

# **FitJourney**

Travail de diplôme Technicien ES

---

*Thomas Fujise*

*CFPT Informatique 2022, Version 1.0*

# Table des matières

---

1. Documentation technique	5
1.1 Résumé	5
1.2 Abstract	5
1.3 Introduction	5
1.4 Cahier des charges	6
1.4.1 Contexte	6
1.4.2 But du projet	6
1.4.3 Cas d'utilisations	6
1.4.4 Spécifications	9
1.4.5 Restrictions	9
1.4.6 Environnement	9
1.4.7 Contact	10
1.4.8 Dates importantes	11
1.4.9 Livrable	11
1.5 Analyse de l'existant	12
1.5.1 Inithy	12
1.6 Analyse fonctionnelle	13
1.6.1 Fonctionnalités	13
1.6.2 Sitemap	15
1.6.3 Maquettage	17
1.7 Analyse Organique	33
1.7.1 Mise en place / Environnement	33
1.7.2 Organisation / Gestion du temps	38
1.7.3 Technologies utilisées	42
1.7.4 Architecture du projet	51
1.7.5 Base de données	56
1.7.6 FitJourney application	62
1.7.7 FitJourney cards checker	84
1.8 Tests	86
1.8.1 Tests unitaires	86
1.8.2 Tests utilisateurs	87

1.9 Conclusion	95
1.9.1 Points positifs	95
1.9.2 Problèmes rencontrés	96
1.9.3 Améliorations possibles	97
1.9.4 Bilan personnel	97
1.10 Sources	98
2. Logbook	99
2.1 Journal de bord - Thomas Fujise	99
2.1.1 Lundi, 04 Avril 2022	99
2.1.2 Mardi, 05 Avril 2022	99
2.1.3 Mercredi, 06 Avril 2022	99
2.1.4 Jeudi, 07 Avril 2022	100
2.1.5 Vendredi, 08 Avril 2022	100
2.1.6 Lundi, 11 Avril 2022	101
2.1.7 Mardi, 12 Avril 2022	101
2.1.8 Mercredi, 13 Avril 2022	102
2.1.9 Lundi, 25 Avril 2022	103
2.1.10 Mardi, 26 Avril 2022	105
2.1.11 Mercredi, 27 Avril 2022	107
2.1.12 Jeudi, 28 Avril 2022	108
2.1.13 Vendredi, 29 Avril 2022	109
2.1.14 Lundi, 2 Mai 2022	112
2.1.15 Mardi, 3 Mai 2022	113
2.1.16 Mercredi, 4 Mai 2022	115
2.1.17 Jeudi, 5 Mai 2022	116
2.1.18 Vendredi, 6 Mai 2022	117
2.1.19 Lundi, 9 Mai 2022	120
2.1.20 Mardi, 10 Mai 2022	124
2.1.21 Mercredi, 11 Mai 2022	124
2.1.22 Jeudi, 12 Mai 2022	125
2.1.23 Vendredi, 13 Mai 2022	125
2.1.24 Lundi, 16 Mai 2022	126
2.1.25 Mardi, 17 Mai 2022	127
2.1.26 Mercredi, 18 Mai 2022	127
2.1.27 Jeudi, 19 Mai 2022	128

2.1.28 Vendredi, 20 Mai 2022	129
2.1.29 Lundi, 23 Mai 2022	129
2.1.30 Mardi, 24 Mai 2022	129
2.1.31 Mercredi, 25 Mai 2022	130
2.1.32 Vendredi, 26 Mai 2022	131
2.1.33 Lundi, 30 Mai 2022	131
2.1.34 Mardi, 31 Mai 2022	132
2.1.35 Mercredi, 01 Juin 2022	133
2.1.36 Jeudi, 02 Juin 2022	133
2.1.37 Vendredi, 03 Juin 2022	134
2.1.38 Mardi, 07 Juin 2022	134
2.1.39 Mercredi, 08 Juin 2022	135
2.1.40 Jeudi, 09 Juin 2022	136

# 1. Documentation technique

---

## 1.1 Résumé

---

FitJourney est une application permettant, d'une part, à un coach sportif de gérer et effectuer le suivi de tous ses clients avec des données d'entrainements récupérées à l'aide de montres connectés Polar. D'autre part, elle permet aux clients d'accéder à leurs programmes d'entrainements et de nutrition. L'application permet également aux clients de visionner le détail de chacune de leurs séances.

FitJourney est une application WEB réalisée avec le framework Flask du langage Python, elle repose principalement sur l'API Polar Accesslink qui permet l'utilisation des données des montres connectées

Ce document reprend toutes les étapes du projet qui a été réalisé dans le cadre du travail de diplôme de la formation Technicien ES en informatique de Thomas Fujise.

## 1.2 Abstract

---

FitJourney is an application that allows personal trainers to follow up all their clients training data that is retrieved using Polar connected watches. It also allows clients to access their training and diet plan. The application allows clients to look at the details of each of their workouts.

FitJourney is a WEB application made with Flask Python framework. It relies mainly on the Polar Accesslink API which allows the use of data from connected watches.

This document contains all the steps of the project, which was carried out within the diploma project of the IT Technician formation of Thomas Fujise

## 1.3 Introduction

---

Dans le cadre de notre deuxième et dernière année au CFPT Informatique en tant que Technicien ES, il nous est demandé de réaliser un travail de diplôme dans le but de valider les compétences acquises tout au long de cette formation. J'ai fait le choix de développer "FitJourney", une application web qui facilite la pratique du métier de coach sportif.

Cette application permet de gérer le suivi des clients allant de l'importation des programmes individuels des clients à la récolte des données d'entrainements à l'aide de montre connecté Polar. Elle permet également de gérer une salle de sport avec les cartes de membres qui permettent l'accès à la salle. Ayant passé un diplôme de coach sportif l'année passée, j'étais à l'aise avec les besoins qu'un professionnel aurait en cas d'utilisation de l'application.

## 1.4 Cahier des charges

---

### 1.4.1 Contexte

---

Développer une application web regroupant des clients et des coachs de sport. L'application permet le suivi des clients et peut récupérer des données d'entraînements directement depuis un appareil connecté Polar.

Il existe aujourd'hui, très peu d'outil qui permet à un coach sportif de gérer sa salle de sport avec le suivi de tous ses clients. C'est pourquoi, j'ai décidé de développer une application qui permettrait de gérer une salle de sports ainsi que le suivi des clients et de l'utiliser en tant que sujet pour mon travail de diplôme.

### 1.4.2 But du projet

---

Le but du projet est de créer une plateforme web regroupant clients et coachs afin de gérer une salle de sport et le suivi des clients.

Les données d'entraînements suivantes sont récupérées depuis une smartwatch Polar:

- Pulsation cardiaque (Repos/Actif/Après effort)
- Le type d'exercice effectué
- Date et durée de l'entraînement
- Nombre de total de calories brûlées
- Nombre de calories active

(Optionnel)

- Les données relatives au sommeil
  - Pulsation cardiaque
  - Cycles de sommeil
  - Interruption
  - Hypnogramme

Il y a également un système de badging pour savoir quand le client rentre et sort de la salle de sport à la fin de son entraînement.

### 1.4.3 Cas d'utilisations

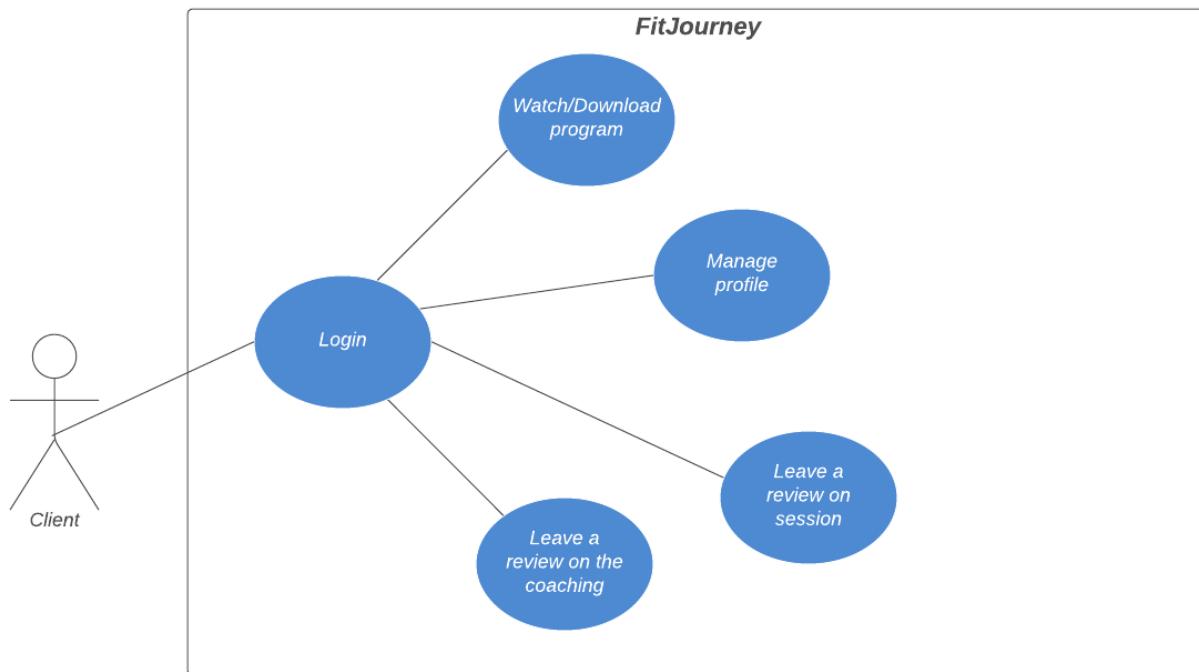
---

2 cas d'utilisations sont possibles avec l'application.

## Client

---

Le cas d'utilisation pour client :



Le client doit se connecter pour avoir accès à l'application, il ne peut pas se créer de compte lui-même.

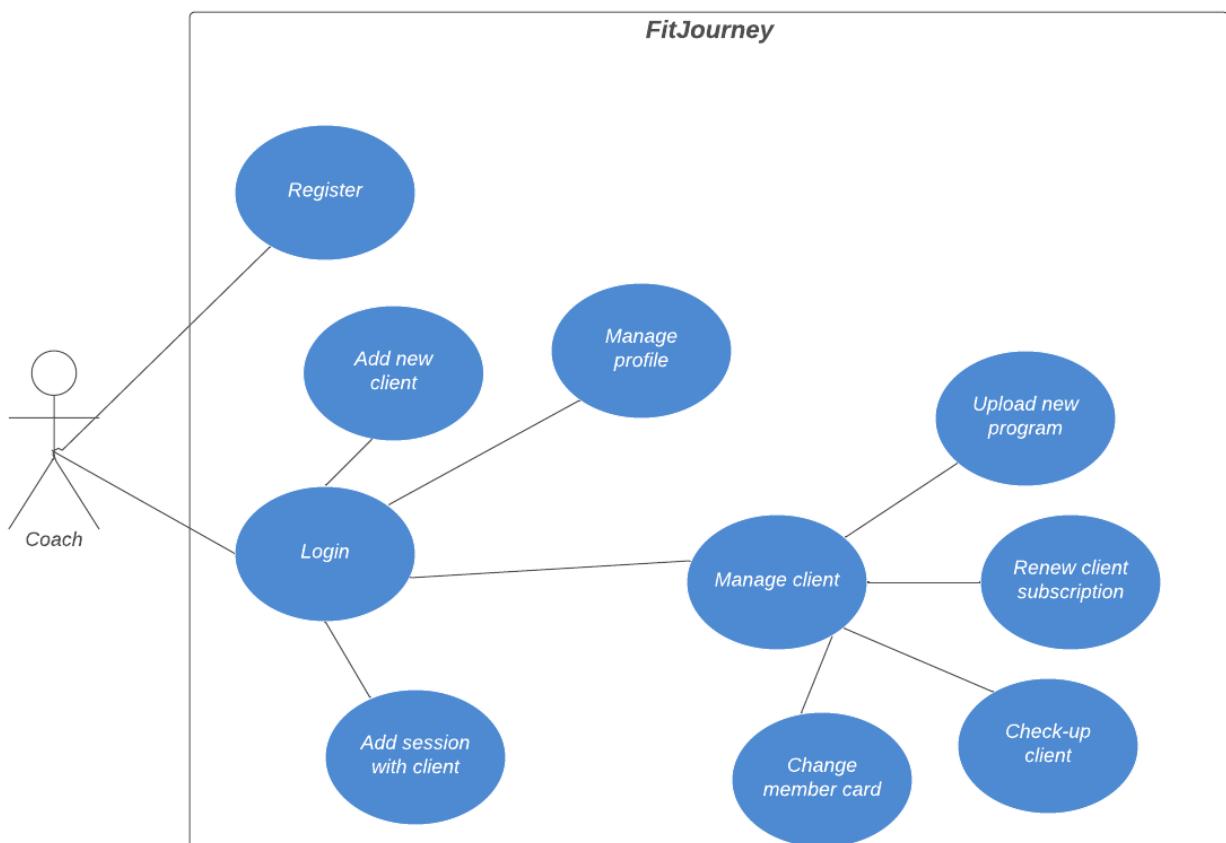
Si le client se connecte à l'application, il a alors accès à plusieurs fonctionnalités :

- Visualiser/Télécharger ses programmes (Entrainement et Nutrition)
- Laisser un retour sur une session effectué
- Laisser un retour sur le coaching de manière générale
- Gérer son profil

## Coach

---

Le cas d'utilisation pour coach :



Le coach à lui également 2 possibilités en arrivant sur l'application (Enregistrement et connexion).

Une fois connecté, le coach a accès à une multitude de fonctionnalités :

- Ajout d'un nouveau client (création du compte client)
- Ajouter une session avec un client (Prise de rendez-vous)
- Gérer son profil

Il a également accès à des fonctionnalités pour gérer ses clients :

- Importation des programmes (entraînement et nutrition)
- Renouveler l'abonnement souscrit par le client
- Effectuer un bilan avec un client
- Changer la carte membre du client

## 1.4.4 Spécifications

---

- Les données d'entrainements sont récupérées avec l'API Accesslink de Polar en Python sous forme d'objet, elles sont vérifiées et stockées directement dans la base de données
- Le système de badge fonctionne à l'aide de cartes RFID et un lecteur NFC (ACS ACR 122u)
- Les graphiques liés aux stats sont effectués avec la librairie JavaScript Chart.js
- Les programmes d'entrainements et de nutritions doivent respecter le format proposé (Lors de l'upload une vérification est effectuée)
- Un système de notification est mis en place pour avertir les utilisateurs des différents événements

## 1.4.5 Restrictions

---

- Pour l'instant, il est uniquement possible d'utiliser des appareils Polar pour les données d'entrainement.
- Le système de badge est utilisé au début et à la fin et non pendant l'entrainement.
- L'API accepte au maximum 500 requête pour 20 utilisateurs différents en 15 min et 5000 pour 100 utilisateurs en 24h
- Les montres utilisées lors des entrainements sont celles du coach (montres déjà enregistrées sur le compte client qui accède à l'API)

## 1.4.6 Environnement

---

Les technologies à utiliser :

- HTML5
- CSS3
- JavaScript
- SQL
- Python
- Flask (Micro Framework Python)
- Lecteur NFC (ACS ACR122U)
- [API AccessLink Polar](#)
- [Chart.js](#)

Système d'exploitation :

- Développement sur Windows

Versionning / Documentation :

- GitHub
- MarkDown (Mkdocs)

## 1.4.7 Contact

---

### Étudiant

---

- **Prénom** : Thomas
- **Nom** : Fujise
- **Email** : thomas.fjs@eduge.ch
- **Téléphone** : +41 (0) 77 463 89 68

### Enseignant

---

- **Prénom** : Sébastien
- **Nom** : Jossi
- **Email** : edu-jossis@eduge.ch
- **Téléphone** : /

## 1.4.8 Dates importantes

---

Date	Tâches
<b>Lundi, 04 avril 2022</b>	Début du travail de diplôme
<b>Mardi, 26 avril 2022</b>	Évaluation intermédiaire 01
<b>Lundi, 09 mai 2022</b>	Rendu du rapport intermédiaire + poster & Rendu du résumé et de l'abstract
<b>Mardi, 10 mai 2022</b>	Évaluation intermédiaire 02
<b>Jeudi, 12 mai 2022</b>	Après-midi/Soirée poster
<b>Mardi, 24 mai 2022</b>	Évaluation intermédiaire 03
<b>Vendredi, 10 juin 2022</b>	Rendu du travail avant 12:00

## 1.4.9 Livrable

---

- Documentation technique + journal de bord
- Manuel utilisateur
- Poster
- Code source

## 1.5 Analyse de l'existant

### 1.5.1 Inithy

Inithy est une application de coaching "tout-en-un" qui propose énormément de fonctionnalités. Elle permet le suivi des clients, le planning de sessions, l'importation de programmes ou encore l'interaction entre utilisateurs de l'application. Elle possède une fonctionnalité de synchronisation avec les applications de sports comme GoogleFit, Apple Santé ou encore Fitbit. La synchronisation avec Polar et Garmin n'est pas disponible avec leur application. Il y a également la possibilité d'effectuer des paiements sur l'application pour acheter un abonnement. C'est une application très complète qui répond à beaucoup de besoins que j'ai identifié précédemment. Le principal point qui me différencie de cette application, est que FitJourney est plus axé pour gérer une salle de sport et les clients membres (pour un coach sportif qui aurait son studio de coaching par exemple) que pour le coaching en ligne comme l'est *Inithy*.



## 1.6 Analyse fonctionnelle

---

### 1.6.1 Fonctionnalités

---

#### Description générale

---

L'application est accessible par n'importe qui possédant un compte. Seuls les coachs ont accès à la page d'enregistrement pour se créer un nouveau compte et c'est également que les coachs qui peuvent créer un compte pour un client. Une fois connecté, l'utilisateur a accès à plusieurs fonctionnalités en fonction de son rôle (coach ou client)

#### Description détaillée

---

##### Application coach

---

- Le coach s'inscrit et s'authentifie avec un formulaire;
- Le coach peut modifier les informations propres à son compte utilisateur (nom, prénom, birthdate, email, photo de profil, adresse et mot de passe);
- Le coach peut ajouter un client en lui créant un compte et en sélectionnant le type d'abonnement que le client souhaite prendre;
- Le coach peut enregistrer une session qu'il prévoit avec un de ses clients en renseignant :
  - La date;
  - L'heure;
  - La durée;
  - Le type d'entraînement prévu;
- Le coach peut voir une vue d'ensemble de la liste de tous les clients dont il a la charge, il voit également la prochaine session qu'il a enregistrée avec un client avec un résumé;
- Le coach a accès à un calendrier pour visionner les sessions qu'il a enregistrées (plusieurs affichages disponibles : mois, semaine, jour ou encore en liste);
- Le coach peut effectuer un nouveau bilan/check-up pour le client

- Le coach à accès aux profils des clients dont il a la charge où il peut :
  - Renouveler/annuler l'abonnement du client;
  - Changer la carte de membre du client;
  - Importer un nouveau programme d'alimentation ou d'entraînement pour le client;
  - Visionner les anciens bilans qu'il a effectués
  - Visionner les reviews sur le coaching et les entraînements que le client a laissés;
  - Visionner les statistiques du client :
    - Nombre de types d'entraînement effectué;
    - Nombre d'entraînements chaque mois;
    - Le poids du client sur l'année;
    - Le nombre de calories brûlées dans la semaine;
    - La moyenne des BPM cardiaques enregistrés dans la semaine;
    - Le total de temps passé à s'entraîner dans la semaine;

## Application client

---

- Le client s'authentifie avec un formulaire à l'aide des identifiants qui lui sont transmis par email (les identifiants peuvent être modifiés par le client plus tard);
- Le client a un aperçu des prochaines sessions qu'il a avec son coach;
- Le client a un aperçu de tous les entraînements qu'il a effectués, en cliquant dessus il peut voir les détails de ce-dernier;
- Le client peut laisser un retour sur un entraînement qu'il a effectué, il peut également laisser un retour sur le coaching de manière générale;
- Le client peut télécharger les programmes que son coach a importés;
- Le client a accès à un aperçu des bilans qu'il a effectués avec son coach;
- Le client peut visionner les statistiques enregistrées pour lui :
  - Nombre de types d'entraînement effectué;
  - Nombre d'entraînements chaque mois;
  - Le poids du client sur l'année;
  - Le nombre de calories brûlées dans la semaine;
  - La moyenne des BPM cardiaques enregistrés dans la semaine;
  - Le total de temps passé à s'entraîner dans la semaine;

## Lecture carte membre

---

La lecture de carte membre est effectuée à l'aide d'un script indépendamment de l'application principale. Aucune interface n'est disponible pour le moment, seul un message est affiché dans le terminal pour afficher si le client a été reconnu ou non. Cette partie permet au client de :

- Scanner sa carte membre en arrivant à la salle de sport, la carte est reconnue et le client est identifié.
- Scanner sa carte de membre en sortant de la salle de sport, les données de l'entraînement effectué sont alors récupérés et enregistrés.

## 1.6.2 Sitemap

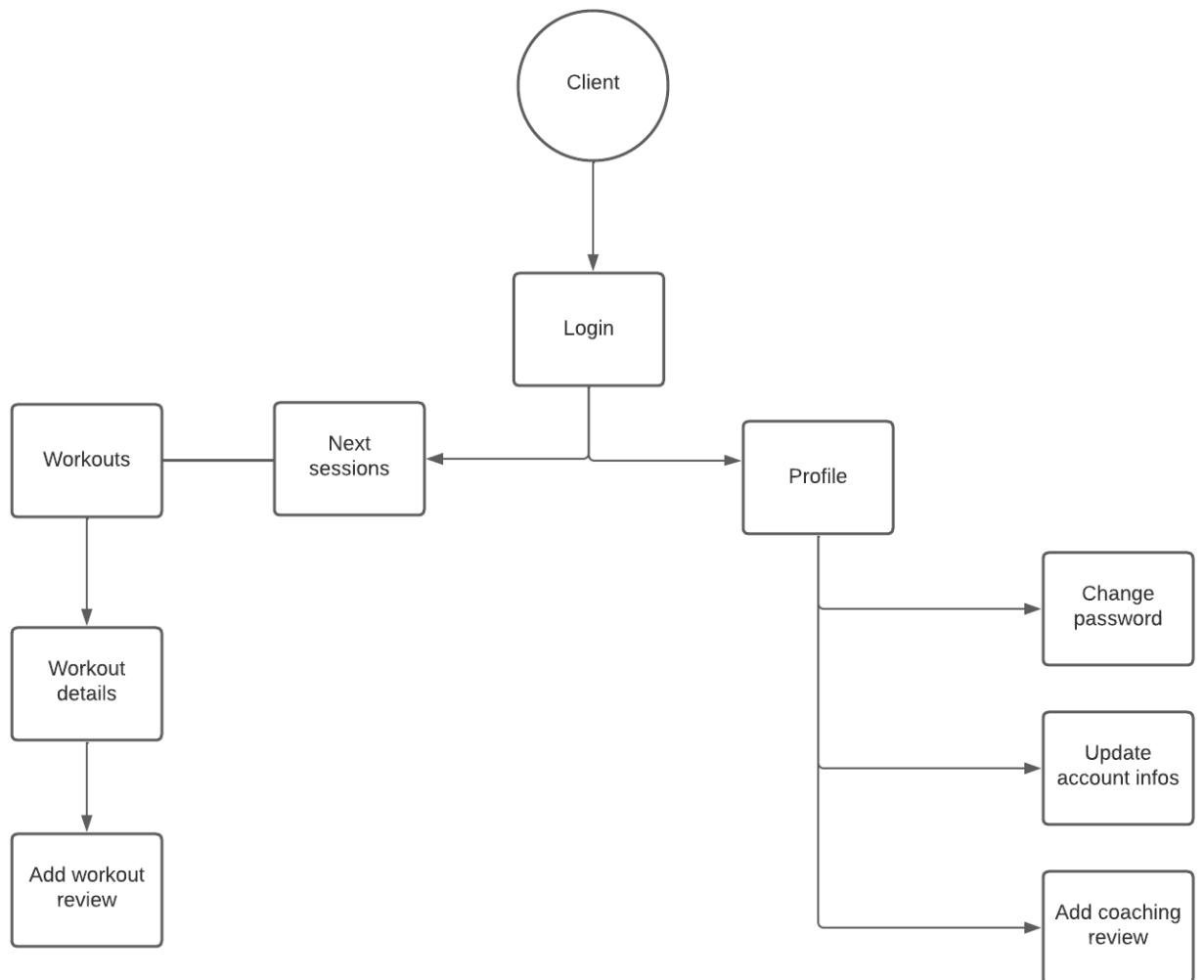
---

La sitemap de l'application possède 2 alternatives, 1 pour les clients et 1 pour les coachs.

