi |

|  |  |  |
| --- | --- | --- |
| En tant que membre | Je veux enregistrer un GPX | Dans le but d'analyser mon activité sportive |

|  |  |  |  |
| --- | --- | --- | --- |
| Déroulement | Étape | Description | Résultat |
| 1 | Membre : Soumet une requête POST à "user/{userid}/activity" |  |
| 2 | API : Valide la requête |  |
| 3 | API : Valide l'autorisation de création de la ressource |  |
| 4 | API : enregistre les informations dans la base de données |  |
| 5 | API : Retourne l'état de l'enregistrement au membre |  |
| Extensions | 2a | Autorisation non valide  API : Retourne une erreur d'autorisation |  |
| 2b | Format de GPX invalide  API : Retourne une erreur de format GPX invalide |  |
| 3a | Format de date de début invalide  API : Retourne une erreur de format de date | Réussi |
| 3b | Format de date de fin invalide  API : Retourne une erreur de format de date | Réussi |
| 3c | ID de type d'activité inexistant  API : Retourne une erreur de format d'activité | Réussi |
| 3d | Champs fichier GPX vide, autres champs vides  API : Retourne une erreur de champs manquant |  |

|  |  |  |
| --- | --- | --- |
| En tant que membre | Je veux enregistrer une activité sportive sans GPX | Dans le but d'analyser mon activité sportive |

|  |  |  |  |
| --- | --- | --- | --- |
| Déroulement | Étape | Description | Résultat |
| 1 | Membre : Soumet une requête POST à "user/{userid}/activity" |  |
| 2 | API : Valide la requête |  |
| 3 | API : Valide l'autorisation de création de la ressource |  |
| 4 | API : enregistre les informations dans la base de données |  |
| 5 | API : Retourne l'état de l'enregistrement au membre |  |
| Extensions | 2a | Autorisation non valide  API : Retourne une erreur d'autorisation |  |
| 3a | Format de date de début invalide  API : Retourne une erreur de format de date |  |
| 3b | Format de date de fin invalide  API : Retourne une erreur de format de date |  |
| 3c | ID de type d'activité inexistant  API : Retourne une erreur de format d'activité |  |
| 3d | Champs fichier GPX vide, autres champs vides  API : Retourne une erreur de champs manquant |  |

## Erreurs restantes

|  |  |  |
| --- | --- | --- |
| Description détaillée | Conséquences sur l’utilisation | Actions possibles |
|  |  |  |
|  |  |  |

*S'il reste encore des erreurs:*

* *Description détaillée*
* *Conséquences sur l'utilisation du produit*
* *Actions envisagées ou possibles*

## Liste des documents fournis

*Lister les documents fournis au client avec votre produit, en indiquant les numéros de versions*

* *le rapport de projet*

le manuel d'Installation (en annexe)

* *le manuel d'Utilisation avec des exemples graphiques (en annexe)*
* *autres…*

# Conclusions

## Objectifs atteints

* L'administrateur de l'application
  + Editer la liste des sports disponibles sur l'application
  + Désactiver un membre
* Le sportif
  + Peut consulter la liste des activités qu'il a réalisé
* Le back-end permet de manipuler des données modélisant des activités sportives
  + La date
  + L'heure de départ
  + Le type d'activité
  + La durée de l'effort
  + Distance parcourue
  + Le parcours (enregistrement des positions GPS)
  + La vitesse moyenne
* Sécurisation du back-end
  + N'être accessible que via le protocole HTTPS
  + Aucun mot de passe ne doit être présent dans la base de données ni dans les logs applicatifs
  + Le code applicatif ne doit pas avoir la capacité de supprimer (ni DELETE ni DROP) des données