### Sitemap Client

---

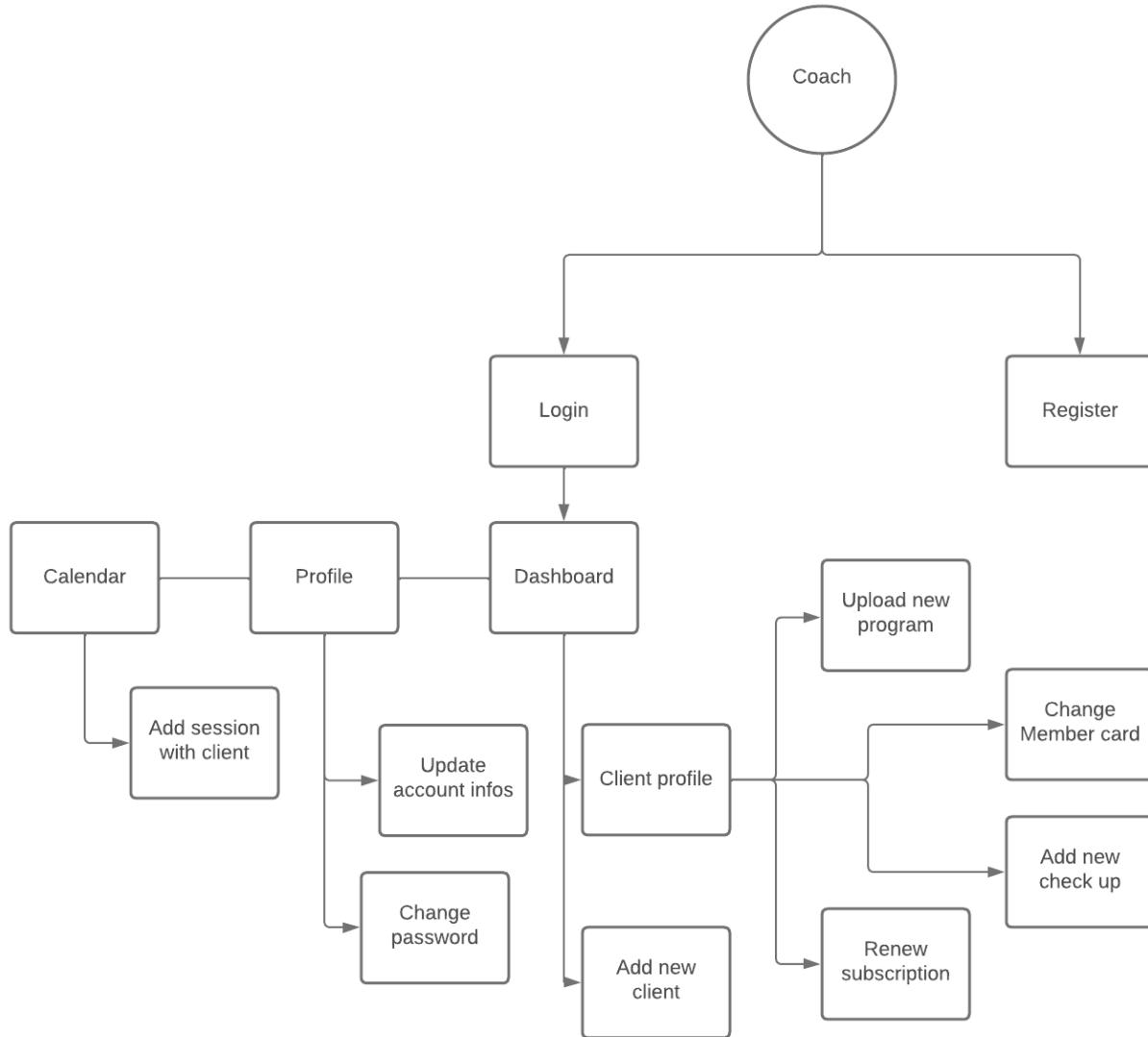
Voici la sitemap client, on y retrouve tous les chemins utilisables en tant que client :



## Sitemap Coach

---

Voici la sitemap coach, on y retrouve tous les chemins utilisables en tant que coach :



## 1.6.3 Maquettage

---

Pour préparer les interfaces, j'ai réalisé des maquettes avec l'outil Figma. Les maquettes m'ont permis de mettre à plat les éléments nécessaires pour les interfaces et ont évité de perdre trop de temps lors de la création des interfaces.

L'application FitJourney propose 2 niveaux d'accès :

- Client
- Coach

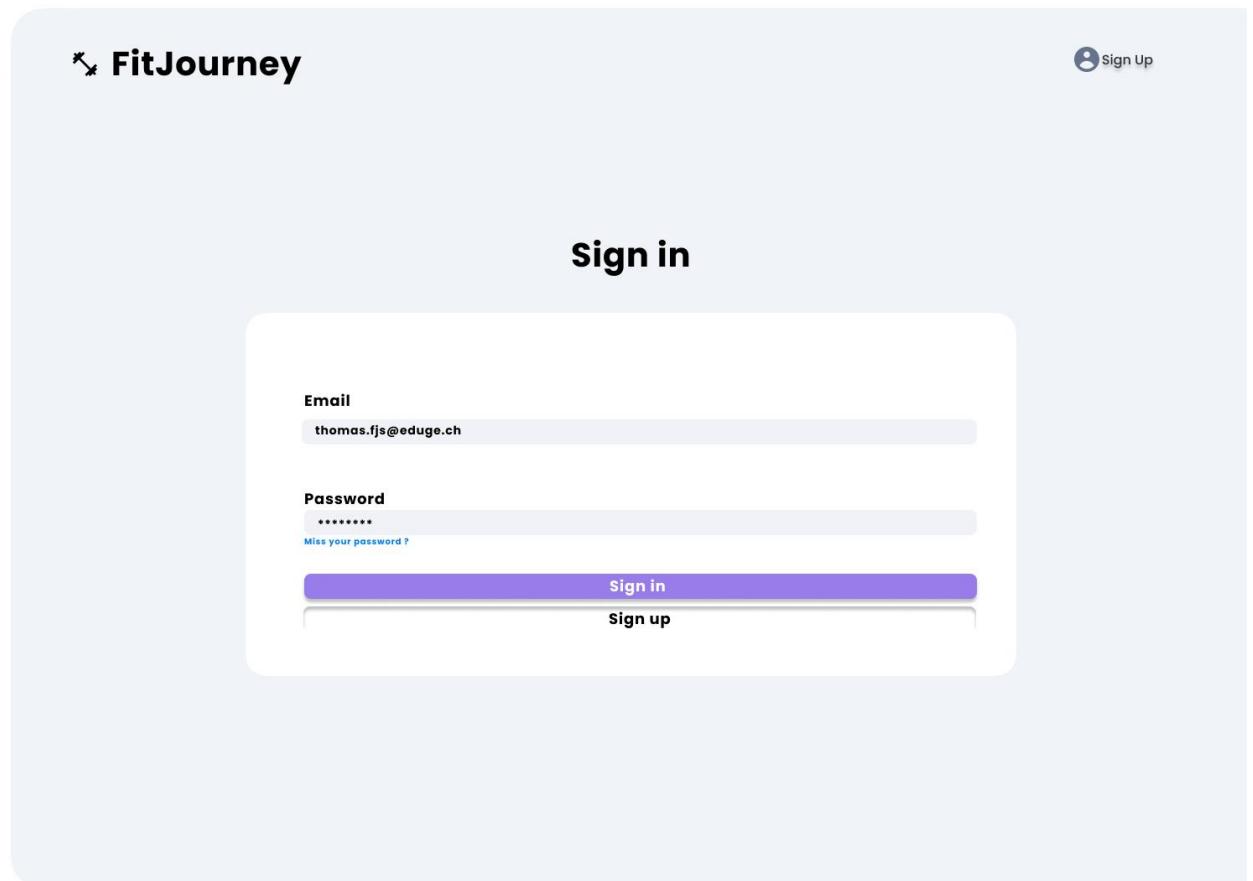
## En tant que client

---

Le client doit se connecter pour avoir accès à l'application, il ne peut pas se créer de compte tout seul.

## Page de connexion

---



La page de connexion permet aux utilisateurs de se connecter. Un lien est disponible si le mot de passe a été oublié.

## Barre de navigation

---



La barre de navigation disponible en tant que client verticalement à gauche de l'écran. 3 boutons de navigation supplémentaires sont disponibles :

- [Profil](#)
- [Agenda](#)
- [Entrainement](#)

## Page profil

**Profile**

Name: Thomas Surname: Fujise

Email: thomas.fjs@eduge.ch Date of birth: 13.04.2001

Address: Chemin des Curiaires 35 NPA: 1233 Bernex

Height: 1.85 m Weight: 99 kg

Subscription until: 21.05.22

**Reports**

Add general report

Last reports: 10.03.2022 General (10.03.2022 Session)

**Workout Activity**

This week gain: 5835 Calories burn

120 bpm Heart rate avg

08 h Time Working out

La page profil permet au client de modifier ses informations personnelles. Plusieurs boutons sont disponibles :

- 1 bouton "Update" pour appliquer les modifications effectuées sur les informations du compte
- 1 bouton "Change password" en jaune pour modifier le mot de passe
- 1 bouton "Change logo" pour importer une photo de profil.

Il a également accès à des statistiques sur les données des entraînements qu'il a effectués cette semaine (en rouge) et la liste des bilans généraux et de session que l'utilisateur a postés (en vert). Il a la possibilité d'ajouter un nouveau bilan général en cliquant sur le bouton au-dessus.

## Page Ajout Bilan

The screenshot shows the FitJourney mobile application interface. At the top, there is a navigation bar with icons for home, profile, and search. The main header reads "FitJourney". On the right side of the header, there is a user profile icon for "Thomas Fujise". Below the header, a large button labeled "New report" is visible. To the right of this button is a red-bordered box containing a user profile picture and the text "Thomas (Coach)". The main content area contains four rating scales from 1 to 10, each with a blue dot indicating the selected value. The scales are: "Satisfaction" (value 7), "Disponibility" (value 9), "Support" (value 8), and "Advice quality" (value 9). Below these scales is a large blue "Comment" input field. At the bottom of the form is a purple "ADD" button.

La page "Ajout Bilan" permet au client de noter, soit la qualité du suivi effectué par le coach, soit une session effectuée.

Dans la zone rouge, l'élément qui est évalué (un coach ou une session).

Dans la zone bleue, les différents éléments à noter ainsi qu'une zone pour ajouter un commentaire.

## Page Bilan

The screenshot shows a user interface for a coaching platform named "FitJourney". At the top, there is a navigation bar with icons for home, profile, and search. The main area displays a "Report" card for a client named "Francisco" posted on "20.04.2022". The report is categorized under "General". The card contains four satisfaction metrics: "Satisfaction : 9/10", "Support : 9/10", "Disponibility : 9/10", and "I want to continue : yes". Below these metrics is a "Comment" section containing the text: "I'm very happy with the help my coach give me, the result are coming so satifying". A red box highlights the "General" category at the top right of the report card.

La page "Bilan" permet de voir le bilan ajouté, soit un bilan sur le suivi de manière générale, soit un bilan sur une session effectuée.

Dans la zone rouge, on peut voir de quel type de bilan il s'agit (Bilan général sur le coaching ou bilan d'une session).

Dans la zone verte, on retrouve le client et la date à laquelle le bilan a été posté.

Dans la zone bleue, on retrouve les éléments qui ont été noté par le client.

## Page liste entraînements

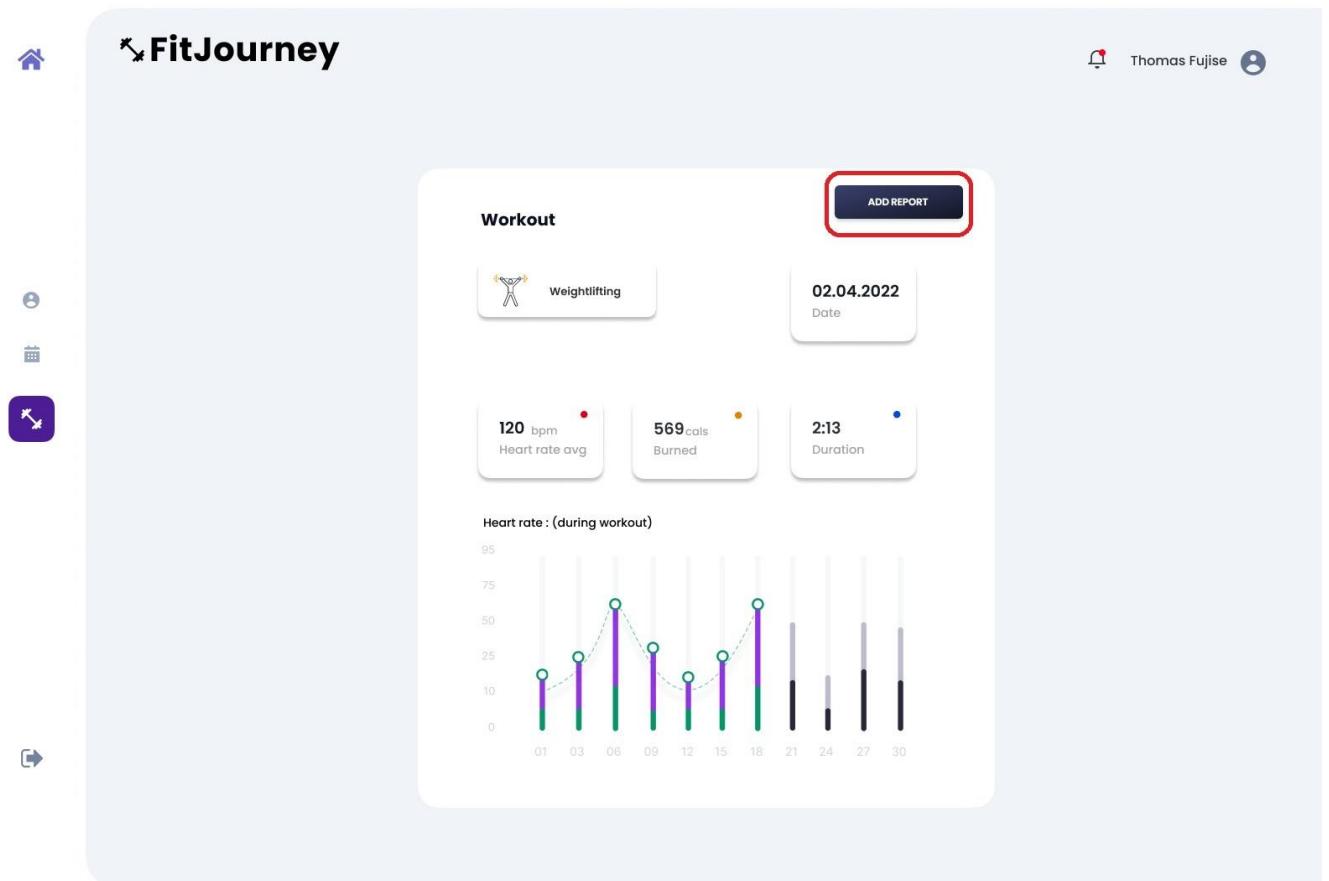
The mockup displays the 'Workouts' section of the FitJourney app. On the left, there is a vertical navigation bar with icons for Home, Profile, Workouts (highlighted in purple), and Statistics. The main area shows a table with four rows of workout data:

Workouts	Date	Duration	Calories burned	Avg Heart Rate
Weightlifting	02.04.2022	2:13	569 cal	111 bpm
Cycling	30.03.2022	0:57	489 cal	89 bpm
Weightlifting	02.04.2022	1:43	369 cal	85 bpm

Below the table, there is a large, semi-transparent button with a white arrow pointing right.

Cette page affiche la liste de tous les entraînements effectués par le client. L'utilisateur peut cliquer sur chaque élément de la liste pour avoir les détails de l'entraînement.

## Page détails entraînement



Cette page affiche les détails d'un entraînement avec les données récupérées à l'aide de la smartwatch. On peut retrouver des informations comme :

- Le nombre de calories brûlées
- Les pulsations cardiaques par minute avec un graphique montrant l'évolution durant l'entraînement
- La durée de l'entraînement
- Le type d'entraînement

Le client peut ajouter un bilan en cliquant sur le bouton (encadré en rouge) pour donner son ressenti sur la séance.

## Page prochaine session

The screenshot shows the 'FitJourney' mobile application interface. At the top, there is a navigation bar with icons for home, profile, and settings. The main content area is titled 'Next sessions:' and displays two upcoming training sessions:

Activity	Time	Date	Action
Weightlifting	14:30	08.04.2022	Open
Cycling	12:30	10.04.2022	Open

On the left side of the main content area, there are three vertical icons: a house (Home), a calendar (Schedule), and a gear (Settings). A back arrow icon is located at the bottom left of the main content area.

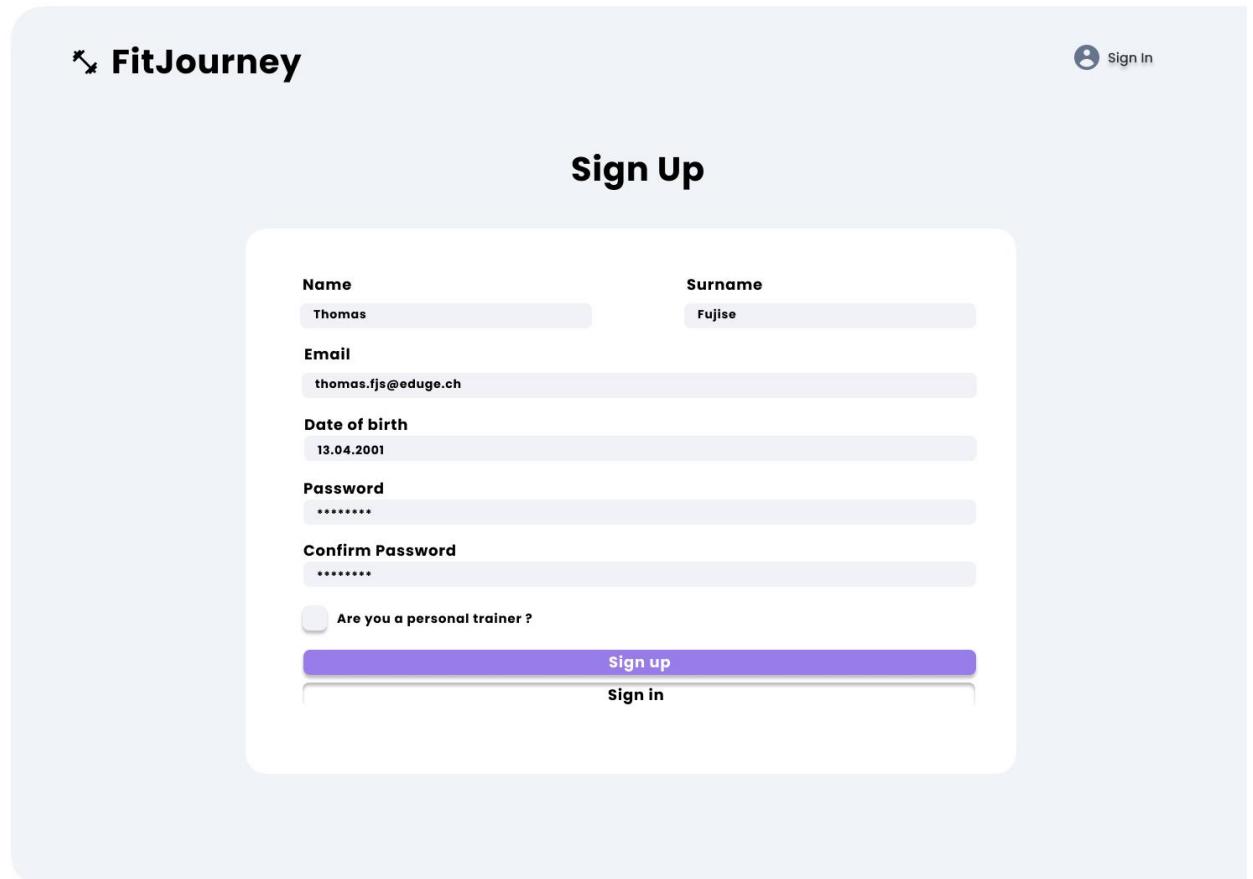
Cette page affiche les prochaines sessions d'entraînements avec un coach du client

## En tant que coach

Le coach peut se créer un compte et se connecter à l'application, une fois connecté, il a accès à toute l'application.

## Page d'enregistrement

---



La page d'enregistrement permet aux coachs de s'enregistrer, une fois enregistré il peut se connecter avec la [page de connexion](#)

## Barre de navigation

---



La barre de navigation disponible en tant que client verticalement à gauche de l'écran. 1 bouton supplémentaire est disponible :

- [Dashboard](#)

## Page tableau de bord

The screenshot shows the FitJourney dashboard interface. At the top, there's a header with a home icon, the brand name 'FitJourney' in bold, and a user profile for 'Thomas Fujise'. Below the header, there are two main sections:

- Next session:** This section displays information for a client named 'Josh'. It includes a small profile picture of Josh, his name, and a status indicator 'Valid'. To the right, it shows 'Last Workout' (04.04.2022 12:00), 'Card ID' (22721902134), and 'Active (last 24h)' with a green dot. Below this, there are three cards: 'Session time' (14:30), 'Duration' (01 h), and a small icon of a person working out.
- Clients:** This section lists clients 'Josh' and 'Francisco'. For 'Josh', it shows 'Subscription Valid', 'Last Workout' (04.04.2022 12:00), 'Card ID' (22721902134), and 'Active (last 24h)' with a green dot. For 'Francisco', it shows 'Subscription Expired', 'Last Workout' (15.03.2021 18:30), 'Card ID' (5432084337), and 'Active (last 24h)' with a red dot. At the bottom of this section is a blue button labeled 'ADD NEW CLIENT'.

La page tableau de bord permet au coach de voir la liste des clients (en rouge) dont il effectue le suivi. Le coach peut cliquer sur le nom d'un de ses clients pour afficher le profil du client concerné.

Il a également accès à un bouton pour ajouter un nouveau client (en bleu).

Les informations de la prochaine session avec un client sont disponibles au-dessus de la liste des clients (en vert).

## Page Ajout de client

The screenshot shows the FitJourney application interface. At the top, there is a navigation bar with icons for home, profile, and search. The main title 'FitJourney' is displayed. On the right side of the header, there is a user profile for 'Thomas Fujise' with a small photo.

The central part of the screen is a form titled 'Add new client'. It contains the following fields:

- Name: Thomas (highlighted with a green border)
- Surname: Fujise
- Email: thomas.fjs@eduge.ch
- Date of birth: 13.04.2001
- Height: 1.85 m
- Weight: 99 kg
- Subscription type: 1 Month (highlighted with a red border)
- Starting date: 04.04.2022
- Subscription until: 04.05.22

At the bottom of the form is a blue 'Add' button.

La page ajout de client permet au coach d'effectuer la prise en charge d'un nouveau client. Si le client possède déjà un compte, le coach peut sélectionner un compte déjà existant (en vert) ou créer un nouveau compte client.

Si le coach sélectionne un compte déjà existant, les champs se remplissent automatiquement. Il faudra uniquement sélectionner le type d'abonnement souhaité (en rouge)

## Page profil client

The page profil client permet au coach de visionner le profil de ses clients (en bleu). Plusieurs boutons sont disponibles :

- 1 bouton pour le changement de la carte de membre (Bouton bleu)
- 1 bouton pour le renouvellement de l'abonnement (Bouton vert)
- 1 bouton pour l'annulation de l'abonnement (Bouton rouge)

Il peut également voir les graphiques/statistiques disponibles sur le profil.

Le coach peut ajouter les nouveaux programmes du client (en rouge) à l'aide des boutons d'importation (1 bouton pour le programme d'entraînement et 1 bouton pour le programme de nutrition). Les anciens programmes sont disponibles en cliquant sur les boutons du programme souhaité (bouton vert et orange), leurs dates d'importation sont affichées à côté.

Dans la zone verte, on peut retrouver les différents bilans de satisfaction que le client a ajoutés. En haut de cette zone, un bouton d'ajout de bilan est disponible. Il va permettre d'effectuer le bilan du client et d'ajouter les nouvelles informations (Poids, Masse grasseuse, etc)

## Page Bilan client

The mockup shows a mobile application interface for FitJourney. At the top, there's a navigation bar with icons for home, profile, calendar, and search. The main header is "FitJourney". On the right, there's a user profile for "Thomas Fujise" with a photo.