## Objectifs non-atteints

* L'administration de l'application
  + Offrir 3 mois gratuit à un ou plusieurs membres
* Le sportif
  + Il peut s'abonner (formule mensuelle) et ainsi accéder à toutes les fonctionnalités des membres
  + L'abonnement peut être mis en "standby" en cas de blessures par exemple. Durant ce type, la consultation reste possible mais il n'est pas possible d'ajouter de nouvelles activités
* Le back-end permet de manipuler des données modélisant des activités sportives
  + Les dénivelés positifs et négatifs
* Le back-end devra être capable de dialoguer avec un composant client qui effectuera les demandes suivantes
  + Enregistrer / Modifier / Supprimer une activité
  + Consulter et analyser les activités sur une période donnée (moyenne, nombre de kilomètres parcourus par sport)
  + S'interfacer avec au moins un acteur majeur de gestion d'activité tes que Garmin Connect, Tom tom
* Authentification des utilisateurs
  + Soit pouvoir se connecter à l'aide d'un service "2-step verification"
  + Soit s'appuyer sur les API de Facebook

## Difficultés particulières

La seule difficulté que j'ai rencontrée lors de la réalisation de ce projet était l'aspect asynchrone de NodeJS. J'avais déjà, auparavant, rencontré des architectures d'application asynchrone. Je n'ai par contre, jamais dû créer mon propre *Framework* au-dessus d'une architecture de nature asynchrone.

Je pense que ce projet m'a apporté beaucoup de connaissances me permettant de conceptualiser des architectures plus robustes. J'avais dans le début du projet, était soumis

Je pense que le temps mit à disposition pour la réalisation du projet ne suffisait pas pour réaliser tous les points détaillés.

## Suites possibles pour le projet

### Terminer le cahier des charges

Avec le travail qui a déjà été effectué, le reste des fonctionnalités pourraient être intégrées assez rapidement.

### Réécriture de l'architecture de l'API

### 

Lors de ce projet, j'ai plusieurs fois changé le cœur de l'API pour répondre au mieux aux objectifs du projet. Je n'ai malheureusement pas eu le temps d'effectuer tous les changements que je souhaitai. Je pense qu'une réécriture de l'API au niveau de la gestion des routes permettrai d'accélérer le développement par la suite, notamment par l'intégration d'une authentification sur les endpoints.

### Amélioration de la gestion d'activité

La gestion des activités étant le centre du projet, je pense que rendre cette partie plus robuste serait un avantage considérable. L'une des principales améliorations que j'envisage serait l'implémentation d'une gestion en arrière-plan du calcul des champs tel que la vitesse moyenne et la distance totale. L'API pourrait par exemple, intégrer un système de notification à l'utilisateur permettant de l'informer lorsque les champs de l'activité ont été calculés.

### Tests automatisés

C’est un élément qui a manqué lors du développement de l’API. C’est un élément qui devient indispensable lorsque le projet prend de l’envergure. Un outil tels que Postman permet de très facilement mettre en place des tests sur les endpoints d’une API.

### Rendre l’API scalable

Lorsque beaucoup de personnes utilisent un service mit à disposition sur un serveur, celui-ci se voit très vite débordé par la charge de calcul. Avoir un système avec plusieurs nœuds de calculs permettrait de facilement adapter l’infrastructure à la charge de calcul et de minimiser les pannes.

# Annexes

## Résumé du rapport du TPI / version succincte de la documentation

## Sources – Bibliographie

Documentation nodeJS  [nodejs.org](https://nodejs.org)

Documentation modules [npmjs.com](https://www.npmjs.com/)

Divers problèmes  [stackoverflow.com](https://stackoverflow.com/)

Validation des JWT  [jwt.io](https://jwt.io/)

Installation Raspberry Pi [www.raspberrypi.org/documentation/](http://www.raspberrypi.org/documentation/)

Architecture API [oreilly.com/learning/how-to-design-a-restful-api-architecture-from-a-human-language-spec](https://www.oreilly.com/learning/how-to-design-a-restful-api-architecture-from-a-human-language-spec)

## Journal de travail

## 

## Manuel d'Installation

## Manuel d'Utilisation

## Archives du projet

*Media, … dans une fourre en plastique*

1. L’utilisateur doit être en mesure de synchroniser ses activités sportives avec un autre Web Service [↑](#footnote-ref-1)
2. Stocké sous forme de hash par exemple [↑](#footnote-ref-2)
3. Pour autant que le téléphone dispose d’un GPS, gyroscope et accéléromètre. [↑](#footnote-ref-3)
4. Le GPX est un format de fichier qui permet d'enregistrer des positions GPS ainsi que des horodatages. [↑](#footnote-ref-4)