**New report**

Name	Surname	Height	Age
Thomas	Fujise	1.85 m	21

Weight 98.7 kg	BMI 28.1	Water % 59.9 %	Protein % 12.7 %
Muscle Mass % 67.3 %	Body Fat % 27.1 %	Bone Mass % 3.6 %	BMR 2045 kcal
Muscle Mass 66.4 kg	Body Fat 26.1 kg	Visceral Fat 10.0 kg	Bone Mass 3.5 kg
Lean Body Mass 72.0 kg			

Photos

Front      Right side      Left side      Back

**ADD**

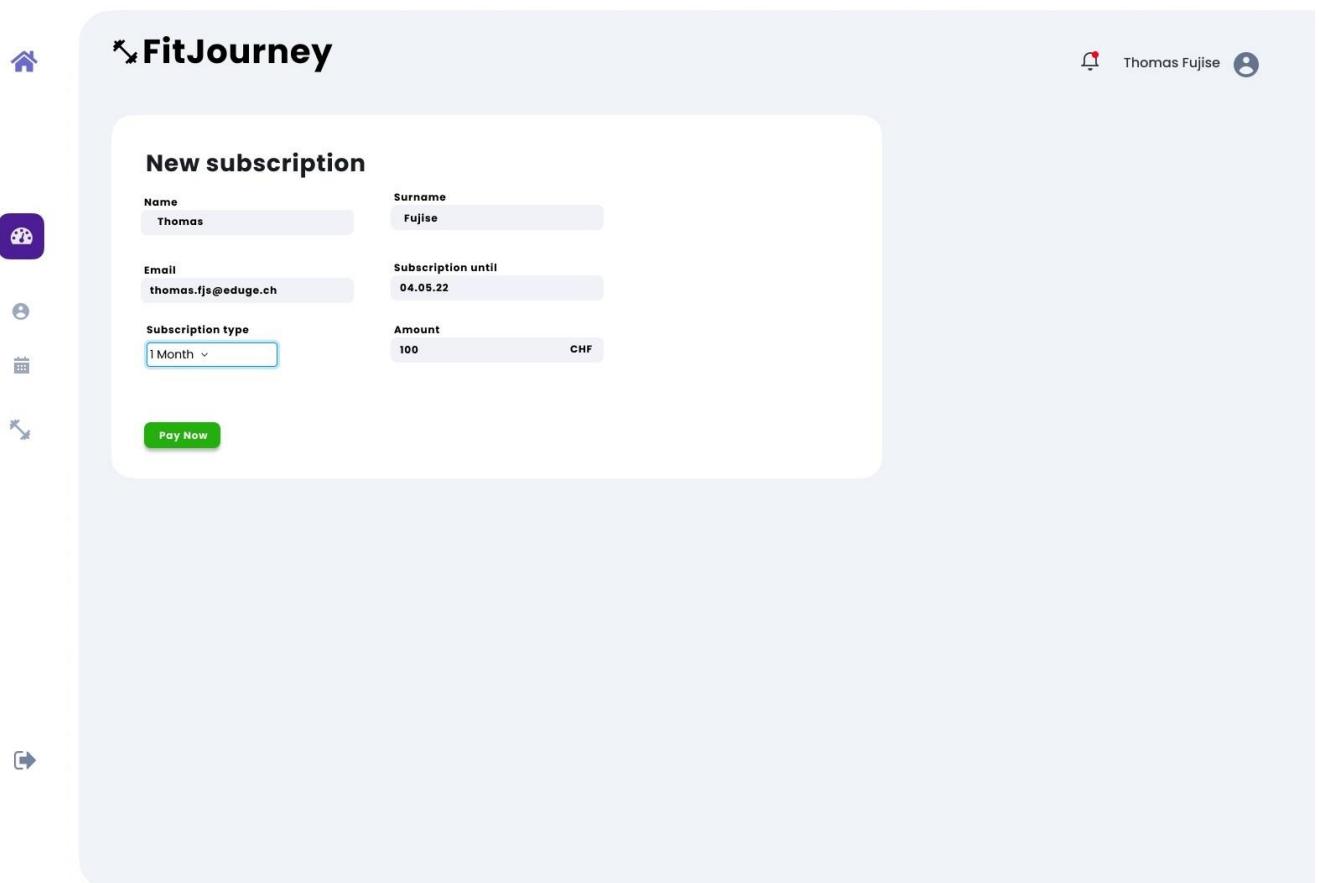
La page Bilan client permet au coach d'ajouter le bilan d'un client.

Dans la zone rouge, on retrouve les infos qui changent le moins, voire jamais (Nom, taille ou encore l'âge).

Dans la zone bleue, les informations importantes qui ont pu être récupérées suite à la pesée.

Dans la zone verte, il est possible d'ajouter des photos du physique du client pour pouvoir éventuellement avoir des avant/après en guise de comparaison.

## Page Abonnement



La page abonnement permet de valider le paiement d'un client (Aucune transaction n'est effectué par l'application). Le coach peut sélectionner le type d'abonnement que le client souhaite prendre, la date d'échéance du nouvel abonnement sélectionné ainsi que son coût seront affichés.

## Page Calendrier

The screenshot shows the FitJourney application's calendar interface. At the top, the 'FitJourney' logo is displayed next to a user profile for 'Thomas Fujise'. On the left side, there are three main navigation icons: a house icon for the home screen, a calendar icon for the current page, and a magnifying glass icon for search. The central part of the screen features a monthly calendar for March 2022, with the specific date '04/03/2022' highlighted. To the right of the calendar, a red-bordered box titled 'Sessions:' lists two sessions: one for 'Francisco' at 14:30 and another for 'Daniel' at 16:00. Each session entry includes a small profile picture, the client's name, the start time, the duration, and a small icon representing the workout type. Below this section, a green-bordered box contains a form for creating a new session. The form includes a dropdown menu for selecting a client ('David'), a time selector (showing '10:00'), and a dropdown menu for the session type ('Weightlifting'). A large blue button labeled 'ADD NEW SESSION' is positioned at the bottom of this form.

La page calendrier permet au coach d'avoir accès à un calendrier et de visionner les rendez-vous enregistrés à la date sélectionnée. Les sessions enregistrées sont affichées dans la zone rouge, avec quelques détails sur la séance.

Dans la zone verte, un bouton pour ajouter une nouvelle session avec un client est disponible. Le coach doit renseigner :

- Le nom du client (Liste déroulante parmi ses clients)
- L'heure de la session
- La durée
- Le type de session

## 1.7 Analyse Organique

---

### 1.7.1 Mise en place / Environnement

---

#### Visual Studio Code

---

J'ai choisi d'utiliser Visual Studio code pour éditer mon code, il est directement relié à mon repos sur Github. Je peux donc directement depuis Visual Studio Code commit tous les changements que j'effectue.

#### Mkdocs

---

Mkdocs permet de générer un site statique pour la documentation. Il prend en compte tous les fichiers Markdown (.md) dans le dossier `docs/`, et un fichier de configuration YAML qui se trouve à la racine du projet. Mkdocs me permet de visionner mes fichiers Markdown en direct et à l'aide de l'extension `with-pdf`, de générer un fichier PDF de tous mes fichiers Markdown.

#### GitHub

---

Pour pouvoir garder un suivi constant de mon projet, j'ai choisi d'utiliser GitHub comme outil de contrôle de version.

Voici comment est structuré le github :

```

FitJourney
├── docs
├── src
├── mkdocs.yml
├── README.md
└── .gitignore

```



- Le dossier `docs/` contient les fichiers de documentation et de journal de bord
- Le dossier `src/` contient tout le code source de l'application

#### Trello

---

Trello est un outil de gestion de projet en ligne, inspiré par la méthode Kanban. Il repose sur une organisation des projets en planches listant des cartes, chacune représentant des tâches.



Afin de créer une roadmap pour mon projet, j'ai utilisé Trello pour lister les différentes étapes de mon projet. J'ai ensuite pu définir les échéances pour chaque tâche/étapes avec mon planning prévisionnel.

J'ai créé 5 colonnes :

- Backlog (Liste de toutes les tâches)
- To-Do (Les tâches qui ont été validées et qui sont à faire)
- Doing (Les tâches en cours)
- Testing (Les tâches en cours de test)
- Done (Les tâches terminées)

## Guide d'installation

---

FitJourney utilise la 3<sup>e</sup> version de Python (donc `python3`)

### Windows

---

Premièrement, cela serait intéressant de travailler sur un environnement WSL avec VSCode. Pour mettre en place WSL, j'ai suivi le guide disponible directement dans la documentation de VSCode : <https://code.visualstudio.com/docs/remote/wsl>

### MySQL

---

Une fois l'environnement de développement installé correctement, il faut installer MySQL (si cela n'est pas déjà fait) à l'aide de la commande :

```
sudo apt install mysql-server
```

Puis lancer le service MySQL:

```
sudo service mysql start
```

et

```
sudo mysql_secure_installation
```

Pour se connecter à la base de données, utiliser un DBMS comme *DBeaver* ou *MySQLWorkbench*.

## pip3

---

Il faut s'assurer que pip3 a bien été installé afin de télécharger tous les paquets/librairies pour faire fonctionner l'application comme il faut.

```
sudo apt install python3-pip
```

Si une erreur survient, utiliser la commande :

```
sudo apt update && sudo apt upgrade
```

Sinon, vous pouvez vérifier que pip3 a bien été installé en utilisant :

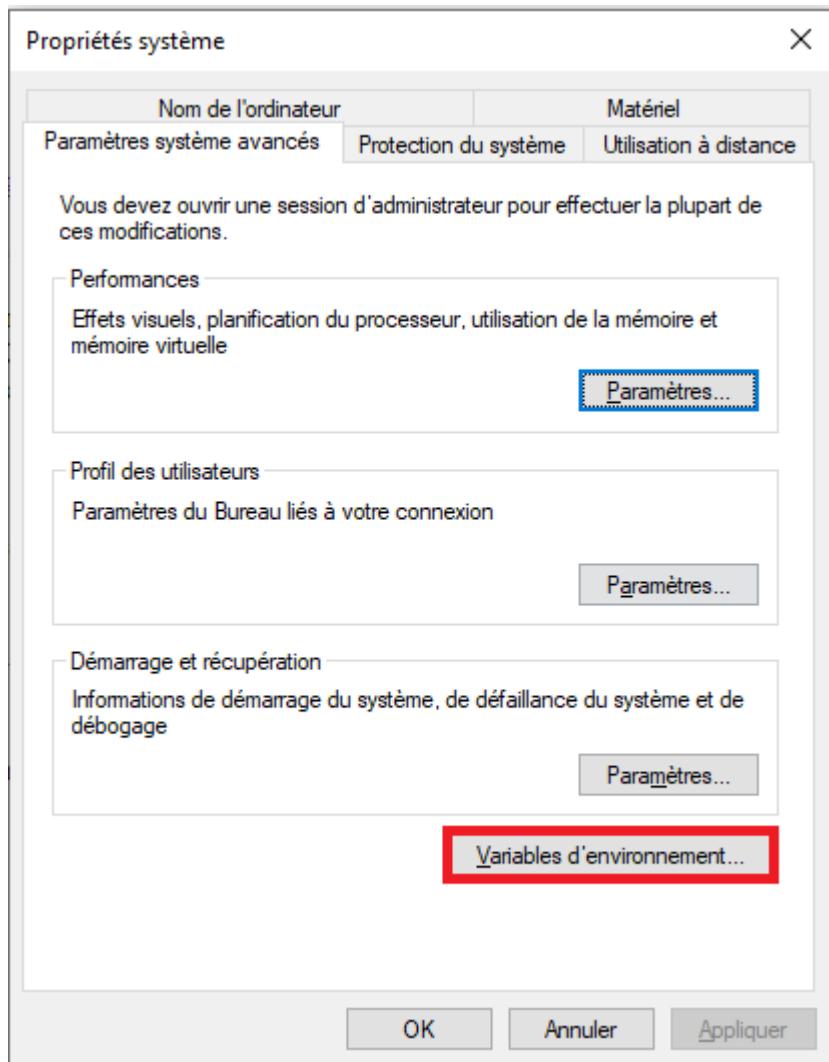
```
pip3 --version
```

## Pyscard

---

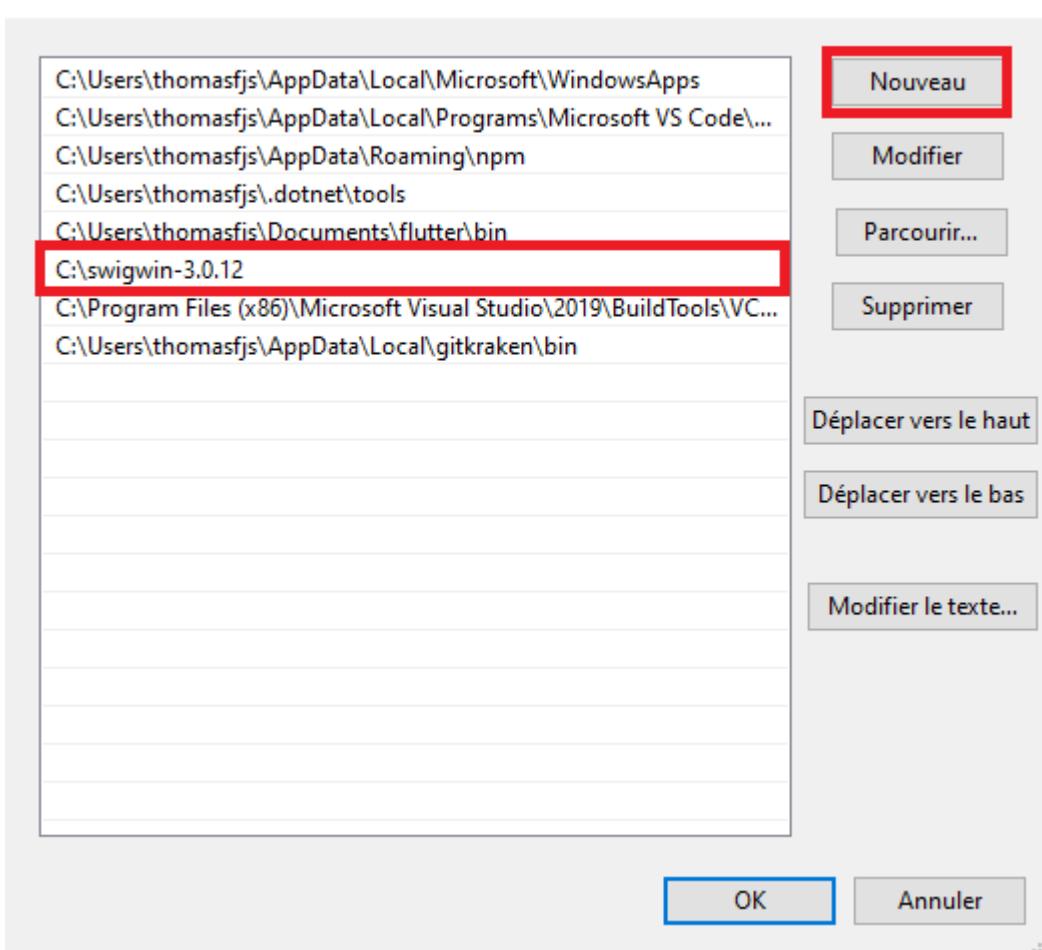
Pour pouvoir installer la librairie `pyscard` sur Windows, il faut au préalable installer **SWIG** depuis le lien suivant : <https://www.swig.org/download.html>

Il faut ensuite l'ajouter directement au PATH en modifiant les variables d'environnement dans les propriétés système.



On peut ensuite ajouter simplement le chemin vers le dossier `swig` que l'on a téléchargé :

### Modifier la variable d'environnement



Une fois ajouté, il faut ensuite installer Visual C++ version 14.0 ou plus récente (directement installable depuis le Visual Studio Installer) (<https://visualstudio.microsoft.com/fr/downloads/>)

## Requirements

Une fois que `swig` a été ajouté au PATH et que `pip3` a été installé, il faut installer tous les paquets nécessaires pour l'application. Un fichier `requirements.txt` contenant toutes les librairies nécessaires est disponible dans le dossier `src/`

```
pip3 install -r requirements.txt
```

!\\ Ne pas toucher ce fichier.

## API

L'API utilisée est l'API Accesslink v3.144.0 de Polar <https://www.polar.com/accesslink-api/#polar-accesslink-api>.

Pour pouvoir utiliser l'API, il faut disposer au préalable d'un compte Polar Flow <https://flow.polar.com/>.

## Création d'un nouveau client pour l'API

---

Aller sur <https://admin.polaraccesslink.com/>. Il faut se connecter avec votre compte Polar Flow et ajouter un nouveau client. Il faut utiliser `http://localhost:5000/oauth2_callback` pour la callback d'autorisation.

## Configuration des identifiants

---

Ajouter l'`id` et le `secret` client dans le fichier `src/cardsChecker/config.yml` comme ceci :

```
client_id: 57a715f8-b7e8-11e7-abc4-cec278b6b50a
client_secret: 62c54f4a-b7e8-11e7-abc4-cec278b6b50a
```

## Authentification

---

Le compte utilisateur doit être relié au client créé avant de pouvoir accéder aux données. Pour le relier, il faut lancer le script `src/cardsChecker/authorization.py`

```
python3 authorization.py
```

puis aller sur la page '[https://flow.polar.com/oauth2/authorization?response\\_type=code&client\\_id=CLIENT\\_ID](https://flow.polar.com/oauth2/authorization?response_type=code&client_id=CLIENT_ID)' pour relier votre compte utilisateur. ('`CLIENT_ID`' doit être remplacé par votre id client)

Une fois ces étapes effectuées, votre compte aura accès aux données Polar.

## Base de données

---

Pour la base de données, il suffit de créer une base nommée `fitjourney`, (respecter le nom sinon l'application ne fonctionnera pas). Vous pouvez ensuite exécuter le script sql disponible à `src/apps/db.sql`.

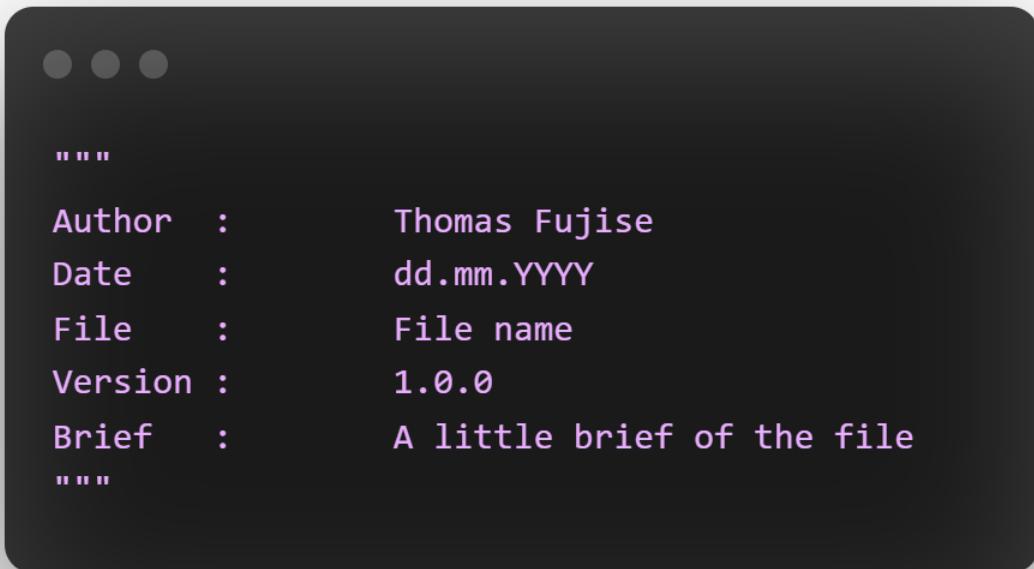
## 1.7.2 Organisation / Gestion du temps

---

### Convention : en-têtes

---

Tous les fichiers développés par moi-même possèdent l'entête ci-dessous :



## Backlog

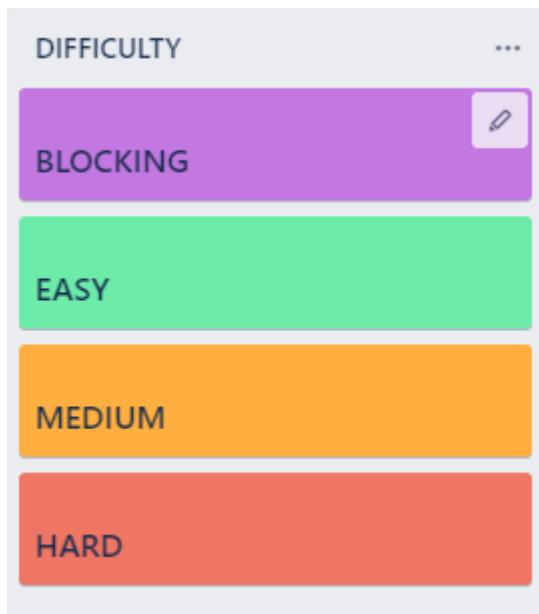
---

Les backlogs ont été identifiés au lancement du projet (certains l'ont été au fur et à mesure que le projet avançait). Ils ont été ajoutés au [Trello](#) qui est l'outil qui a été choisi pour gérer le projet. Avec les cartes Trello, j'ai défini la difficulté et la priorité des tâches. Il était donc plus simple pour moi de visualiser quelles tâches devaient être réalisé en priorités.

## Difficulté

---

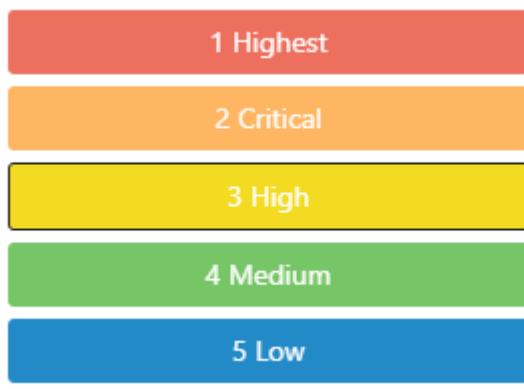
La difficulté est définie par une étiquette de couleur selon la charte que j'ai définie :



## Priorité

---

La priorité est définie également par une étiquette qui provient de l'extension "CardsPriority" sur Trello. Les priorités assignées suivent la charte :



## Planning

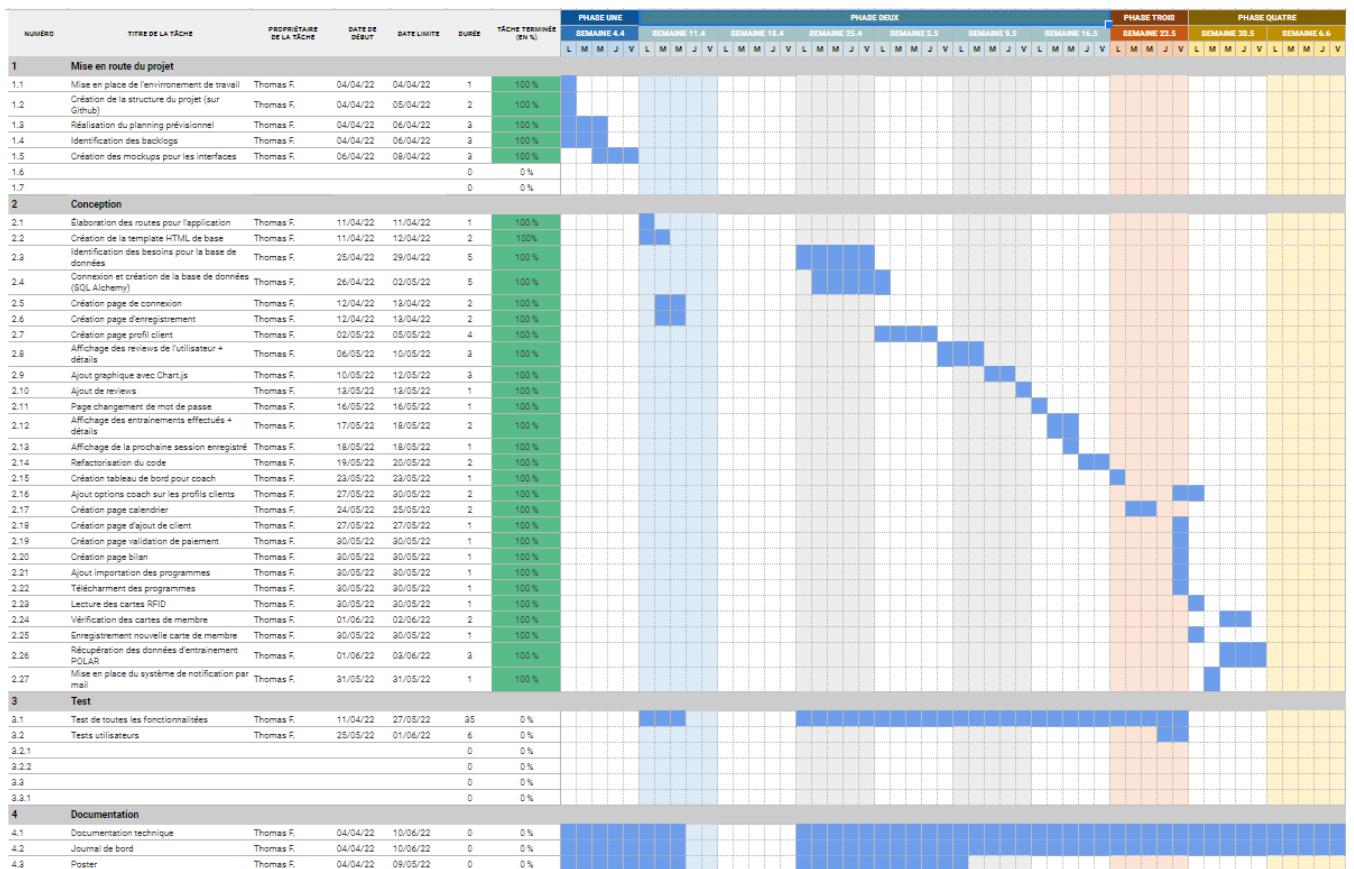
---

Un planning prévisionnel a été réalisé au début du projet afin de lister les tâches et d'estimer leurs durées. En regardant les graphiques, on peut directement apercevoir que les deux n'ont aucun point en commun.

## Planning prévisionnel

NUMÉRO	TITRE DE LA TÂCHE	PROPRIÉTAIRE DE LA TÂCHE	DATE DE DÉBUT	DATE LIMITÉE	DURÉE	PHASE UN		PHASE DEUX						PHASE TROIS		PHASE QUATRE												
						SEMAINE 4.4		SEMAINE 11.4			SEMAINE 18.4			SEMAINE 25.4		SEMAINE 2.5		SEMAINE 9.5		SEMAINE 16.5		SEMAINE 23.5		SEMAINE 30.5		SEMAINE 6.6		
			L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	
<b>1</b>	<b>Mise en route du projet</b>																											
1.1	Mise en place de l'environnement de travail	Thomas F.	04/04/22	04/04/22	1																							
1.2	Création de la structure du projet (sur GitHub)	Thomas F.	04/04/22	05/04/22	2																							
1.3	Réalisation du planning prévisionnel	Thomas F.	04/04/22	06/04/22	2																							
1.4	Identification des backloggs	Thomas F.	04/04/22	06/04/22	2																							
1.5	Création des mockups pour les interfaces	Thomas F.	06/04/22	08/04/22	2																							
1.6					0																							
1.7					0																							
<b>2</b>	<b>Conception</b>																											
2.1	Élaboration des routes pour l'application	Thomas F.	11/04/22	11/04/22	1																							
2.2	Création de la template HTML de base	Thomas F.	11/04/22	12/04/22	2																							
2.3	Identification des besoins pour la base de données (MySQL)	Thomas F.	12/04/22	13/04/22	2																							
2.4	Connexion et création de la base de données (SQL Alchemy)	Thomas F.	25/04/22	26/04/22	2																							
2.5	Création page de connexion	Thomas F.	27/04/22	27/04/22	1																							
2.6	Création page d'enregistrement	Thomas F.	28/04/22	28/04/22	1																							
2.7	Création page profil client	Thomas F.	29/04/22	02/05/22	2																							
2.8	Ajout graphique avec Chart.js	Thomas F.	03/05/22	04/05/22	2																							
2.9	Création tableau de bord pour coach	Thomas F.	05/05/22	05/05/22	1																							
2.10	Ajout option coach sur les profils clients	Thomas F.	05/05/22	05/05/22	1																							
2.11	Ajout importation des programmes	Thomas F.	06/05/22	11/05/22	4																							
2.12	Création page affichage programmes	Thomas F.	12/05/22	13/05/22	2																							
2.13	Lecture des cartes RFID	Thomas F.	13/05/22	16/05/22	2																							
2.14	Vérification des cartes de membre	Thomas F.	16/05/22	16/05/22	1																							
2.15	Enregistrement nouvelle carte de membre	Thomas F.	17/05/22	18/05/22	2																							
2.16	Réajustement des données d'entraînement POLAR	Thomas F.	18/05/22	23/05/22	4																							
2.17	Mise en place du système de notification par mail	Thomas F.	24/05/22	26/05/22	2																							
<b>3</b>	<b>Test</b>																											
3.1	Test de toutes les fonctionnalités	Thomas F.	11/04/22	27/05/22	35																							
3.2	Tests utilisateurs	Thomas F.	25/05/22	01/06/22	6																							
3.2.1					0																							
3.2.2					0																							
3.3					0																							
3.3.1					0																							
<b>4</b>	<b>Documentation</b>																											
4.1	Documentation technique	Thomas F.	04/04/22	10/06/22	0																							
4.2	Journal de bord	Thomas F.	04/04/22	10/06/22	0																							
4.3	Poster	Thomas F.	04/04/22	09/05/22	0																							

## Planning effectif



Lors de la réalisation du projet, beaucoup de tâches sont venues s'ajoutées. Toute la mise en place de la structure de l'application avec Python Flask m'a prise beaucoup plus de temps que prévu. Les besoins de la base de données qui ont été revu à maintes reprises.

Le planning initial n'a vraiment pas pu être respecté à cause des différentes tâches qui ont été identifiées uniquement après avoir commencé le projet.

## 1.7.3 Technologies utilisées

### NFC Reader ACR122U

Le lecteur NFC ACR122U est un appareil permettant de lire et d'écrire sur des cartes sans contact. Il est basé sur la technologie Mifare 13,56 MHz (RFID) et suit les standards de la norme ISO 18092. L'ACR122U est développé et vendu par ACS Ltd.

Pour pouvoir utiliser ce lecteur, j'ai utilisé la librairie *pyscard* qui me permet de récupérer les infos des cartes à puces que le lecteur lit



## RFID

---

### Qu'est-ce que la technologie RFID ??

RFID (Radio Frequency Identification), est une technologie qui permet d'enregistrer des données sur un support et de les récupérer à distance. Elle est apparue dans les années 1940 et était utilisée uniquement par l'armée pour l'identification des avions de guerre qui entraient dans l'espace aérien du Royaume-Uni. Elle s'est ensuite répandue dans différents secteurs industriels à partir des années 1980.

### Comment fonctionne la RFID ?

Les étiquettes RFID sont composées d'une puce RFID et d'une antenne et sont collées sur un produit. Elles enregistrent les données et avec un lecteur électromagnétique on peut ensuite lire les ondes radio présentes sur la puce RFID grâce à l'antenne.

### Pourquoi utiliser RFID

RFID est un système de traçabilité. À l'aide d'une seule puce, il est possible de tracer les produits pendant tout le processus de production, de transport et de distribution, voire jusqu'à leur fin de vie.

Dans le cadre de mon projet, RFID est une solution adéquate pour gérer les entrées/sorties des clients dans la salle d'entraînement.

### Différences entre RFID et NFC



Dans le cadre de mon projet, j'utilise parfois le terme NFC. La technologie NFC (Near Field Communication) est un dérivé de la RFID qui a commencé à être utilisé dès 2011. Le NFC repose sur le même concept que la RFID. C'est une technologie qui fonctionne avec une puce permettant d'échanger des données entre un lecteur et n'importe quel terminal avec un simple rapprochement ou contact entre les deux objets.

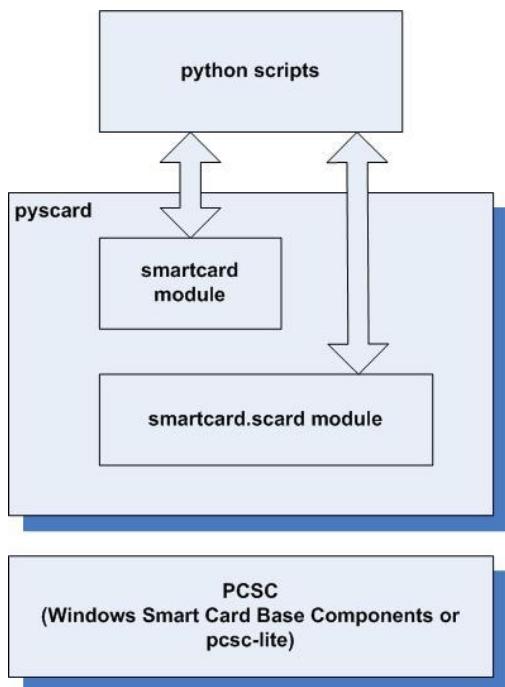
La communication sans fil ne fonctionne qu'à courte portée et haute fréquence, une distance d'environ 10 cm. La technologie NFC se retrouve dans la plupart des smartphones, consoles de jeux ou cartes bancaires. Le lecteur de carte que j'utilise fonctionne à l'aide de cette technologie également.

Les principales différences entre RFID et NFC résident dans la portée plus courte et sécurisée pour NFC (10 cm) contre jusqu'à 10 m pour RFID. La technologie NFC peut transmettre toute sorte de données contre RFID qui ne transmet que l'ID. La lecture fait, quant à elle, défaut à la technologie NFC qui ne peut lire qu'une puce à la fois, ce qui peut limiter ses cas d'utilisation.

## Pyscard - Librairie Python Smart card

---

Pyscard est un module python qui permet d'utiliser les cartes à puce (PC/SC) avec python. Il donne accès à plusieurs classes et fonctions donnant accès aux cartes et aux lecteurs.



Architecture pyscard :

- smartcard.scard est un module d'extension enveloppant l'API WinSCard (Les composants de base smartcard) aussi connue sous le nom PC/SC (Personal computer / Smart Card)
- smartcard est un framework Python construit à partir de l'API PC/SC

## SWIG

---



SWIG est un outil logiciel open source qui permet de connecter des logiciels ou librairies écrites en C/C++ avec des langages de scripts tels que : *Perl, Python, Ruby, PHP* ou d'autres langages de programmation comme *Java* ou *C#*.

## Polar Accesslink API

---



Accesslink est une API qui donne accès aux données d'entraînement et d'activité journalière enregistrés par les appareils Polar. Pour pouvoir l'utiliser, il est nécessaire de posséder un compte Polar Flow afin de créer un client sur [admin.polaraccesslink.com](https://admin.polaraccesslink.com) qui nous donnera accès à l'API.

Accesslink utilise [OAuth2](#) comme protocole d'authentification. Les utilisateurs enregistrés ont besoin de s'authentifier pour pouvoir avoir accès aux données.

Fonctionnalités de base d'Accesslink :

<b>Fonctionnalité</b>	<b>Description</b>
Utilisateurs	Permet d'enregistrer, supprimer et récupérer les informations de base de l'utilisateur
Pull Notification	Permet de vérifier si l'utilisateur a des données disponibles à récupérer
Donnée d'entraînement	Permet d'accéder aux données d'entraînements de l'utilisateur
Activité journalière	Permet d'accéder aux données de l'activité journalière de l'utilisateur
Info physique	Permet d'accéder aux informations physiques de l'utilisateur (Ex : Taille/Poids)
Modèle de données	Décrit tous les objets qui transportent les données entre le serveur et le client
Annexes	Contient des exemples et des détails sur l'interface de l'application

## Choix dans le projet - FitJourney

---

Je souhaitais utiliser les données d'entraînements en provenance de montre connectée et j'avais déjà en ma possession une montre Polar. Mon choix s'est donc naturellement orienté vers Polar, lorsque j'ai pris connaissance de l'API développée en Python qu'ils mettaient à disposition.

## Python

---

Python est un langage très flexible qui propose une approche modulaire et orienté objet de la programmation. Il est placé sous une licence libre et fonctionne sur la plupart des plates-formes informatiques. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.



## Choix dans le projet - FitJourney

---

Durant ma dernière année en tant que Technicien ES, j'ai beaucoup utilisé python dans mes projets. En trouvant de plus en plus d'outils comme la librairie *pyscard* ou encore l'API Polar en Python, je me suis très vite décidé sur le langage que j'allais utiliser pour ce projet.

## Python Flask (backend)

---

Flask est un micro-framework Python qui permet la création d'applications web évolutives. Flask dépend de la boîte à outils WSGI (Web Server Gateway Interface) de [Werkzeug](#) et du moteur de templates [Jinja](#). Le dossier `app/` représente une application Flask, elle est entre-autre homogène à une fonction WSGI.



# Flask

web development,  
one drop at a time

## Utilisation Flask

---

Pour lancer une application Flask, il faut utiliser la méthode de l'objet Flask :

```
.run()
```

ou lancer directement l'application à l'aide de la commande :

```
Flask run
```

## Micro framework

---

Un micro framework est un framework qui tente de fournir uniquement les composants absolument nécessaires à un développeur pour la création d'une application. Par exemple, dans le cas d'une application Web, un micro framework peut être spécifiquement conçu pour la construction d'API pour un autre service/application.

Le terme *micro* dans le micro framework signifie que Flask vise à garder le code de base simple, mais extensible. Flask ne prendra pas beaucoup de décisions, par exemple quelle base de données utiliser. Les décisions qu'il prend, telles que le moteur de templates à utiliser, sont faciles à modifier. Tout le reste est libre, de sorte que Flask puisse répondre à tous nos besoins et à tous ce que vous ne voulez pas en même temps.

En définissant uniquement le moteur de templates et un système de routes, Flask laisse le choix de personnaliser (en ajoutant des packages) pour la gestion des formulaires par exemple.

## Choix dans le projet FitJourney

---

Dans le cadre de ce projet, j'ai préféré utiliser Flask comme framework à la place d'un autre, car j'utilise l'API Polar Accesslink qui est fait en Python. Je souhaitais garder le même langage pour éviter de partir dans tous les sens.

## Architecture (Blueprint)

---

Afin de bien structurer mon projet, j'ai décidé d'utiliser les Flask Blueprint. Chaque Flask Blueprint est un objet et fonctionne de manière très similaire à une application Flask. Ils peuvent tous les deux avoir des ressources, comme des fichiers statiques, des templates et des vues qui sont associées aux routes.

Malgré tout, un Flask Blueprint n'est pas exactement comme une application Flask car il a besoin d'être enregistré dans l'application pour être lancé. Lorsqu'on enregistre un Blueprint à l'application, on étend l'application avec le contenu du Blueprint. Les Blueprints enregistrent toutes les opérations à exécuter et ne les exécutent qu'une fois enregistré dans l'application

Les Blueprints m'ont permis de découper l'application principale en plusieurs parties et de structurer mon projet.

## Argon Design System

---

Argon est un "design system" open source basé sur le framework CSS Bootstrap 4. Il propose plus de 100 composants individuels ce qui permet une certaine liberté.



Argon m'a permis de ne pas perdre trop de temps sur le côté "design" de l'application

## Chart.JS

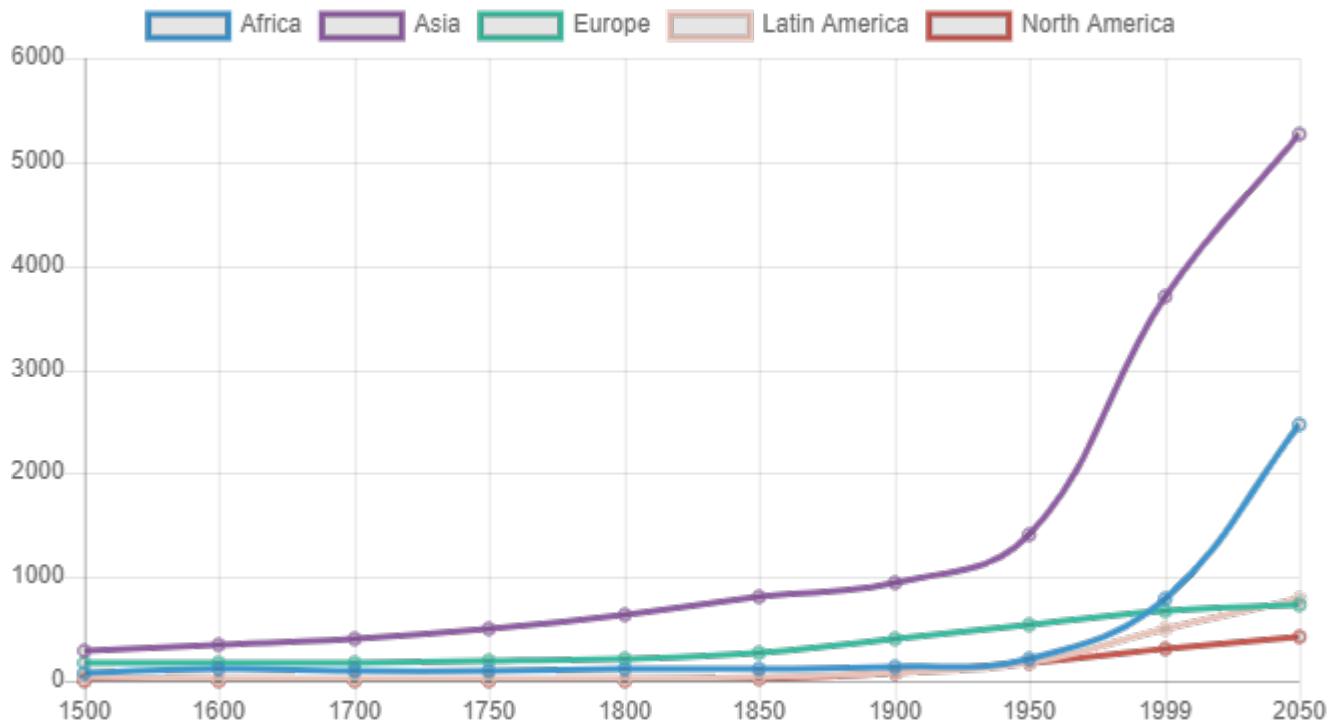
---

Chart.js est une librairie open source, elle permet la visualisation de données en utilisant JavaScript. Elle est similaire à *Chartist* ou *Google Chart*. Chart.js supporte 8 différents types de graphique et sont tous responsive. Pour pouvoir utiliser Chart.js il faut :

- Définir ou dessiner le graphique sur notre page
- Définir quel type de graphique afficher
- Définir les données, labels et toutes les autres options



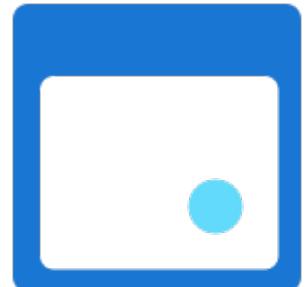
Voici un exemple de graphique que peut générer chart.js :



## FullCalendar.io

---

FullCalendar est une librairie JavaScript qui s'intègre facilement avec des frameworks JavaScript populaires tels que Vue, React et Angular. Elle permet d'implémenter un calendrier avec des événements. Cette librairie est très utile pour gérer l'agenda du coach.



**Demos**

Drag-n-Drop Events

Drag these onto the calendar:

- Event 1
- Event 2

More info »

Resource Timeline >

Resource Time Grid >

Selectable Dates >

Background Events >

Theming >

Locales >

Time Zones >

Jul 25 – 31, 2021

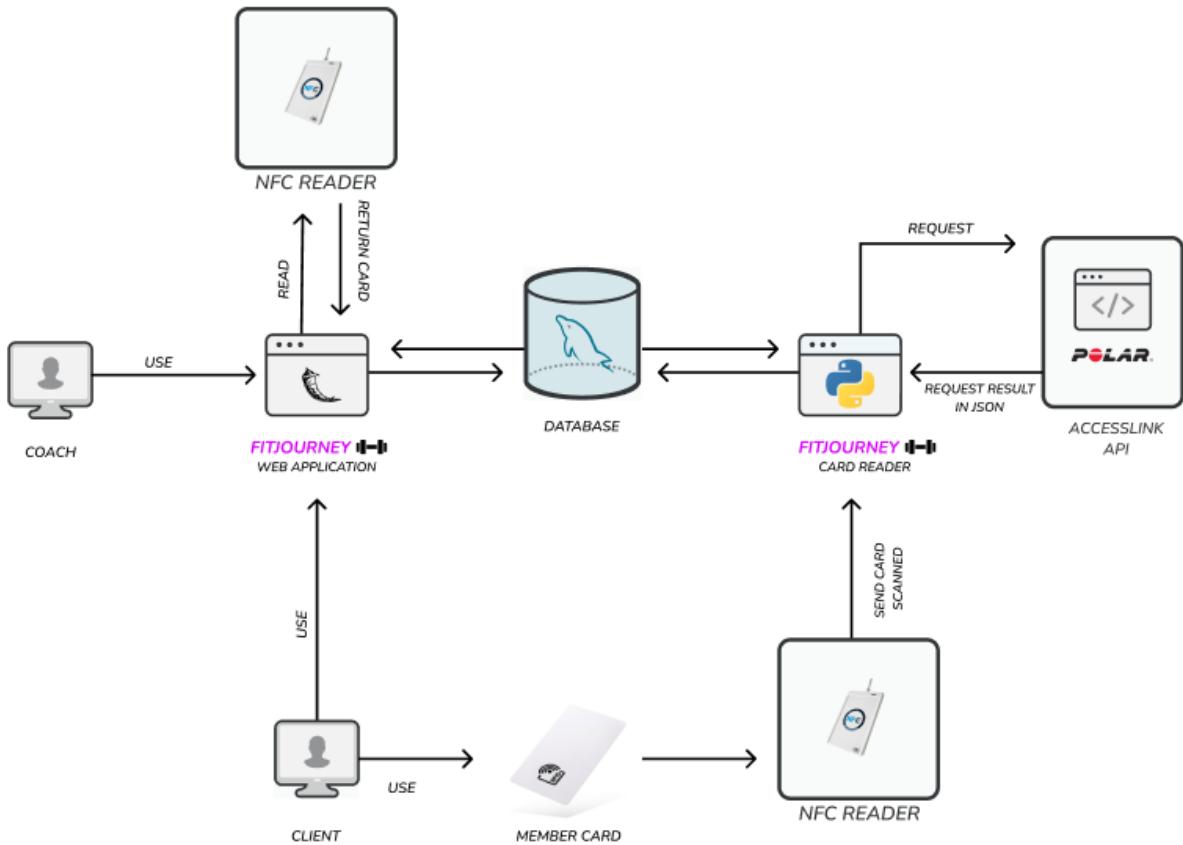
today

month week day list

	Sun 7/25	Mon 7/26	Tue 7/27	Wed 7/28	Thu 7/29	Fri 7/30	Sat 7/31
all-day				Conference Click for Google			
6am							
7am							
8am							
9am							
10am							
11am							
12pm							
1pm							
2pm							
3pm					Meeting		
4pm							
5pm							
6pm					Happy Hour		
7pm							
8pm						Dinner	
9pm							

## 1.7.4 Architecture du projet

Voici l'architecture du projet Fitjourney :



## Arborescence application principale

```

run.py
|
apps/
|
|__ __init__.py
|__ config.py
|
|__ authentication/
|   |__ __init__.py
|   |__ forms.py
|   |__ models.py
|   |__ routes.py
|   |__ util.py
|
|__ client/
|   |__ __init__.py
|   |__ forms.py
|   |__ routes.py
|   |__ util.py
|
|__ coach/
|   |__ __init__.py
|   |__ forms.py
|   |__ routes.py
|   |__ util.py
|

```

```

└── static/assets/
    ├── css/
    ├── fonts/
    ├── img/
    ├── js/
    └── vendor/.py

└── templates/
└── accounts/
└── client/
└── coach/
└── includes/
└── layout/

```

## Fichier "run.py"

---

Le fichier `run.py` est le seul fichier qui est en dehors du dossier principal de l'application. Il permet de créer et de lancer l'application Flask. On peut lancer directement l'application en exécutant ce script.

```
python3 ./run.py
```

## Dossier "Apps"

---

Le dossier `Apps` est le dossier principal de l'application. Il comprend l'ensemble du code source du projet excepté le fichier `"run.py"`. Il comprend lui-même plusieurs sous-dossiers expliqués dans les chapitres suivants.

### Dossier "authentication"

---

Le dossier `authentication` représente le Blueprint `authentication_blueprint`. On retrouve tous les fichiers utilisés pour l'implémentation des fonctionnalités d'authentification. Avec Flask, j'implémente un ORM nommé SQL Alchemy qui implémente le design pattern `Data Mapper` pour lire les données de la base de données. Chacune des tables est représentée par un modèle qui est utilisé pour interagir avec la table en question.

Sachant que l'application `FitJourney` n'a pas de fonctionnalités disponibles avant que l'utilisateur ne s'authentifie, le fichier `"models.py"` se trouve dans ce dossier, on y retrouve notamment tous les modèles.

### Fichier "authentication/routes.py"

---

Le fichier `routes` contient des fonctions python représentant les vues de la partie authentification de l'application. Chaque fonction permet de générer une vue à partir des templates `Jinja2`, à l'aide de la fonction Flask `render_template` qui provient du package `Flask.templating`.

## Dossier "client"

---

Le dossier *client* représente le Blueprint *client\_blueprint*. On retrouve tous les fichiers utilisés pour l'implémentation des fonctionnalités client.

Il contient un fichier de routes, avec toutes les routes disponibles en tant que client sur l'application et un fichier "forms.py" qui contient tous les formulaires qui peuvent être disponible en tant que client.

## Dossier "coach"

---

Le dossier *coach* représente le Blueprint *coach\_blueprint*. On retrouve tous les fichiers utilisés pour l'implémentation des fonctionnalités coach. Comme le dossier "client" il contient également un fichier de routes et un fichier "forms" pour les formulaires.

## Dossier "static/assets/css"

---

Le dossier "css" contient tous les fichiers CSS de l'application. Il contient également un dossier "bootstrap" contenant des fichiers de style de l'ensemble Bootstrap

## Dossier "static/assets/fonts"

---

Le dossier "fonts" contient les fichiers de police d'écriture utilisés dans l'application

## Dossier "static/assets/img"

---

Le dossier "img" contient toutes les images de l'application, dont notamment les photos de profil des utilisateurs.

## Dossier "static/assets/js"

---

Le dossier "js" contient tous les fichiers JavaScript nécessaire pour l'application.

## Dossier "static/assets/vendor"

---

Le dossier "vendor" contient toutes les bibliothèques tierces qui sont nécessaires au projet (ressources externes). C'est généralement dans ce dossier où sont stockées les dépendances à télécharger avec un packet manager.

## Dossier "templates"

---

Le dossier templates est structuré en plusieurs parties. Il contient les fichiers .html de l'application. Chaque Blueprint de l'application possède son dossier ici qui contient les templates nécessaire pour les vues. En

plus des dossiers représentant les Blueprints, un dossier includes est disponible. Il contient les parties à inclure sur les différentes pages de l'application comme la barre de navigation ou encore les importations de fichiers CSS ou JavaScript. Il y a également un dossier layout qui contient un fichier de base .html qui contient la structure HTML de l'application.

## Fichier "\_\_init.py"

---

Le fichier "init" est un fichier python contenant les méthodes d'initialisation de l'application. C'est notamment ici que les blueprints sont enregistrés dans l'application.

## Fichier "config.py"

---

Le fichier "config" est un fichier python contenant la configuration de l'application. Il contient toutes les constantes nécessaires au fonctionnement de l'application

## Arborescence lecture de carte

---

```

cardsChecker/
|
|__ accesslink/
|
|__ accesslink_polar.py
|
|__ authorization.py
|
|__ config.py
|
|__ config.yml
|
|__ db.py
|
|__ main.py
|
|__ utils.py

```

## Dossier "accesslink/"

---

Le dossier "accesslink" contient tous les scripts python permettant l'authentification, l'exécution des différentes transactions et récupérer des données avec l'API Polar Accesslink. La plupart des fichiers présents dans ce dossier sont fournis par Polar.

## Fichier "accesslink\_polar.py"

---

Le fichier "accesslink\_polar" contient un objet Python qui enveloppe toutes les fonctionnalités de l'API Polar à l'aide des objets et méthode disponible dans le dossier "accesslink/"

## Fichier "authorization.py"

---

Le fichier "authorization" est un script python qui permet d'authentifier un compte Polar pour avoir accès à l'API

## Fichier "config.py"

---

Le fichier "config.py" contient toutes les constantes nécessaires pour le bon fonctionnement de la lecture des cartes de membre. On y retrouve notamment la configuration pour se connecter à la base de données.

## Fichier "config.yml"

---

Le fichier "config.yml" contient la configuration du client pour avoir accès aux données avec l'API Polar. Il Contient :

- L'id de l'utilisateur
- L'id du client (Pour l'accès)
- Le secret du client
- Le token d'accès (Il est généré avec le fichier *authorization.py*)

## Fichier "db.py"

---

Le fichier "db.py" est utilisé pour se connecter à la base de données, on y retrouve également les fonctions effectuant des actions avec cette dernière.

## Fichier "main.py"

---

Le fichier "main.py" contient le script python principal qui permet la lecture des cartes et la récupération des données avec l'API Polar. C'est ce script qu'il faut exécuter pour lancer le programme.

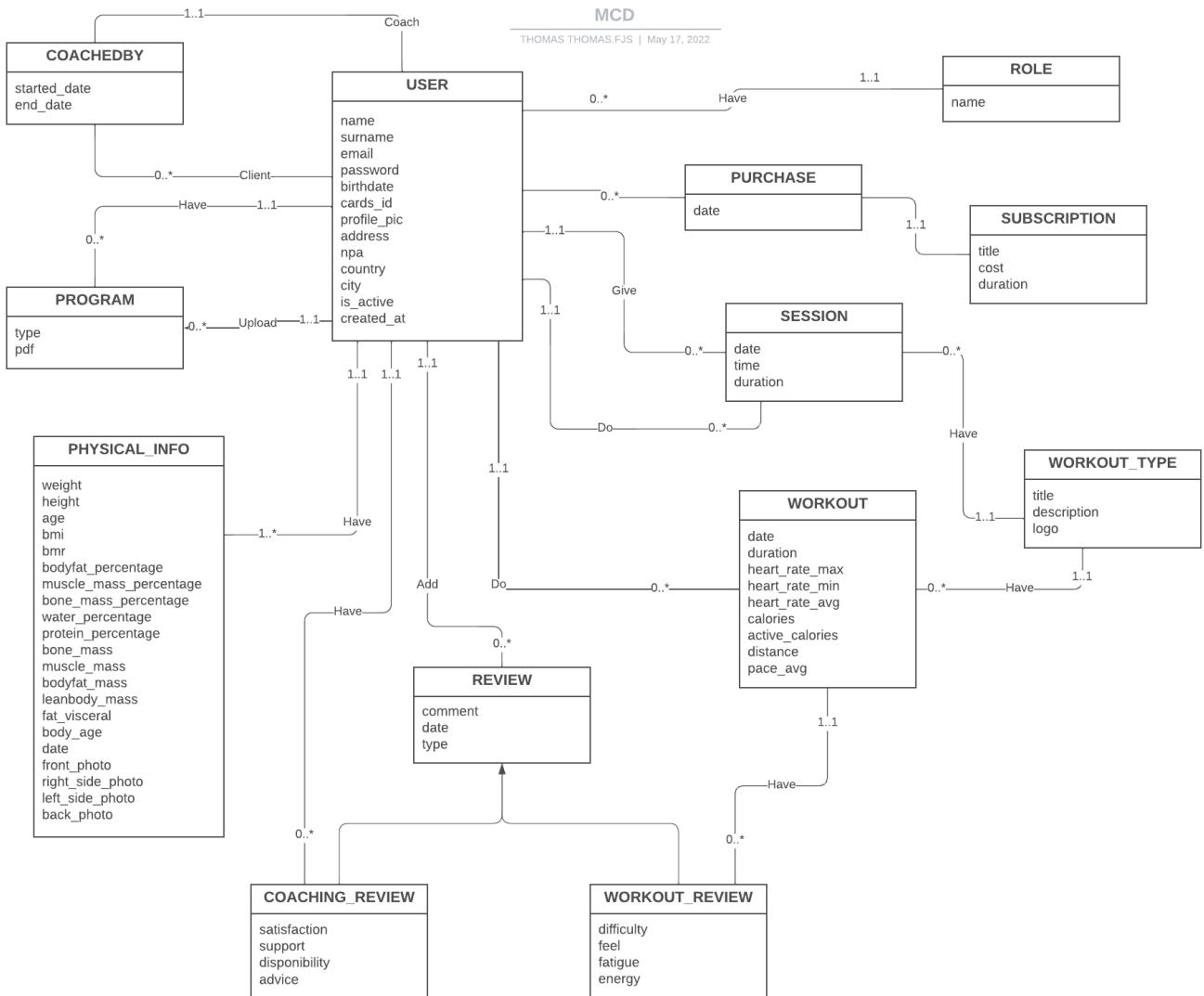
## 1.7.5 Base de données

---

Pour permettre le stockage des données, j'ai créé une base de données nommée "fitjourney". Cette base de données me permet d'enregistrer et stocker toutes les données requises pour le bon fonctionnement de l'application.

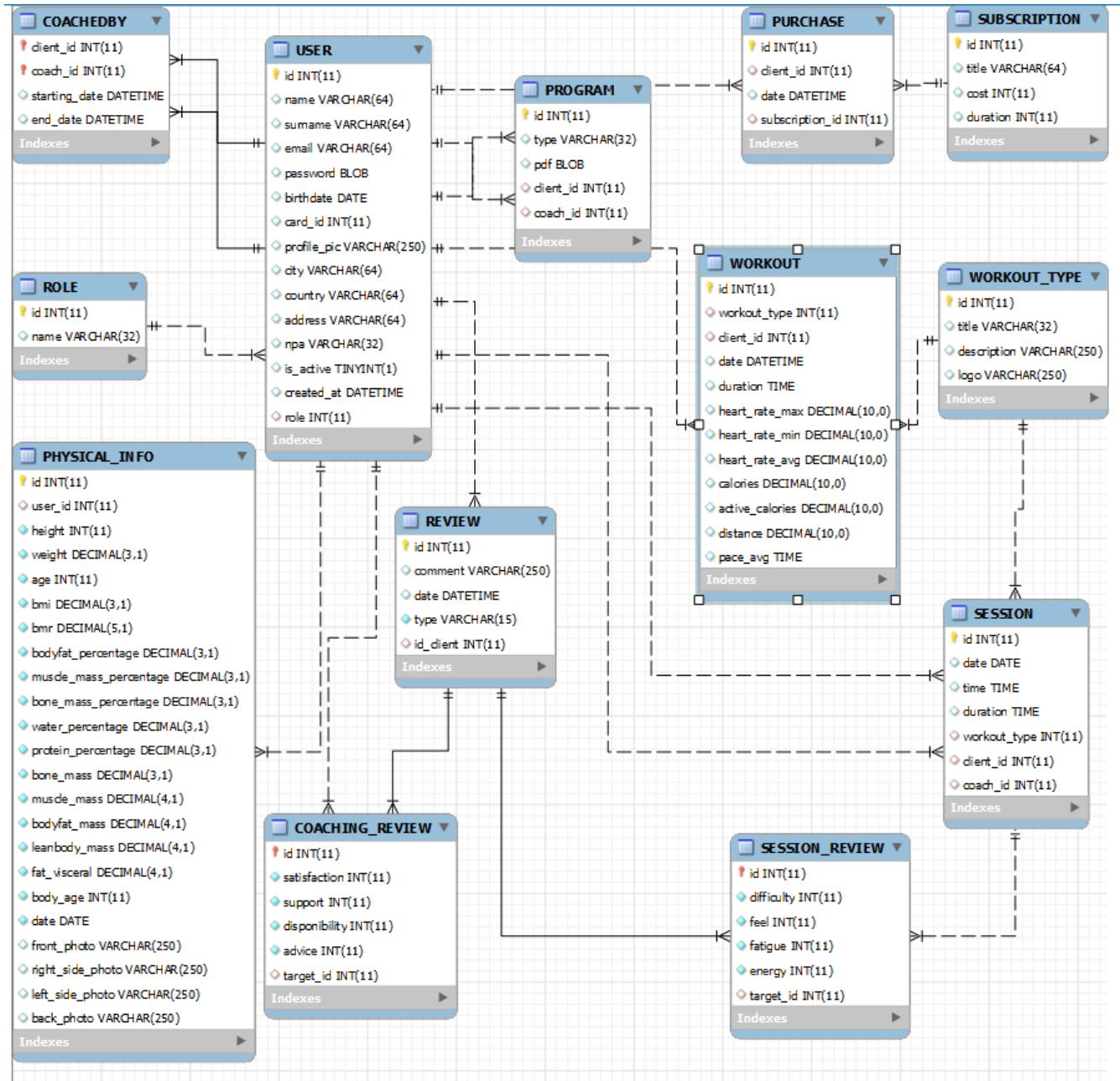
## MCD

Au lieu de créer la base de données directement, j'ai commencé par créer un MCD pour définir tous les besoins de l'application au niveau de la base de données. Pour faire mon MCD, je suis allé sur LucidChart qui est une plateforme de collaboration en ligne permettant la création de diagrammes et la visualisation de données et autres schémas conceptuels.



## Modèle physique

Une fois les besoins identifiés à l'aide du MCD, j'ai pu utiliser SQL Alchemy pour créer ma base de données directement.



## SQLAlchemy

SQLAlchemy est un ORM (mapping objet-relationnel) écrit en Python, il utilise le pattern [Data Mapper](#) et me permet de créer directement mes tables.

Exemple d'initialisation d'une table avec SQLAlchemy :

```

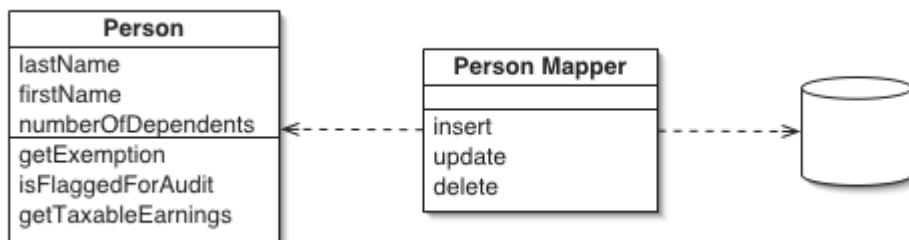
class Workout(db.Model):
    __tablename__ = 'WORKOUT'

    id = db.Column(db.Integer, primary_key=True)
    workout_type = db.Column(db.Integer, db.ForeignKey('WORKOUT_TYPE.id'))
    client_id = db.Column(db.Integer, db.ForeignKey('USER.id'))
    date = db.Column(db.DateTime)
    duration = db.Column(db.Time)
    heart_rate_max = db.Column(db.Numeric)
    heart_rate_min = db.Column(db.Numeric)
    heart_rate_avg = db.Column(db.Numeric)
    calories = db.Column(db.Numeric)
    active_calories = db.Column(db.Numeric)
    distance = db.Column(db.Numeric, nullable=True)
    pace_avg = db.Column(db.Time, nullable=True)

```

## Data Mapper

Data Mapper est un pattern qui sépare les objets en mémoire de la base de données. Il consiste à transférer les données entre les deux et à les isoler l'une de l'autre. Avec le pattern *Data Mapper*, les objets en mémoire ne doivent même pas savoir qu'une base de données est présente, ils n'ont pas besoin de code d'interface SQL, et certainement pas de connaissance du schéma de la base de données. (Le schéma de la base de données ignore toujours les objets qui l'utilisent).



## Tables

---

### Table *USER*

---

La table *USER* contient tous les utilisateurs de l'application, les coachs ainsi que les clients. les champs *email* et *card\_id* sont uniques. Le champ *card\_id* représente l'ID de la carte de membre (RFID) qui est attribuée à l'utilisateur.

### Table *PHYSICAL\_INFO*

---

La table *PHYSICAL\_INFO* contient toutes les informations physiques récupérées par le coach lors d'un bilan ou d'une prise en charge. La date permet de garder un historique pour visualiser la progression du client. La plupart des données insérées dans cette table peuvent être récupérées à l'aide d'une balance connectée.

### Table *COACHEDBY*

---

Cette table permet de différencier un coach d'un client et permet de retrouver tous les clients d'un coach. Les dates de début et de fin permettent de retrouver des anciens coachs/clients si plusieurs coachs travaillent dans la salle de sport.

### Table *PROGRAM*

---

La table *PROGRAM* contient les programmes d'entraînement et de nutrition ajouté par un coach.

### Table *ROLE*

---

La table *ROLE* contient tous les rôles de l'application. Elle permet de définir les accès que possèdent les utilisateurs.

### Table *PURCHASE*

---

La table *PURCHASE* contient l'historique de toutes les transactions effectuées. Elle permet de retrouver le type d'abonnement que chaque client a souscrit.

### Table *SUBSCRIPTION*

---

La table *SUBSCRIPTION* contient tous les différents types d'abonnement disponible. Elle permet de connaître la durée et le coût des abonnements achetés par les clients.

## Table SESSION

---

La table *SESSION* contient l'historique de toutes les sessions effectuées par les coachs avec la date et l'heure de la session ainsi que sa durée. Pour rappel, une session représente un rendez-vous avec un coach. Cela peut être pour un entraînement ou encore un bilan.

## Table WORKOUT\_TYPE

---

Cette table contient les différents types d'entraînement. Elle permet d'identifier les entraînements effectués par les clients.

## Table WORKOUT

---

La table *WORKOUT* contient toutes les données d'entraînements des séances effectuées. La majorité des données sont obtenues à l'aide des capteurs sur les montres connectées. Les données contenues dans cette table permettent de vérifier l'efficacité et l'intensité de la séance et peuvent être utilisés pour des comparaisons.

## Table REVIEW

---

La table *REVIEW* est la table "mère" des 2 tables : *COACHING\_REVIEW* et *SESSION\_REVIEW*. Elle permet de relier les reviews aux clients.

## Table COACHING\_REVIEW

---

Cette table contient tous les retours client sur le coaching effectué par le coach. Les champs disponibles ont la satisfaction, le support le coach lui apporte, la disponibilité du coach en cas de besoin et si le client souhaite continuer son suivi.

## Table SESSION\_REVIEW

---

Cette table contient tous les retours client sur les sessions qu'il effectue avec un coach. Les champs disponibles sont la difficulté, le ressenti de la séance, le niveau de fatigue à la fin de la séance et l'énergie que le client avait en arrivant.

## 1.7.6 FitJourney application

---

### Développement avec Python Flask

---

L'application web FitJourney est développé avec Python flask qui utilise Jinja2 pour le templating et Werkzeug pour faire la communication entre serveur web et application web.

#### Templating Ninja2

---

Le templating permet de créer des pages dynamiques en fonction des données fournies à la page. Dans le cas de l'application, cela permet d'afficher les données de l'utilisateur connecté ainsi qu'afficher certaines parties en fonction du niveau d'accès de l'utilisateur.

#### Template de base

---

Le concept très utile qui est la possibilité de créer une template parent. Pour illustrer cela, j'ai un fichier *add\_program.html* qui va utiliser tout le contenu du fichier *base.html*, mais qui va ajouter certaines parties en plus, selon les instructions.

Voici un aperçu du fichier *base.html* et on peut y retrouver plusieurs instructions comme :

```
{% block content %}{% endblock content%}
```

Ces instructions permettent d'insérer du contenu depuis un fichier enfant.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <title>
        FitJourney - {% block title %} {% endblock %}
    </title>
    {% include "includes/css.html" %}
    <!-- Specific CSS HERE -->
    {% block stylesheets %}{% endblock stylesheets %}
</head>
<body class="">
    {% include "includes/nav.html" %}
    <div class="main-content" id="panel">
        <!-- Header -->
        <div class="header bg-gradient-purple py-7 py-lg-8">
            <div class="container">
                <div class="header-body text-center mb-7">
                    <div class="row justify-content-center">
                        <div class="col-lg-5 col-md-6">
                            <h1 class="text-white">
                                <a target="_blank" class="text-white" href="">FitJourney</a>
                            </h1>
                            <p class="text-lead text-light">
                                The application to manage your coaching
                            </p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    {% block content %}{% endblock content %}
    </div>
    {% include "includes/scripts.html" %}
    <!-- Specific JS HERE -->
    {% block javascripts %}{% endblock javascripts %}
</body>
</html>
```

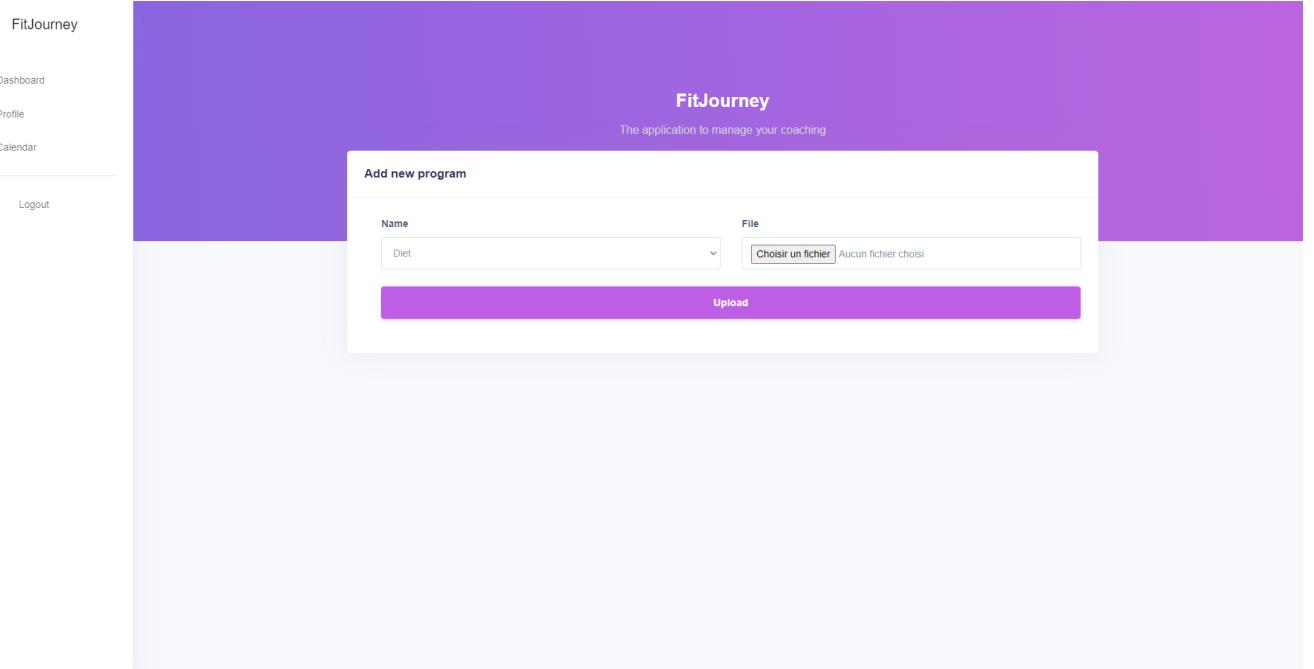
Pour créer un fichier enfant, il suffit d'ajouter l'instruction :

```
{% extends 'base.html' %}
```

pour étendre le fichier de parent. Voici un exemple de fichier enfant avec le fichier *add\_program.html* :

```
...  
  
{% extends 'layout/base.html' %}  
  
{% block title %} Add new program {% endblock title %}  
  
{% block content %}  
  
<div class="container-fluid mt--8" style="margin: auto;">  
  
    {% for category, message in get_flashed_messages(with_categories=true) %}  
  
        <div class="alert alert-{{category}} alert-dismissible fade show" role="alert">  
            {{ message }}  
            <button type="button" class="btn btn-secondary" data-bs-dismiss="alert" aria-  
                label="Close">Close</button>  
        </div>  
  
    {% endfor %}  
  
    <div class="row justify-content-center">  
        <div class="col-xl-8 ">...  
    </div>  
  
    {% endblock content %}  
  
<!-- Specific JS goes HERE -->  
{% block javascripts %}  
{% endblock javascripts %}
```

La vue de cette page :



## Routes

---

Les routes avec Python Flask sont reliées directement à des méthodes. Ces méthodes doivent forcément retourner quelque chose. L'instruction :

```
@blueprint.route('/dashboard')
```

permet de dire que la méthode qui suit l'instruction sera liée au blueprint et la route "/dashboard". (Il est possible de relier plusieurs routes à la même méthode)

```
@blueprint.route('/dashboard')
@login_required
def dashboard():
    #Check if user is coach
    if not current_user.is_coach():
        return redirect(url_for('authentication_blueprint.login'))

    nextClient = get_coach_next_session(current_user.id)
    clientLastWorkout = get_last_workout(nextClient.id) if nextClient != None else None

    clients = get_clients(current_user.id)

    return render_template('coach/dashboard.html', segment='coach_dashboard',
                           nextClient=nextClient, clientLastWorkout=clientLastWorkout, clients=clients)
```

Comme préciser ci-dessus, il est obligatoire de retourner quelque chose. Pour utiliser du templating Jinja2, Flask propose une méthode nommée `render_template()`. Le premier paramètre de cette méthode est le fichier de template que l'on veut utiliser, puis les paramètres suivants sont des données que l'on veut transmettre à la vue. Dans le cas de la route `/dashboard`, je donne le prochain client qui a rendez-vous avec le coach, la date du dernier entraînement qu'il a effectué ainsi que la liste de tous les autres clients dont il a la charge.

Comme on peut le voir sur l'image ci-dessus, il est possible d'ajouter l'instruction

```
@login_required
```

avant la méthode pour préciser que cette route n'est accessible que si un utilisateur est connecté. Cette instruction est utilisable à l'aide la librairie *Flask-Login*.

## Les routes de l'application

---

Comme mentionné plus haut dans l'[architecture blueprint](#), dans mon application, chaque route est reliée au blueprint correspondant. Voici les différentes routes qui ont été créées pour l'application *FitJourney*.

Endpoint public - Visiteur

Méthodes	Endpoint	Description
POST	/login	Permet de connecter un utilisateur
--	--	--

## Endpoint privé - Client

Méthodes	Endpoint	Description
GET	/index	Récupère les prochaines sessions que le client a planifiées
POST	/profile	Récupère toutes les données du client, permet de modifier les informations personnelles du client
GET	/review	Récupère les détails d'une review
POST	/add_review	Permet d'ajouter une review
GET	/workouts	Récupère tous les entraînements effectués par un client
GET	/workout	Récupère les détails d'un entraînement
GET	/checkup	Récupère les détails d'un bilan
POST	/change_password	Permet de modifier le mot de passe de l'utilisateur
GET	/program	Permet de télécharger le fichier pdf du programme
---	---	---

## Endpoint privé - Coach

Méthodes	Endpoint	Description
POST	/register	Permet d'inscrire un nouvel utilisateur <b>coach</b>
GET	/dashboard	Récupère les informations de la prochaine session du coach et la liste de tous ses clients
POST	/calendar	Récupère toutes les sessions d'un coach et permet d'ajouter une nouvelle session avec un client
POST	/client	Récupère toutes les données d'un client
GET	/review	Récupère les détails d'une review
POST	/add_client	Permet de créer un nouvel utilisateur client
POST	/add_program	Permet d'ajouter un programme à un client
GET	/program	Permet de télécharger le fichier pdf du programme
GET	/checkup	Récupère les détails d'un bilan
POST	/check_up	Permet d'ajouter un bilan
POST	/new_subscription	Permet de renouveler un abonnement
POST	/new_card	Permet de mettre à jour la carte membre d'un client
POST	/cancel_subscription	Permet d'annuler l'abonnement d'un client
---	---	---

## Blueprints

Avec l'utilisation des "[Flask Blueprints](#)", lors de l'initialisation de l'application, les modules sont ajoutés à l'application à l'aide de la méthode `register_blueprints`. Dans le but de mettre en production l'application plus tard, cela va être très utile, car pour enlever toute la partie coach de l'application par exemple, il suffit de retirer le dossier 'coach' de la boucle `for` qui parcourt les modules.

```
def register_blueprints(app):
    """
    Register all blueprints
    """

    for module_name in ('authentication', 'client', 'coach'):
        module = import_module('apps.{}.routes'.format(module_name))
        app.register_blueprint(module.blueprint)
```

## Client

---

### Prochaines sessions

---

La page "prochaines session" de l'application a pour but d'afficher toutes les sessions à venir pour le client. Cela permet au client d'avoir une vue d'ensemble sur les différentes sessions qu'il a prévues avec son coach. Pour récupérer toutes les prochaines sessions, j'utilise la méthode

```
get_next_session(userid)
```

Elle se trouve, avec toutes les autres méthodes utilisées pour les routes client, dans le fichier *client/utils.py*. Je n'ai besoin que de l'id du client dont on veut récupérer les sessions.

```

def get_next_session(userId):
    """
    Get all the next sessions

    Parameter(s) :
    NAME      |  TYPE   | DESC
    userId    |  INT    | The id of the user

    Return :
    | ARRAY[] | Array with all the next sessions planed for a user
    """
    sessions = db.session.query(Session.start_time, Session.end_time, Session.duration,
    WorkoutType.title, WorkoutType.logo, User.name, User.surname)
        .join(WorkoutType, Session.workout_type==WorkoutType.id)
        .join(User, Session.coach_id==User.id)
        .filter(Session.client_id==userId)
        .filter(Session.start_time>datetime.now())
        .order_by(Session.start_time)

    return sessions

```

Le tableau retourné contenant toutes les sessions est ensuite envoyé à la vue à l'aide de la méthode `render_template()`

## Profil

---

La page "profil" est une page qui est utilisé pour les 2 types d'utilisateurs, seulement plus d'informations sont affichées lorsque c'est un client qui est connecté. Cette page sert essentiellement à modifier ses informations personnelles à l'aide d'un formulaire. Beaucoup de données sont récupérées pour tout afficher sur cette page. La modification des données de l'utilisateur se fait à l'aide du model `SQLAlchemy User`. Ce qui rend la tâche beaucoup plus simple, en modifiant les propriétés correspondant aux champs modifiés on peut mettre à jour toutes les valeurs. Il suffit d'effectuer un commit à la fin pour valider les changements. Si l'utilisateur importe une nouvelle photo de profil, l'image est sauvegardée localement avec un nom unique et ce nom est enregistré dans la base de données.

```

try:
    db.session.commit()
    saver.save(os.path.join(Config.UPLOAD_FOLDER, pic_name))
    flash("Account Updated successfully !", 'success')
    return render_template("client/profile.html",
        form=update_form,
        reviewFields=coachingReviewFields,
        reviewTargetId=coachId
    )
except:
    flash("Error! Looks like there was a problem.. try again!", 'danger')
    return render_template("client/profile.html",
        form=update_form,
        reviewFields=coachingReviewFields,
        reviewTargetId=coachId
    )

```

On peut voir ici que j'essaye de commit les changements effectués, je sauvegarde l'image importée et si cela fonctionne le fichier "profile.html" est retourné avec la méthode `render_template()`. Avec la méthode `flash()`, je peux afficher un message pour confirmer à l'utilisateur que la mise à jour a fonctionné ou pas.

## Changement de mot de passe

---

La page "changement de mot de passe" est également accessible par les 2 types d'utilisateurs, elle est directement reliée à la page profil. En précisant l'instruction suivante dans le lien de référence d'un bouton ou d'un lien, on peut arriver directement sur cette page :

```
 {{ url_for('client_blueprint.change_password') }}`
```

la méthode `url_for()` permet d'obtenir l'URL qui redirige vers une route ou un blueprint en l'occurrence.

Le changement de mot de passe s'effectue en précisant son ancien mot de passe, le nouveau ainsi qu'une confirmation pour éviter les fautes de frappe. L'ancien mot de passe est ensuite vérifié à l'aide de la méthode `verify_pass()` qui permet de comparer le mot de passe saisi avec celui hashé en base de données.

```
def verify_pass(provided_password, stored_password):
    """
        Function that verify a stored password against one provided by the user
        Given an encoded password and a plain text one which is provided by the user, it
        verifies whether the provided password matches the encoded one.

    Parameter(s) :
        NAME           |   TYPE   | DESC
        provided_password | STRING | The password provided by the user
        stored_password   | BYTES  | The hashed with salt password stored in db

    Return :
        | BOOLEAN | True if passwords are the same, else False

    """

    stored_password = stored_password.decode('ascii')
    salt = stored_password[:64]
    stored_password = stored_password[64:]
    pwdhash = hashlib.pbkdf2_hmac('sha512', provided_password.encode('utf-8'),
        salt.encode('ascii'), 100000)

    pwdhash = binascii.hexlify(pwdhash).decode('ascii')

    return pwdhash == stored_password
```

Pour effectuer la vérification, je prends les 64 premiers caractères qui correspondent aux salt. Je hash ensuite le mot de passe saisis avec le même salt puis je compare avec celui enregistré en base.

Si l'ancien mot de passe est correctement renseigné, alors la propriété *password* du model User est modifiée en hashant le mot de passe à l'aide de la méthode *hash\_pass()*. Pour hashé le mot de passe, j'utilise l'algorithme "SHA-512" sur le mot de passe ainsi que le salt qui est généré.

```

def hash_pass(password):
    """
    Function that hash a password for storing
    Encodes a provided password in a way that is safe to store on a database

    Parameter(s) :
    NAME      |  TYPE   | DESC
    password  | STRING  | The password to hash

    Return :
    | BYTES | The password hashed with salt

    """
    salt = hashlib.sha256(os.urandom(60)).hexdigest().encode('ascii')
    pwdhash = hashlib.pbkdf2_hmac('sha512', password.encode('utf-8'), salt, 100000)
    pwdhash = binascii.hexlify(pwdhash)

    return (salt + pwdhash) # return bytes

```

Un message est ensuite affiché en fonction de la réussite de l'opération à l'aide de la méthode *flash()*

## Entrainements

---

La page entraînements est la page qui permet au client de visionner l'ensemble des entraînements qu'il a effectués depuis son inscription.

```

@blueprint.route('/workouts')
@login_required
def workouts():
    # Get all data from workout
    workouts = get_workouts(current_user.id)
    return render_template('client/workouts.html', segment='workouts', workouts=workouts)

```

J'utilise la méthode *get\_workouts()* qui permet de récupérer tous les entraînements d'un client et je les passe ensuite à la vue avec la méthode *render\_template()*

```

def get_workouts(clientId):
    """
    Get all the workouts done by a client

    Parameter(s) :
        NAME      |  TYPE   | DESC
        clientId |  INT    | The id of the client

    Return :
        | ARRAY[WORKOUT()] | Array with all workouts made by user
    """

    workouts = db.session.query(WorkoutType.title, Workout.id, Workout.date,
                                Workout.duration, Workout.heart_rate_avg, Workout.calories)
        .join(WorkoutType, Workout.workout_type == WorkoutType.id)
        .filter(Workout.client_id==clientId)
        .order_by(Workout.date.desc())

    return workouts

```

Les entraînements sont récupérés dans l'ordre par date décroissante (le plus récent en premier).

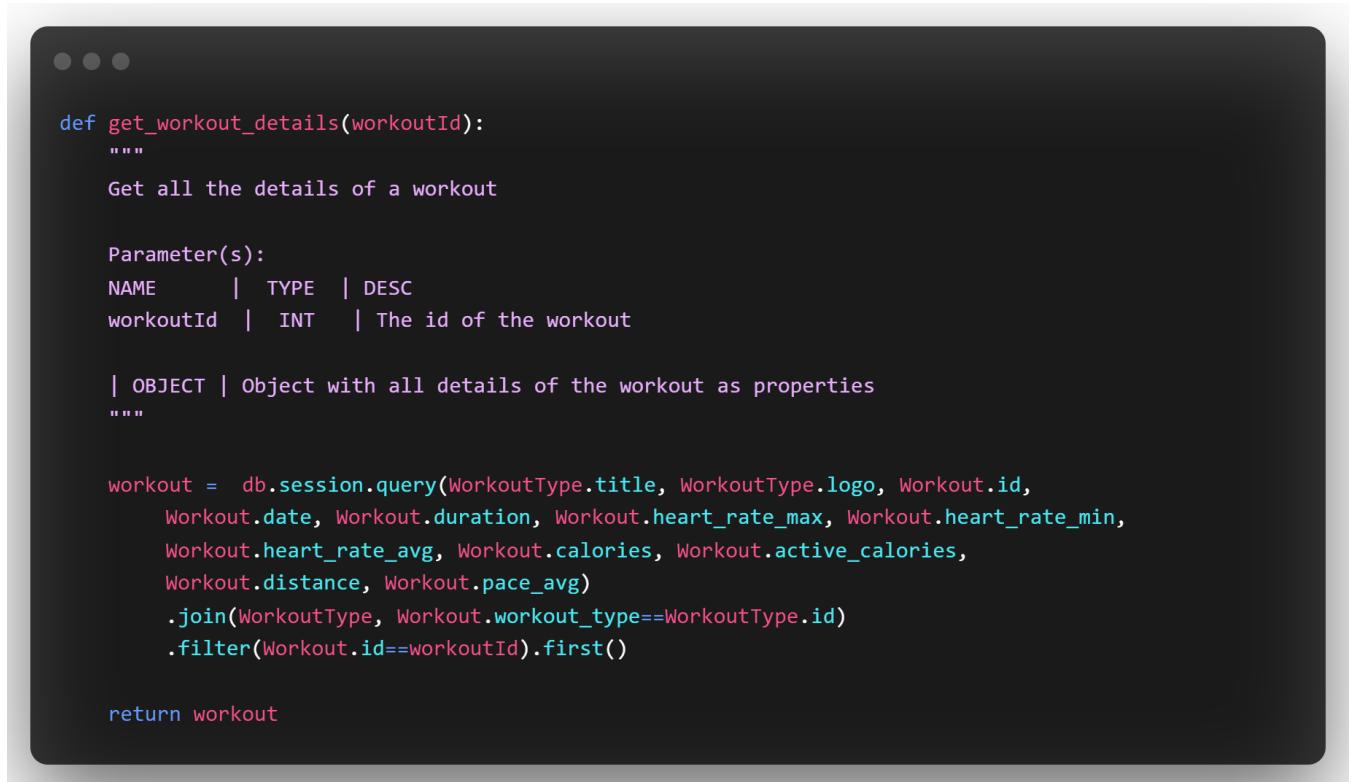
## Entrainement

---

La page "entraînement" est utilisé uniquement pour afficher les détails d'un entraînement depuis la page "entraînements" d'un client. Les liens disponibles sur la page "entraînements" qui permettent la redirection sur la page "entraînement" affichant les détails de ce dernier sont générés avec l'instruction :

```
 {{ url_for('client_blueprint.workout', Id=workout.id) }}
```

En passant, un paramètre supplémentaire à la méthode *url\_for* je peux passer des valeurs en paramètre. Je fais donc passer l'id de l'entraînement que je récupère ensuite sur cette page entraînement pour aller chercher les données correspondantes. J'utilise la méthode *get\_workout\_details()* qui me permet d'aller chercher toutes les informations nécessaires pour afficher les détails de l'entraînement en question.



```

def get_workout_details(workoutId):
    """
    Get all the details of a workout

    Parameter(s):
    NAME      |  TYPE   | DESC
    workoutId |  INT    | The id of the workout

    | OBJECT | Object with all details of the workout as properties
    """

    workout = db.session.query(WorkoutType.title, WorkoutType.logo, Workout.id,
        Workout.date, Workout.duration, Workout.heart_rate_max, Workout.heart_rate_min,
        Workout.heart_rate_avg, Workout.calories, Workout.active_calories,
        Workout.distance, Workout.pace_avg)
        .join(WorkoutType, Workout.workout_type==WorkoutType.id)
        .filter(Workout.id==workoutId).first()

    return workout

```

Les données des entraînements sont récupérées avec l'[API Polar Accesslink](#) qui donne accès aux enregistrements des montres connectés Polar.

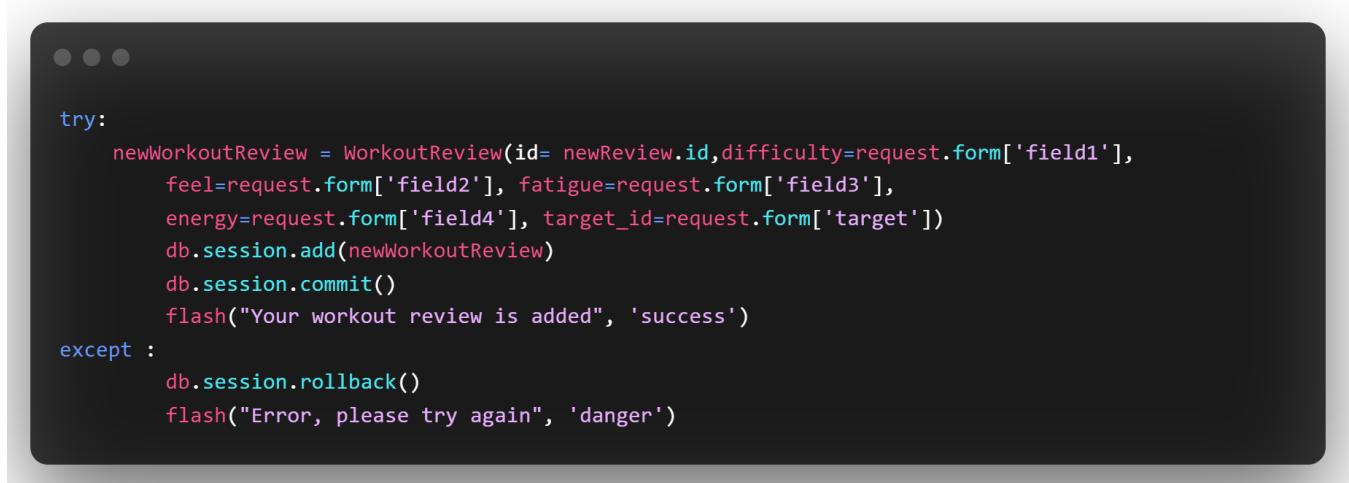
## Ajout de review

---

La page "Ajout de review" est un simple formulaire qui permet d'ajouter une review sur le coaching de manière générale ou sur un entraînement effectué. Le type de review ajouté est défini en fonction d'où le client va cliquer sur l'application. Les boutons redirigeant sur cette page possèdent un paramètre *type* qui permet d'identifier les champs à afficher ainsi que le type de review qui va être ajouté. Il y a également un paramètre *target* qui est ensuite inséré dans un input *hidden* représentant l'id de la "cible" de cette review.

Les reviews sur le coaching peuvent être ajouté n'importe quand, par contre, un entraînement ne peut posséder qu'une seule review. Pour ajouter une review, j'utilise le model SQL Alchemy *WorkoutReview* pour les reviews d'entraînements et *CoachingReview* pour les reviews de coaching. Il suffit de renseigner les propriétés avec les champs renseigner, et d'effectuer un commit pour valider l'insertion.

Exemple avec une review d'entraînement :



```

try:
    newWorkoutReview = WorkoutReview(id= newReview.id,difficulty=request.form['field1'],
        feel=request.form['field2'], fatigue=request.form['field3'],
        energy=request.form['field4'], target_id=request.form['target'])
    db.session.add(newWorkoutReview)
    db.session.commit()
    flash("Your workout review is added", 'success')
except :
    db.session.rollback()
    flash("Error, please try again", 'danger')

```

## Review

---

La page "Review" permet l'affichage d'une review que le client a ajouté. Les reviews ajoutés par le client sont listés sur son profil, il peut donc accéder aux détails de ces dernières en cliquant dessus. À nouveau, à l'aide de la méthode *url\_for* je fais passer en paramètre l'id et le type de la review sélectionné. Cela permet la récupération des infos une fois sur cette page. Les détails sont récupérés à l'aide de la méthode *get\_review\_details()* et sont ensuite passés en paramètre à la méthode *render\_template()*.

## Bilan

---

La page "Bilan" ressemble aux pages "Review" et "Entrainement" car elle ne sert qu'à afficher les valeurs qui ont été renseigné lors d'un Bilan avec le coach. L'id du bilan est passé en paramètre GET comme pour les deux autres pages (Review et Entrainement). Une fois récupéré, j'utilise la méthode *get\_check\_up()* en donnant l'id du bilan en paramètre pour avoir toutes les données et les passer ensuite en paramètre à la méthode *render\_template()* qui les fera passer à la vue

## Coach

---

### Tableau de bord

---

La page "Tableau de bord" permet d'afficher la prochaine session du coach ainsi que la liste de tous ses clients. Une vérification est effectuée au début de la route pour éviter qu'un client se retrouve sur cette page.

```
...  
@blueprint.route('/dashboard')  
@login_required  
def dashboard():  
    #Check if user is coach  
    if not current_user.is_coach():  
        return redirect(url_for('authentication_blueprint.login'))  
  
    nextClient = get_coach_next_session(current_user.id)  
    clientLastWorkout = get_last_workout(nextClient.id) if nextClient != None else None  
  
    clients = get_clients(current_user.id)  
  
    return render_template('coach/dashboard.html', segment='coach_dashboard',  
                           nextClient=nextClient, clientLastWorkout=clientLastWorkout, clients=clients)
```

La vérification est effectuée avec la méthode *is\_coach()* de l'objet *User* qui permet de vérifier si l'utilisateur connecté possède le rôle de coach. Si l'utilisateur est un coach, alors les infos du prochain client, la date de son dernier entraînement ainsi que la liste de tous les clients dont le coach a la charge sont récupérés et donnés en paramètres à la vue.

## Calendrier

---

La page "Calendrier" permet d'afficher un agenda affichant toutes les sessions que le coach a enregistrées. Il peut également en ajouter une avec le formulaire qui est disponible juste à côté. Le calendrier est affiché à l'aide de la bibliothèque JavaScript *FullCalendar.io*, il est initialisé en JavaScript. Les événements à inscrire dans le calendrier doivent être assignés à la propriété *events*: de l'objet JavaScript *FullCalendar.Calendar()*. La méthode *get\_session()* est donc utilisé pour récupérer toutes les sessions enregistrées pour le coach connecté dans le bon format pour que les sessions soient interprétés comme un événement pour le calendrier.

```

def get_sessions(coachId):
    """
    Get all sessions as event and create array to be used with FullCalendar.io

    Parameter(s):
    NAME      |  TYPE   | DESC
    coachId   |  INT    | the id of the coach

    Return :
    | Array[Obj()] | Array of object contains properties required to be an event with
    FullCalendar (title, start, end)
    """

    sessions = db.session.query(User.name, User.surname, Session.start_time,
        Session.end_time)
        .join(User,Session.client_id==User.id)
        .filter(Session.coach_id==coachId)

    result = []

    for session in sessions:
        result.append(
            {
                'title': session.name + " " + session.surname,
                'start' : session.start_time.strftime("%Y-%m-%dT%H:%M:%S"),
                'end'   : session.end_time.strftime("%Y-%m-%dT%H:%M:%S")
            })
    return result

```

La méthode récupère toutes les sessions à venir et les ajoutent dans un tableau sous la forme d'objets avec 3 propriétés

```
{
'title':
'start':
'end':
}
```

Si le coach ajoute une session à l'aide du formulaire, les valeurs des champs sont récupérés et utilisés pour avoir le format désiré. Si le coach n'a pas de client à charge, il ne sera pas en mesure de sélectionner un client pour la session qu'il souhaite ajouter. S'il essaye, une erreur sera affichée à l'aide de la méthode `flash()`. Le `DateTime` de fin est généré automatiquement à l'aide de la date et l'heure de début sélectionné ainsi que la durée. Pour enregistrer sous le format `DateTime`, l'heure de la session est ajouté à la date. La session est ajoutée à l'aide du model SQLAlchemy `Session`, l'ajout est validé avec un `commit`.

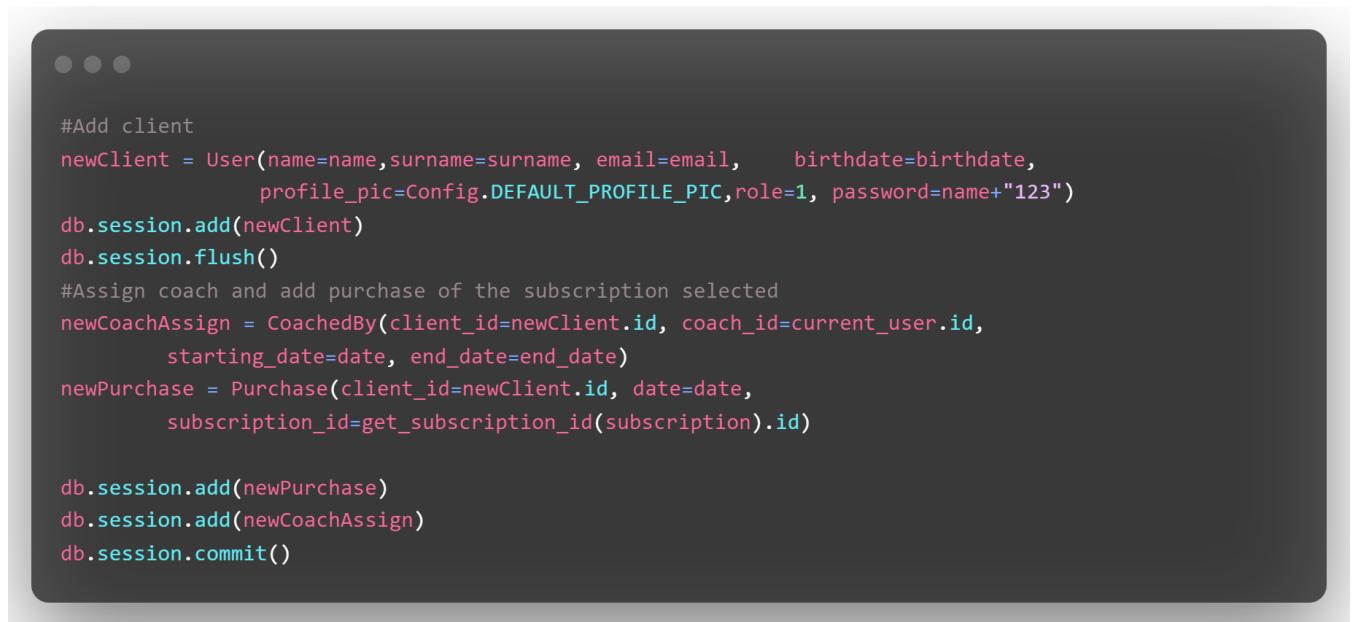
Une fois la session ajoutée, un mail est envoyé au client avec les informations de la session.

## Ajout client

La page "Ajout client" permet de créer un nouveau compte client et d'assigner son suivi au coach qui crée son compte. Cette page contient également une vérification avec la méthode `is_coach()` pour vérifier que l'utilisateur sur cette page est bien un coach. L'input de type `select` pour sélectionner le type d'abonnement souhaité est généré automatiquement, les différents abonnements sont récupérés depuis la base de données avec la méthode `get_all_subscription()`. Pour ajouter le client dans la base de données, j'utilise à nouveau les modèles `SQLAlchemy`, cette fois-ci plusieurs sont nécessaires (`User`, `CoachedBy` et `Purchase`). Pour pouvoir effectuer l'insertion comme il faut, je commence par l'utilisateur puis j'utilise l'instruction :

```
db.session.flush()
```

qui me permet d'obtenir l'id de l'utilisateur qui va être inséré (utilisé pour l'insertion avec les 2 autres modèles). Une fois les 3 modèles remplis et ajouté, j'effectue un commit pour valider les insertions.



```
#Add client
newClient = User(name=name, surname=surname, email=email, birthdate=birthdate,
                  profile_pic=Config.DEFAULT_PROFILE_PIC, role=1, password=name+"123")
db.session.add(newClient)
db.session.flush()
#Assign coach and add purchase of the subscription selected
newCoachAssign = CoachedBy(client_id=newClient.id, coach_id=current_user.id,
                           starting_date=date, end_date=end_date)
newPurchase = Purchase(client_id=newClient.id, date=date,
                       subscription_id=get_subscription_id(subscription).id)

db.session.add(newPurchase)
db.session.add(newCoachAssign)
db.session.commit()
```

Une fois le commit effectué, un mail est envoyé au client pour l'informer de ses identifiants de connexion à l'application. Le mot de passe est généré automatiquement.

## Client

La page "Client" permet aux coachs de visionner le profil d'un de leurs clients. Toutes les données nécessaires sont récupérés depuis la base de données et envoyées à la vue. On y retrouve énormément de données comme sur la page profil du client.

```
return render_template('coach/client.html', segment='client', form=form, reviews=reviews,
    client=clientDetails, subscriptionUntil=subscriptionUntil, wrktTypeCount=wrktTypeCount,
    wrktTypeList=wrktTypeList, nbWorkoutPerMonth=nbWorkoutPerMonth,
    workoutProgram=workoutProgram, dietProgram=dietProgram, physicalInfo=physicalInfo,
    avgCalories=avgCalories, avgHeartRate=avgHeartRate, totalTime=totalTime,
    weightUpdate=weightUpdate, checkUps=checkUps)
```

Sur la template Jinja2, on retrouve énormément de condition comme par exemple pour les programmes d'entraînements qui peuvent ne pas avoir été importé par le coach :

```
{% if workoutProgram != None %}
```

Cela permet d'afficher quand même quelque chose même s'il n'y a pas encore de valeurs qui a été ajouté en base.

## Ajout programme

---

La page "Ajout programme" est également soumise à une vérification, pour éviter qu'un client arrive sur cette page. Pour ajouter un programme, il suffit de sélectionner le type et d'ajouter le pdf que l'on souhaite importer. Le pdf est lu avec la méthode python `read()` qui permet de retourner les bytes du fichier. Ils sont ensuite enregistrés dans la base de données dans un champ `LONGBLOB`. Une fois le programme enregistré, un mail est envoyé au client pour le notifier que son nouveau programme est disponible.

Pour télécharger le programme, j'utilise la route "/program" et je passe en paramètre GET l'id du programme. La route retourne la méthode python `send_file()` qui permet de télécharger un fichier. J'utilise également l'objet `BytesIO` qui permet d'écrire le fichier à partir des bytes qui ont été enregistré en base.

```

@blueprint.route('/program', methods=['GET'])
@login_required
def program():

    program_id = request.args.get('programId')
    program_type = request.args.get('programType')

    program = get_program_by_id(program_id)

    return send_file(BytesIO(program.pdf), attachment_filename=str(program_type+'.pdf'),
                     as_attachment=True)

```

L'ajout de programme devait se faire par import de fichier Excel sous format uniformisé à la base. Après plusieurs réflexions, j'ai décidé de changer cela en acceptant uniquement les pdf. Cela permet plus de liberté aux coachs et évitera de "casser" leurs habitudes s'ils utilisent l'application. Les coachs peuvent garder leurs méthodes de rendu pour leurs programmes et simplement importer les pdfs sur l'application.

## Ajout Bilan

---

La page "Ajout bilan" ne contient qu'un formulaire pour ajouter les valeurs du bilan. Les valeurs demandées sont toutes récupérables à l'aide d'une balance connectée. L'id, le nom, prénom et l'âge du client sont récupérés et insérés automatiquement et sont statiques, le coach ne peut pas modifier ses informations. L'âge du client est calculé en fonction de sa date de naissance et la date d'aujourd'hui.

```

static_infos = {
    "id" : infos.id,
    "name" : infos.name,
    "surname" : infos.surname,
    "age" : (today.year - infos.birthdate.year - ((today.month, today.day) <
                                                 (infos.birthdate.month, infos.birthdate.day)))
}

```

## Modification de carte

---

La modification de carte de membre s'effectue depuis la [page client](#). Une fois le bouton cliqué, la méthode `get_card_id()` est appelée. Elle va lire à l'aide du [Lecteur NFC](#) une carte que le coach va scanner. Un timeout de 30 secondes est défini pour éviter d'attendre trop longtemps, si les 30 secondes sont écoulées et que le coach n'a pas présenté de carte alors une erreur `flash()` est affichée. La méthode `get_card_id()` utilise la

librairie python [smartcard](#) qui permet d'utiliser le lecteur. Si le lecteur détecte une carte, alors une connexion est établie. À l'aide de la transmission, je peux récupérer l'UID de la carte sous le format :

```
[12, 123, 89, 09]
```

Je n'ai plus qu'à modifier ce format en string pour pouvoir l'insérer en base plus tard.

```
def get_card_id():
    card_type = AnyCardType()

    request = CardRequest(timeout=30, cardType=card_type)

    card = None

    while card == None:
        time.sleep(0.1)

        service = None
        try:
            service = request.waitforcard()
        except:
            print("ERROR: No card detected")
            break

        conn = service.connection
        conn.connect()

        get_uid = util.toBytes("FF CA 00 00 00")

        data, sw1, sw2 = conn.transmit(get_uid)

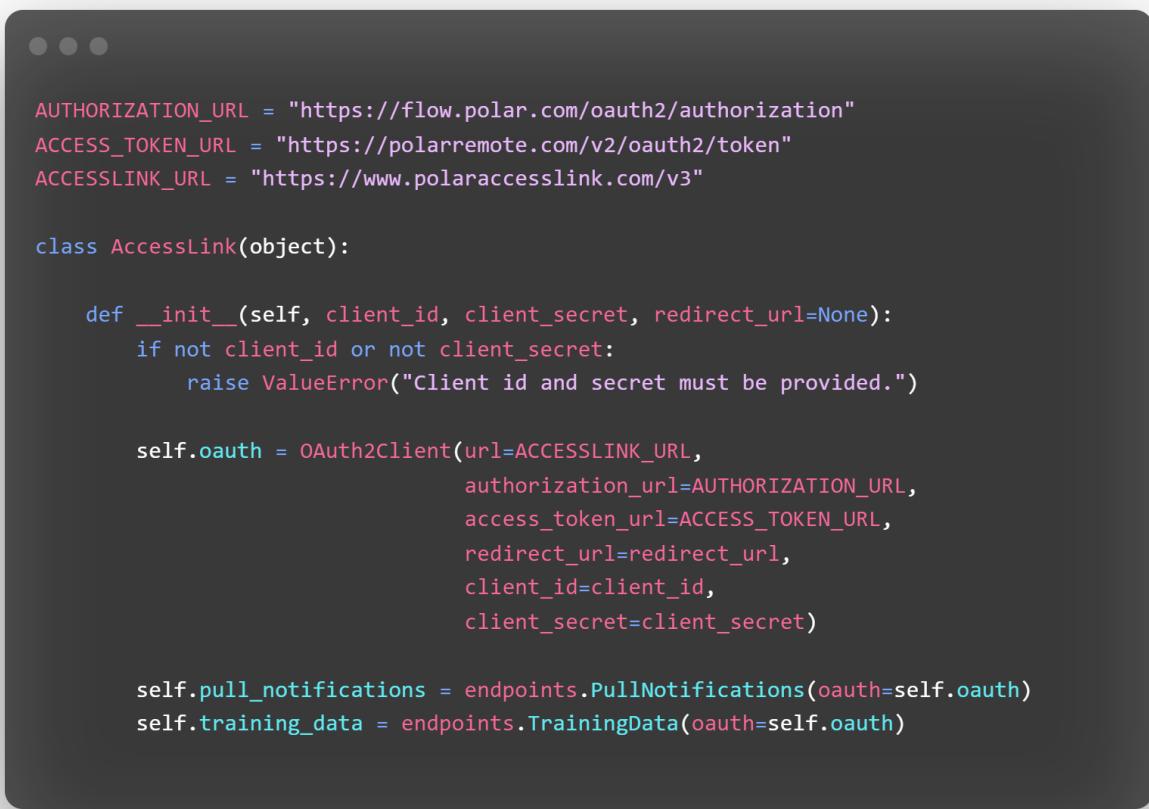
        card = ' '.join(str(e) for e in data)

    return card
```

## 1.7.7 FitJourney cards checker

### Authentification à l'API

Comme expliqué plus haut dans la section [Entrainement](#), les entraînements sont récupérés à l'aide de l'API Polar. Pour s'authentifier envers l'API, j'utilise l'objet `Accesslink` qui utilise l'objet `OAuth2Client` qui est fourni par Polar afin de s'authentifier en utilisant le protocole OAuth2. Les identifiants sont quant à eux récupérés du fichier `config.yml`. Si les identifiants ne sont pas dans ce fichier, il faut alors exécuter le script `authorization.py` qui va se charger de générer et d'écrire les identifiants et le token d'accès dans le fichier `config.yml`.



```

AUTHORIZATION_URL = "https://flow.polar.com/oauth2/authorization"
ACCESS_TOKEN_URL = "https://polarremote.com/v2/oauth2/token"
ACCESSLINK_URL = "https://www.polaraccesslink.com/v3"

class AccessLink(object):

    def __init__(self, client_id, client_secret, redirect_url=None):
        if not client_id or not client_secret:
            raise ValueError("Client id and secret must be provided.")

        self.oauth = OAuth2Client(url=ACCESSLINK_URL,
                                 authorization_url=AUTHORIZATION_URL,
                                 access_token_url=ACCESS_TOKEN_URL,
                                 redirect_url=redirect_url,
                                 client_id=client_id,
                                 client_secret=client_secret)

    self.pull_notifications = endpoints.PullNotifications(oauth=self.oauth)
    self.training_data = endpoints.TrainingData(oauth=self.oauth)

```

### Utilisation de l'API

Dans le dossier `endpoints/` on retrouve les objets python qui permettent d'effectuer les requêtes à l'API. La principale requête que j'utilise est celle des "training data". C'est cette requête qui permet de récupérer les

informations de l'entraînement enregistré avec les montres connectées. Les données sont récupérés en format JSON, voici un exemple de retour de cette requête :

```
{  
    "id": 1937529874,  
    "upload-time": "2008-10-13T10:40:02Z",  
    "polar-user": "https://www.polaraccesslink/v3/users/1",  
    "transaction-id": 179879,  
    "device": "Polar M400",  
    "device-id": "1111AAAA",  
    "start-time": "2008-10-13T10:40:02Z",  
    "start-time-utc-offset": 180,  
    "duration": "PT2H44M",  
    "calories": 530,  
    "distance": 1600,  
    "heart-rate": {  
        "average": 129,  
        "maximum": 147  
    },  
    "training-load": 143.22,  
    "sport": "OTHER",  
    "has-route": true,  
    "club-id": 999,  
    "club-name": "Polar Club",  
    "detailed-sport-info": "WATERSPORTS_WATERSKI",  
    "fat-percentage": 60,  
    "carbohydrate-percentage": 38,  
    "protein-percentage": 2  
}
```

Les données récupérées sont pour la plupart dans le bon format, la durée de l'entraînement est malheureusement dans un format très spécial. La durée est représentée par un *string* commençant par

"PT" et est suivie par les valeurs (heures, minutes, secondes). Chaque valeur est suivie d'une lettre pour les séparer :

- La valeur pour les heures est suivie du caractère 'H'
- La valeur pour les minutes est suivie du caractère 'M'
- La valeur pour les secondes est suivie du caractère 'S'

Mais s'il n'y a pas de valeur pour les heures par exemple, alors le caractère 'H' ne figurera pas dans le string.

J'ai donc effectué 3 conditions pour pouvoir bien formater dans chaque cas le string avec la méthode `strptime` de la librairie `datetime`

```
if 'H' in exercise_summary["duration"]:
    duration = datetime.strptime(exercise_summary["duration"], 'PT%HH%MM')
elif 'M' in exercise_summary["duration"]:
    duration = datetime.strptime(exercise_summary["duration"], 'PT%MM%S.%fs')
elif 'S' in exercise_summary["duration"]:
    duration = datetime.strptime(exercise_summary["duration"], 'PT%S.%fs')
```

J'insère ensuite les données voulues dans la base avec la librairie `mysql.connector`.

## 1.8 Tests

---

### 1.8.1 Tests unitaires

---

Aucun test unitaire n'a été réalisé pour l'application. Cependant, toutes les fonctionnalités de l'application ont été testées "manuellement" avec une partie de débogage. Les 2 parties de mon projet étant développé majoritairement en Python, il serait intéressant d'ajouter des tests unitaires à l'aide du framework [Pytest](#) par exemple. Cela serait très utile d'avoir des tests unitaires dans le but de mettre en production l'application.

## 1.8.2 Tests utilisateurs

---

FitJourney Web application

---

En tant que coach :

---

Action	Valeur(s)	Attente(s)	Résultat
Création d'un compte avec 2 mots de passes différents	<ul style="list-style-type: none"> <li>• Name : "Thomas";</li> <li>• Surname : "Fujise" ;</li> <li>• Email : "thomas.fjs@eduge.ch";</li> <li>• Birthdate : 15.09.2000;</li> <li>• Password : "Super2012";</li> <li>• Confirm Password : "Super";</li> </ul>	Un message d'erreur "Passwords Must Match!" s'affiche.	OK
Création d'un compte coach	<ul style="list-style-type: none"> <li>• Name : "Thomas";</li> <li>• Surname : "Fujise" ;</li> <li>• Email : "thomas.fjs@eduge.ch";</li> <li>• Birthdate : 15.09.2000;</li> <li>• Password : "Super2012";</li> <li>• Confirm Password : "Super2012";</li> </ul>	Redirection sur la page "login" et un message apparaît pour confirmer la création	OK
Création d'un compte avec un email existant	<ul style="list-style-type: none"> <li>• Name : "Thomas";</li> <li>• Surname : "Fujise" ;</li> <li>• Email : "thomas.fjs@eduge.ch";</li> <li>• Birthdate : 15.09.2000;</li> <li>• Password : "Super2012";</li> <li>• Confirm Password : "Super2012";</li> </ul>	Message d'erreur "Email already registered"	OK
Connexion avec le compte et un mot de passe erroné	<ul style="list-style-type: none"> <li>• Email : "thomas.fjs@eduge.ch";</li> <li>• Password : "Super";</li> </ul>	Un message d'erreur apparaît	OK
Connexion avec le compte	<ul style="list-style-type: none"> <li>• Email : "thomas.fjs@eduge.ch";</li> <li>• Password : "Super2012";</li> </ul>	Redirection sur la page tableau de bord	OK
	/		OK

Action	Valeur(s)	Attente(s)	Résultat
Clique sur le bouton "Add new client"		Redirection sur le formulaire d'ajout d'un nouveau client	
Ajout d'un nouveau client	<ul style="list-style-type: none"> <li>• Name : "John";</li> <li>• Surname : "Doe";</li> <li>• Email : "thomasfjs@gmail.com";</li> <li>• Birthdate : 12.10.2001;</li> <li>• Height : 178;</li> <li>• Starting date : 08.06.2022</li> <li>• Subscription type : "1 Month"</li> </ul>	Un message de confirmation est affiché, le client est ajouté à la liste des clients et un email est envoyé à l'email renseigné contenant les identifiants du compte client	OK
Clique sur le nom d'un client dans la liste	/	Redirection sur la page profil du client sélectionné	OK
Ajout d'une carte de membre	<ul style="list-style-type: none"> <li>• Carte utilisé : 832141011</li> </ul>	Une fois la carte scanné, un message de confirmation est affiché et le numéro <b>832141011</b> figure dans le champ "Card ID"	OK
Pas de carte scannée pendant >30s	/	Un message d'erreur "No card detected" apparait	OK
Clique sur le bouton "Upload new program"	/	Redirection vers le formulaire d'ajout de programme	OK
Ajout d'un programme	<ul style="list-style-type: none"> <li>• Type : "Workout"</li> <li>• File : "workout.pdf"</li> </ul>	Un message de confirmation apparait, on peut voir un bouton "Workout" dans la section "Programs" apparaitre et un mail est envoyé à l'email du client pour le notifier.	OK

Action	Valeur(s)	Attente(s)	Résultat
Clique sur le bouton "Check up"	/	Redirection vers la page d'ajout d'un nouveau bilan	OK
Ajout d'un bilan	<ul style="list-style-type: none"> <li>• Weight : 80.2</li> <li>• Height : 178</li> <li>• Water : 41.6</li> <li>• Protein : 10</li> <li>• Muscle Mass % : 55</li> <li>• Body Fat % : 41.7</li> <li>• Bone mass % : 3.3</li> <li>• BMR : 1336.0</li> <li>• Muscle Mass (kg) : 45.9</li> <li>• Body fat (kg) : 34.8</li> <li>• Visceral fat (kg) : 10</li> <li>• Bone mass (kg) : 2.8</li> <li>• Lean body mass (kg) : 48.7</li> <li>• Body age : 43</li> <li>• BMI : 31.4</li> </ul>	Un message confirmation est affiché, on peut voir le bilan s'ajouter sur la droite dans la section "Check up"	OK
Clique sur le bouton "Details" à côté d'un bilan	/	Redirection sur la page de détails du bilan	OK
Renouvellement d'abonnement	<ul style="list-style-type: none"> <li>• Starting date : 10.06.2022</li> <li>• Subscription type : "3 Months"</li> </ul>	Un message de confirmation est affiché, la date de fin d'abonnement a été modifier dans le champ "Subscription Until"	OK
Annulation d'un abonnement	/	Le dernier abonnement souscrit est supprimé, la date de fin d'abonnement redevient celle du précédent abonnement	OK
	/		OK

Action	Valeur(s)	Attente(s)	Résultat
Clique sur le bouton "Profile" dans la barre de navigation		Redirection sur la page profil du coach	
Modification profil coach	<ul style="list-style-type: none"> <li>• Last name : "Fuji"</li> <li>• Address : "Chemin des curiades 35"</li> <li>• City : "Bernex"</li> <li>• Country : "Switzerland"</li> <li>• Postal code : "1233"</li> </ul>	Les modifications sont effectuées et un message de confirmation apparaît.	OK
Clique sur le bouton "Calendar" dans la barre de navigation	/	Redirection vers l'agenda du coach ainsi que le formulaire d'ajout d'une nouvelle session	OK
Inscription d'une session avec un client	<ul style="list-style-type: none"> <li>• Client : "John Doe"</li> <li>• Date : 10.06.2022</li> <li>• Time for the session : 10:00</li> <li>• Type : "Weightlifting"</li> <li>• Duration (h) : 1</li> </ul>	Un message de confirmation apparaît, la session a été ajoutée au calendrier	OK
/	/	La prochaine session est visible sur le tableau de bord avec les détails de la session	OK

## En tant que client

---

Action	Valeur(s)	Attente(s)	Résultat
Connexion avec les identifiants fournis par mail	<ul style="list-style-type: none"> <li>Email : "thomas.fjs@eduge.ch";</li> <li>Password : "John123";</li> </ul>	Redirection sur la page d'accueil client	OK
/	/	Les prochaines sessions sont affichées sur la page d'accueil client	OK
Clique sur le bouton "Profile" dans la barre de navigation	/	Redirection sur la page profil	OK
Modification du profil	<ul style="list-style-type: none"> <li>Birthdate : 13.10.2001</li> <li>Profile picture : "logo.jpg"</li> <li>Country : Switzerland</li> </ul>	Un message de confirmation apparaît, la photo de profil et les informations sont modifiés	OK
Clique sur le bouton "change password"	/	Redirection vers le formulaire de changement de mot de passe	OK
Changement de mot de passe avec 2 mots de passe différents	<ul style="list-style-type: none"> <li>Old password : "John123"</li> <li>New password : "Super"</li> <li>Confirm new password : "Super2012"</li> </ul>	Un message d'erreur "Passwords must match" apparaît	OK
Changement de mot de passe	<ul style="list-style-type: none"> <li>Old password : "John123"</li> <li>New password : "Super2012"</li> <li>Confirm new password : "Super2012"</li> </ul>	Redirection sur la page profil et message de confirmation apparaît	OK
Téléchargement du programme	/	En cliquant sur le bouton "Workout program" le pdf est téléchargé	OK

Action	Valeur(s)	Attente(s)	Résultat
Clique sur le bouton "Add new review"	/	Redirection vers le formulaire d'ajout de review sur le coaching	OK
Ajout d'une review coaching	<ul style="list-style-type: none"> <li>• Satisfaction : 8</li> <li>• Support : 8</li> <li>• Disponibility : 10</li> <li>• Advice : 9</li> <li>• Comment : "This coach is super !"</li> </ul>	Redirection vers la page profil, un message de confirmation d'ajout apparaît et on peut voir la review dans la section "Reports" en bas à droite	OK
Clique sur le bouton "Details" à côté d'une review	/	Redirection sur la page de détails de la review	OK
Clique sur le bouton "Workouts" dans la barre de navigation	/	Redirection sur la page affichant tous les entraînements effectués	OK
Clique sur le bouton "Details" à côté d'un entraînement	/	Redirection sur la page de détail de l'entraînement	OK
Clique sur le bouton "Add review" sur la page de l'entraînement	/	Redirection sur la page pour ajouter une review sur l'entraînement	OK
Ajout d'une review d'entraînement	<ul style="list-style-type: none"> <li>• Difficulty : 7</li> <li>• Feel : 8</li> <li>• Fatigue : 6</li> <li>• Energy: 8</li> <li>• Comment : "This was a great workout"</li> </ul>	Redirection sur la page listant tous les entraînements effectués	OK
Rajout d'une review sur le même entraînement		Redirection sur la page listant tous les entraînements avec un message d'erreur	OK

Action	Valeur(s)	Attente(s)	Résultat
	<ul style="list-style-type: none"> <li>• Difficulty : 7</li> <li>• Feel : 8</li> <li>• Fatigue : 6</li> <li>• Energy: 8</li> <li>• Comment : "This was a great workout"</li> </ul>	"Workout already reviewed"	
Clique sur le bouton "Logout" dans la barre de navigation	/	L'utilisateur est déconnecté et redirigé vers la page de connexion	OK

## FitJourney card checker

---

Action	Valeur(s)	Attente(s)	Résultat
Scan d'une carte inconnue	<ul style="list-style-type: none"> <li>• Carte utilisée : 5874146165</li> </ul>	Message s'affiche dans la console : "Unrecognized card"	OK
Scan d'une carte de membre assignée en arrivant	<ul style="list-style-type: none"> <li>• Carte utilisée : 832141011</li> </ul>	La carte est reconnue et un message de bienvenue avec le nom du client est affiché dans la console : "Welcome John Doe"	OK
Scan de la même carte de membre pour la sortie	<ul style="list-style-type: none"> <li>• Carte utilisée : 832141011</li> </ul>	La carte est reconnue et un message est affiché en console : "Goodbye John Doe"	OK

## 1.9 Conclusion

---

### 1.9.1 Points positifs

---

Globalement, j'ai pu tirer pas mal de points positifs concernant ce travail de diplôme. Premièrement, le fait de travailler sur un projet de cette ampleur. Lors de ma formation en tant que Technicien ES, j'ai que très rarement eu l'occasion de travailler sur des projets similaires. L'expérience acquise lors d'un projet comme ceci ne peut être que positif pour le futur.

Deuxièmement, le fait d'apprendre et d'améliorer ses connaissances sur des nouvelles technologies est un point très positif. Je suis beaucoup plus à l'aise avec Python Flask par exemple, ce qui n'était pas forcément le cas au début du projet.

Pour terminer, je ressors beaucoup de positif des étapes suivies tout le long du projet. J'ai très rarement suivi toutes les étapes dans l'ordre comme je l'ai fait pour ce projet, par exemple le fait de commencer par effectuer des maquettes d'interfaces ou encore faire un MCD au préalable. Ce sont tous des longs points effectués au début, mais s'ils sont bien faits, cela permet un gain de temps conséquent. Il y a également le fait de poser mes réflexions dans le journal de bord, cela m'a permis d'identifier des problèmes avant de tomber dessus.

## 1.9.2 Problèmes rencontrés

---

### Mise en place de la structure du projet

---

La mise en place de la structure du projet n'était pas réellement un gros problème, mais j'ai vraiment pris beaucoup de temps pour l'effectuer comme il faut. N'étant pas parfaitement à l'aise avec Python Flask, je souhaitais vraiment avoir une très bonne structure de projet. J'ai découvert les "[Flask Blueprint](#)" qui m'ont beaucoup aidé à effectuer une structure propre et découpée.

### Oublie de certains points très importants

---

Un des gros problèmes que j'ai pu rencontrer est que je suis malheureusement passé à côté de plusieurs parties très importantes dans le métier de coach sportif. Cela a engendré de longues réflexions pour l'implémentation ainsi que l'ajout de nombreuses tâches "nécessaires" ainsi que de nombreux ajustements de la base de données. C'est pour cela que dans le [planning effectif](#), on peut retrouver beaucoup plus de tâches que dans le [planning prévisionnel](#).

### Compte Google

---

Un autre problème que j'ai rencontré sur la fin de ce travail de diplôme est que Google ne prend plus en charge les applications "moins sécurisées" depuis le 30 mai. Cette décision a été prise pour améliorer leurs normes de sécurité. Le problème est que j'utilisais cette fonctionnalité pour utiliser le compte Google fitjourney pour envoyer les mails. Désormais, pour utiliser un compte Google pour envoyer des mails, il faut utiliser leurs API pour récupérer une clé qui permet l'accès. Je n'avais malheureusement pas le temps d'implémenter cette solution avec le temps qu'il me restait. C'est pour cela que j'ai utilisé [MailTrap](#) un service 'sandbox' pour email comme solution temporaire.

### 1.9.3 Améliorations possibles

---

J'ai retenu plusieurs points au niveau des améliorations possibles et envisageables pour la suite du projet. Premièrement, cela serait utile d'ajouter la possibilité de transférer la prise en charge d'un client à un autre coach. Cette fonctionnalité peut être nécessaire dans plusieurs cas, comme par exemple si plusieurs coachs travaillent dans la même salle et qu'un coach est débordé, il pourrait assigner des clients à son collègue. Dans le cas où un client change d'objectif et qu'un autre coach est plus adapté à ses besoins aussi.

Un deuxième point qui me paraît très intéressant serait d'ajouter la possibilité de modifier ou supprimer une session. En cas d'imprévu, le coach ne peut pas supprimer ou modifier la date d'une session qu'il a enregistré avec un client.

Un autre point qu'il faudrait rajouter serait l'ajout des photos du client lors de bilan. Les champs sont déjà prévus dans la base de données, il ne manque que les inputs dans le formulaire de bilan.

Il y a également les données d'entrainements qui sont actuellement récupérées avec l'API Polar, cela serait intéressant d'ajouter d'autre montre connectée (Apple Watch, FitBit, Garmin, etc.). Même si Polar est une marque très utilisée dans le milieu, cela reste limitant pour les utilisateurs qui ne possèdent pas d'appareils Polar.

Dans le but de mettre en production l'application, cela serait très intéressant de l'adapter pour téléphone mobile. Il faudrait également séparer les 2 versions de l'application (Coach/Client) qui n'en font qu'une pour l'instant. Comme mentionné plus haut, les [blueprints](#) Flask permettent d'effectuer cela sans trop de soucis.

La dernière amélioration serait de pouvoir sélectionner les valeurs utilisées dans les graphiques montrant l'évolution du poids sur la page profil du client. Cela serait très utile pour le coach et même pour le client d'avoir accès à une courbe montrant l'évolution d'autres données comme la masse grasseuse ou musculaire.

### 1.9.4 Bilan personnel

---

Globalement, je suis très satisfait du travail rendu au bout de ces 9 semaines. Ce projet me tenait particulièrement à cœur et je suis content de l'avancement de ce dernier. En travaillant pendant 9 semaines le projet a bien évolué, beaucoup de fonctionnalités sont venus renforcer l'idée de départ et rendre cette application de plus en plus complète. J'ai pu prendre en main de nouvelles technologies comme Python Flask que j'ai vraiment apprécié. J'ai également appris à utiliser les lecteurs RFID ce que j'ai trouvé très intéressant, malgré une documentation très archaïque de la librairie `pyscard` qui me permet d'utiliser les lecteurs en Python.

## 1.10 Sources

---

- **Polar Flow** : <https://flow.polar.com/>
- **Polar API** : <https://www.polar.com/accesslink-api/>
- **Python Flask** : <https://flask.palletsprojects.com/en/2.1.x/>
- **FullCalendar** : <https://fullcalendar.io/>
- **Flask-Mail** : <https://pythonhosted.org/Flask-Mail/>
- **Flask-Login** : <https://flask-login.readthedocs.io/en/latest/>
- **SQLAlchemy** : <https://www.sqlalchemy.org/>
- **Chart.js** : <https://www.chartjs.org/>
- **LucidCharts** : <https://www.lucidchart.com/>
- **Figma** : <https://www.figma.com/>
- **Pyscard** : <https://pyscard.sourceforge.io/>
- **Argon Design System** : <https://demos.creative-tim.com/argon-design-system/docs/getting-started/overview.html>
- **Python** : <https://www.python.org/>
- **MailTrap** : <https://mailtrap.io/>
- **Ray.so** : <https://ray.so/>

## 2. Logbook

---

### 2.1 Journal de bord - Thomas Fujise

---

#### 2.1.1 Lundi, 04 Avril 2022

---

Premier jour du travail de diplôme, il faudrait réussir à mettre en place l'environnement de travail aujourd'hui. Suite à la présentation du déroulement du TD, je change de salle pour m'installer dans la salle à côté. Je vais prendre contact avec M. Jossi, qui est responsable du suivi de mon projet, pour connaître ses exigences en ce début de projet. Je commence à lister les backlogs identifiables à ce stade sur un Trello. Je réalise en parallèle le planning prévisionnel sur Excel. La journée se termine un peu plus tôt (vers 15h) car nous avons une visite à l'HEPIA.

#### 2.1.2 Mardi, 05 Avril 2022

---

Deuxième jour du travail de diplôme, aujourd'hui il faut que j'avance un maximum sur le planning prévisionnel et l'identification des backlogs. J'ai rajouté les difficultés, la priorité et les dépendances sur les tâches dans le Trello. Cela me permet de mieux m'organiser dans l'exécution de tous les backlogs. M. Garcia est venu vers moi pour définir le nom de mon projet (je l'avais appelé "CoachingTools" par défaut), j'ai finalement décidé de nommer mon projet "FitJourney". Je vais terminer de définir les backlogs avant la fin de la journée.

#### 2.1.3 Mercredi, 06 Avril 2022

---

Aujourd'hui, je vais modifier encore quelques détails sur mon planning prévisionnel avant qu'il soit terminé. Le planning prévisionnel est donc terminé ainsi que l'identification des backlogs (du moins ceux identifiables à ce stade). Je vais commencer le maquettage des interfaces avec Figma. Pour les maquettes je vais essayer de faire quelque chose d'assez simple, pour éviter de prendre trop de temps sur cette tâche. Je n'avais pas beaucoup utilisé Figma auparavant mais j'ai réussi à bien prendre l'outil en main durant la matinée. J'ai pu terminer déjà quelques maquettes (Profil, Login, Register) et bientôt celle du tableau de bord pour coach. J'ai pu avancer sur les maquettes, il ne me manque plus que la maquette de la page pour créer une séance avec un coach. J'ai pris étonnamment beaucoup moins de temps que prévu pour réaliser les maquettes. Je vais donc pouvoir très bientôt commencer à coder. **RAPPEL:** Il faut que je commence la documentation demain pour déjà documenter les maquettes que j'ai réalisées.

## 2.1.4 Jeudi, 07 Avril 2022

---

Aujourd'hui, je vais terminer les maquettes et commencer la documentation technique. Je vais documenter déjà toutes les maquettes que j'ai réalisé pour ne pas avoir à le faire plus tard. J'ai pu terminer les maquettes avant la fin de la matinée. Il faut maintenant que je créer la structure de la documentation technique, je pourrai ensuite commencer à documenter les maquettes que j'ai réalisé.

Suite à une discussion avec un ami coach, j'ai découvert l'existence d'une solution similaire à mon projet [Inithy](#). Inithy propose globalement les mêmes fonctionnalités que FitJourney à l'exception que leurs application est orienté coaching à distance là où moi je m'oriente plus sur le coaching en salle (d'où les cartes de membres RFID). Inithy est toujours au stade de démo, il faut les contacter pour avoir un accès à leurs application qui n'est pas encore ouvert à tous.

Je vais leurs envoyer une demande pour essayer d'avoir accès à leurs application et pouvoir analyser leurs application un peu plus en détail.

J'ai pu commencer la documentation des maquettes que j'ai réalisé. J'ai eu une réflexion sur l'enregistrement d'un nouveau coach, j'envisage deux cas possible :

- Soit l'application est délivré avec un compte coach(admin) et le formulaire d'enregistrement n'est disponible que lorsqu'on est authentifié en tant que coach(admin) (C'est le coach qui rempli le formulaire pour les nouveaux clients)
- Soit rajouter un champ lors de l'inscription avec un code pour créer un nouveau compte coach.

## 2.1.5 Vendredi, 08 Avril 2022

---

Fin de la première semaine sur ce travail de diplôme, j'ai pu commencer la documentation et documenter les maquettes que j'ai réalisé. Suite à ma réflexion sur l'enregistrement d'un nouveau coach, j'ai décidé de simplement rajouter une option sur le formulaire d'enregistrement pour créer un compte coach. Si une personne venait à créer un compte coach sans l'être ce ne serait pas très dérangeant car un coach n'a accès qu'aux comptes des clients qu'il suit et pour suivre un client il faut que le client confirme de son côté.

Pour un client 2 manières d'avoir un compte :

- Par le formulaire d'enregistrement au préalable, le coach n'aura qu'à sélectionner le profile client lors de la prise en charge.
- Par le coach (un coach peut créer un compte lors de l'ajout d'une prise en charge d'un client, le compte sera créé avec les infos basiques qui seront demandées (Nom, prénom, date de naissance, adresse) et un mot de passe est généré pour permettre au client de se connecter plus tard)

J'ai ajouté la structure de l'application et initialisé les blueprints. (Les blueprints permettent de séparer l'application en plusieurs modules qui sont ensuite importé au même endroit)

J'ai créer un fichier de config qui permet d'avoir 2 mode :

- Debug
- Production

Qui permet d'avoir des paramètres différents. J'ai également ajouté un fichier run qui permet de lancer l'application. Il charge la configuration correspondant au mode actuel (Debug/Production) et run l'application Flask.

## 2.1.6 Lundi, 11 Avril 2022

---

Début de la deuxième semaine du travail de diplôme, aujourd'hui je vais commencer mes pages HTML. Je pense que cela risque de prendre un peu de temps mais en tout cas les maquettes que j'ai réalisé vont me faire gagner pas mal de temps. J'ai réussi à créer une structure de base pour les fichiers html. J'ai décidé d'utiliser [Sneat](#) qui est un thème css open-source publié par [ThemeSelection](#) sous licence MIT. J'ai ajouté un fichier HTML pour les liens CSS qui sera inclus dans toutes mes pages.

M. Maréchal est venu voir l'avancer du projet car il va superviser mon travail de diplôme de "loin". M. Jossi reste mon suiveur principal. M. Maréchal m'a conseillé de faire un diagramme explicatif du fonctionnement de l'application.

J'ai ajouté la route par défaut pour pouvoir tester une page index avec le css (pour vérifier que tout marche bien). Pour l'instant, je n'ai pas de problème tout à l'air de fonctionner comme il faut. J'ai fait une class "prototype" Users avec SQLAlchemy pour empêcher des erreurs bloquantes avec Flask-Login.

Flask-Login est une librairie qui permet de gérer les sessions utilisateurs pour Flask. (Flask-Login gère les connexions, déconnexions et garde la session utilisateur pour savoir l'utilisateur connecté ou qui vient de se déconnecter)

J'ai maintenant une page index qui peut afficher des composants à l'aide du thème CSS Sneat. Demain je poursuivrai la création de mes pages HTML et il faut également que je commence à rajouter des éléments dans la documentation

### **Ne pas oublier d'avancer la documentation**

## 2.1.7 Mardi, 12 Avril 2022

---

Aujourd'hui je vais essayer de terminer la partie authentification de l'application (Login/Register). J'utilise FlaskForm pour créer mes formulaires et je pense utiliser Mashmallow pour valider les champs en relation avec la BD (Je regarde quelques exemples d'utilisation avec SQLAlchemy : [Exemple](#)).

Il faut également que je fasse le diagramme qui m'a été conseillé par M.Maréchal pour la description du fonctionnement de l'application.

Pour la sécurité des mots de passe j'ai déjà rajouté 2 méthodes pour permettre de hasher et de vérifier les mots de passe. J'utilise la librairie Python hashlib qui me permet d'hasher les mots de passe avec un salt. Pour éviter d'enregistrer le salt en base, le salt est hashé en sha256 et est placé avant le mot de passe dans la chaîne. Comme un sha256 a toujours 64 caractères, pour vérifier le mot de passe je saute les 64 premiers caractères de la chaîne pour pouvoir comparer avec la saisie de l'utilisateur.

J'ai pu faire un diagramme explicatif des différents processus lors de l'utilisation de mon application en fonction de notre rôle (Client, coach)

### Image

J'ai également terminé le login (le register ne devrait pas prendre beaucoup de temps). J'ai eu quelques soucis pour inclure les formulaires avec FlaskForm (avec les imports python). J'ai ajouté une navbar pour mes pages qui servira pour la navigation dans l'application. J'ai juste encore quelques soucis avec le css mais ce n'est pas du tout prioritaire pour le moment.

J'ai regardé pour utiliser un outil de génération de documentations, je pense utiliser Pdoc qui à l'air assez complet et qui permet de générer la documentation sur les librairies incluses dans le projet.

## 2.1.8 Mercredi, 13 Avril 2022

---

Dernier jour avant les vacances de pâques, j'essaye d'intégrer pdoc à mon projet mais lorsque je lance la commande pour générer le doc j'obtiens une erreur. Il n'arrive pas à trouver apps qui est le dossier principal.

J'ai pris un peu de retard par rapport à ma planification initiale mais j'ai avancé sur des points que j'étais censé effectuer plus tard dans le projet donc au final je n'ai pas beaucoup de retard. Je vais juste terminer encore ma page register qui va de pair avec la page Login. Il faudra ensuite que je commence à documenter tous le back-end que j'ai déjà effectué. Je pourrai ensuite commencer à travailler sur la base de données car pour le moment j'utilisais une base de données temporaire pour tester mon login.

En faisant des tests sur le login je me rends compte que j'ai une erreur qui s'affiche lors de la connexion. Lorsque j'effectue la requête pour voir si le mail que l'utilisateur a saisi existe je ne rencontre aucune erreur si le mail inséré n'est pas dans la base mais lorsque le mail rentré se trouve dans la base de données alors une erreur s'affiche :

```
string argument without an encoding
```

Après quelques heures d'incompréhension sur cette erreur, je viens d'en trouver la raison. J'avais seulement mis le mauvais type dans l'initialisation du champ mot de passe avec SQLAlchemy.

J'ai enfin finis le register et le login, tout fonctionne comme il faut. Je n'ai plus qu'à rajouter les champs que je veux dans le register (tout le back-end est fonctionnelle). Je vais commencer à documenter les points que j'ai avancé jusque là je pourrai ensuite commencer à travailler sur la base de données.

J'ai pu commencé à documenter l'environnement de mon projet mais il reste encore beaucoup de point que je peux documenter déjà maintenant. Pour avancer la documentation en même temps que le projet je vais faire ça en priorité et je continuerai sur la base de données une fois la documentation à jour.

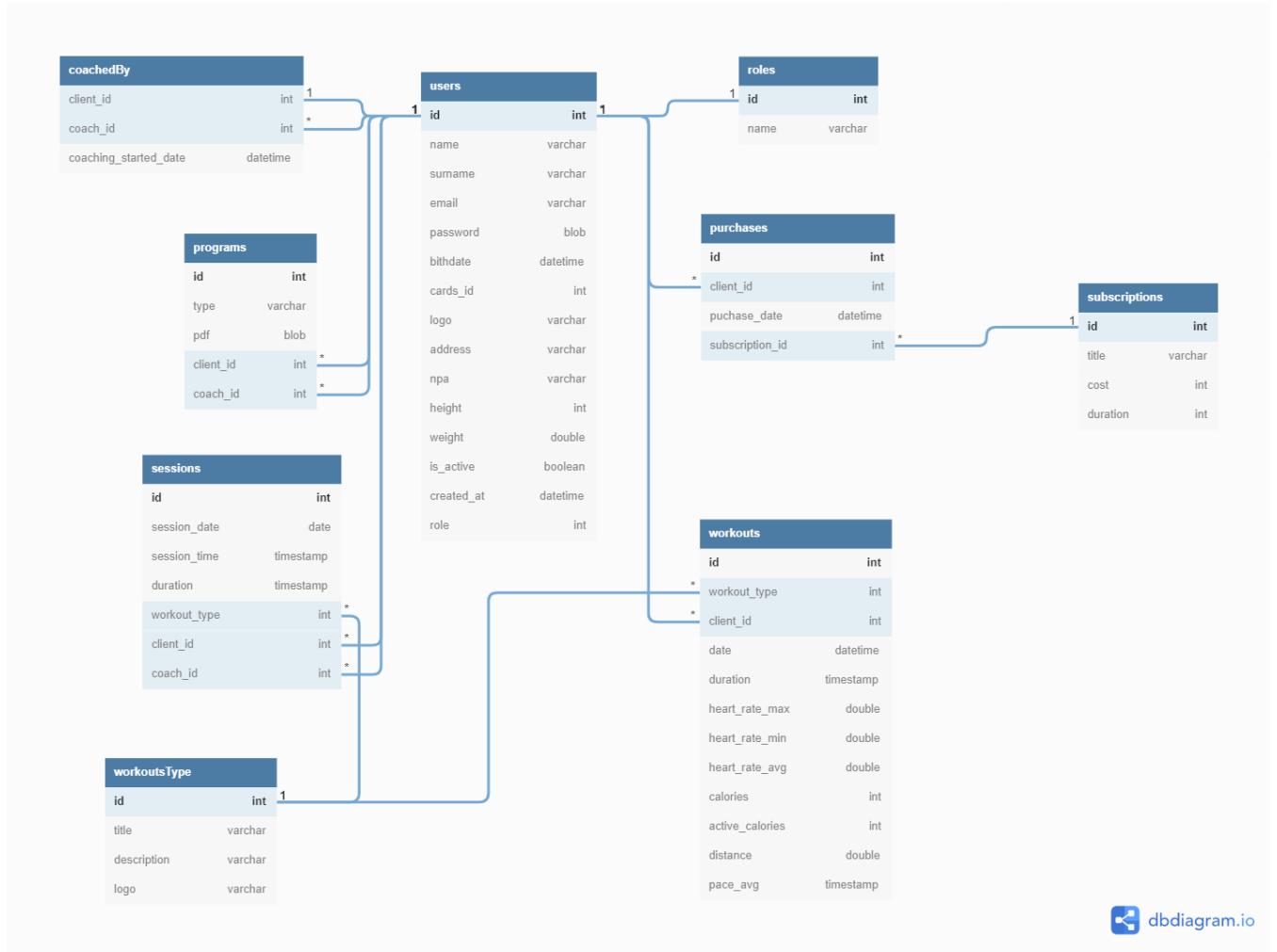
## 2.1.9 Lundi, 25 Avril 2022

---

Retour de vacance, je n'ai malheureusement pas pu énormément avancer pendant la semaine et demie de vacance. Le premier rendu intermédiaire a lieu demain. Je vais terminer le diagramme UML que j'ai commencé pour la base de données. J'utilise l'application WEB [dbdiagram.io](https://dbdiagram.io) qui permet de créer un diagram et de l'exporter directement en SQL par la suite.

Ma machine a eu un blue screen qui a été très long (30min) à redémarrer. Je n'ai malheureusment pas eu le temps de sauver le diagram que j'avais quasiment terminé, je vais devoir recommencer.

J'ai pu terminer le diagramme UML à l'aide de dbdiagram.io :



Je peux maintenant commencer la création des tables avec SQLAlchemy. J'ai découvert un outil qui est actuellement en BETA [dbdocs](#) qui permet de générer une documentation WEB d'une base de données. Je regarderai l'outil un peu plus en détail plus tard mais je pense que cela pourrait être une bonne idée d'avoir cette documentation en plus de celle qui sera présente dans le document de documentation technique.

Je viens de me rendre compte que je n'avais pas créé le planning effectif pour l'instant. J'ai dupliqué le planning prévisionnel et effectué les quelques modifications pour le planning effectif.

M.Jossi est venu voir l'avancement du projet. Nous avons discuté de plusieurs points notamment le diagramme que j'avais fait pour expliquer l'utilisation de l'application. Le diagramme que j'ai fait était un mix d'un sitemap et d'un diagramme de cas d'utilisation. Je vais donc les refaire séparément. Je vais également faire un MCD pour être sûr de ne rien oublier pour ma base de données. Je pourrai ensuite poursuivre la création de ma base avec SQLAlchemy.

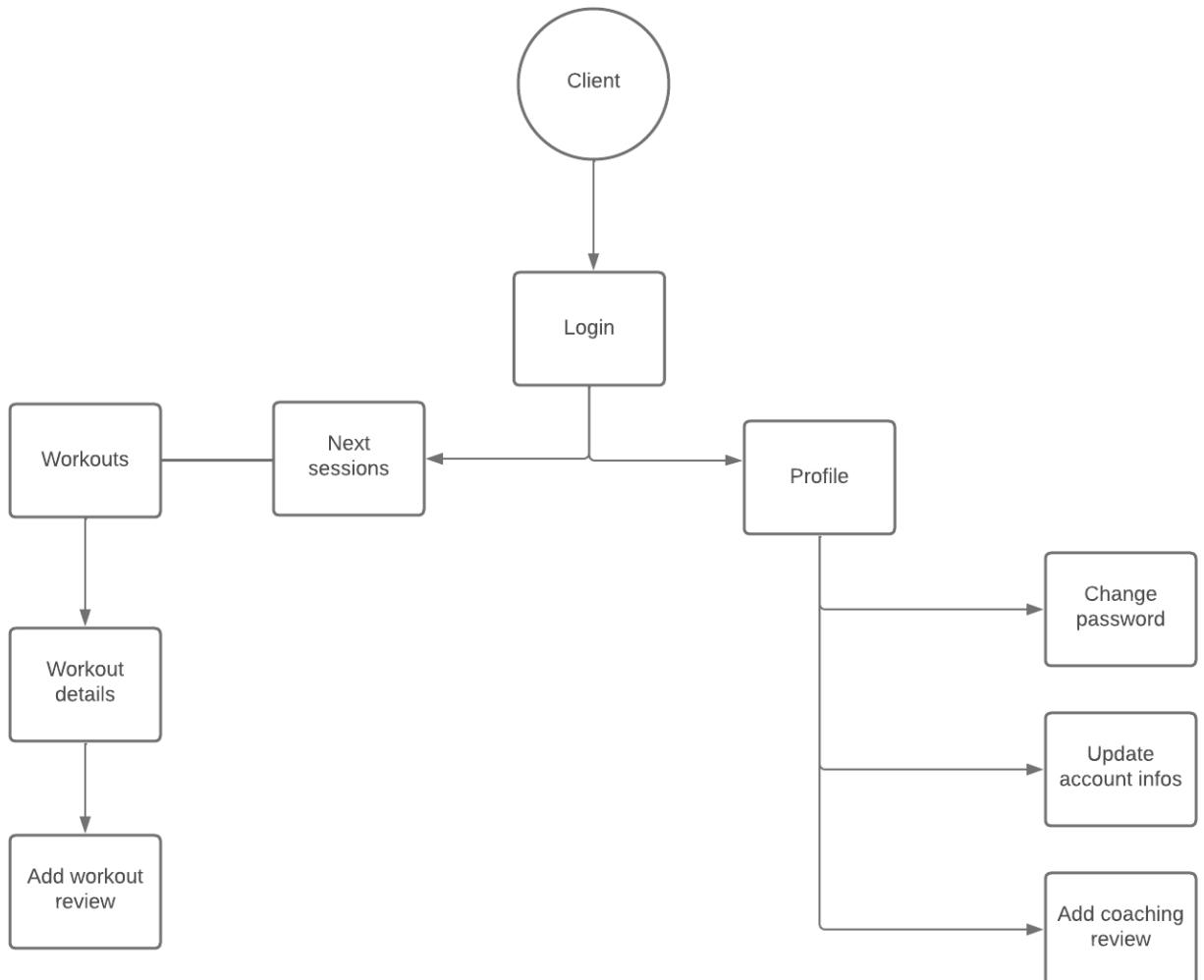
Pour l'évaluation intermédiaire, M.Jossi m'a demandé de remplir la grille d'évaluation de mon côté et nous referons un point mercredi pour voir si tout est ok.

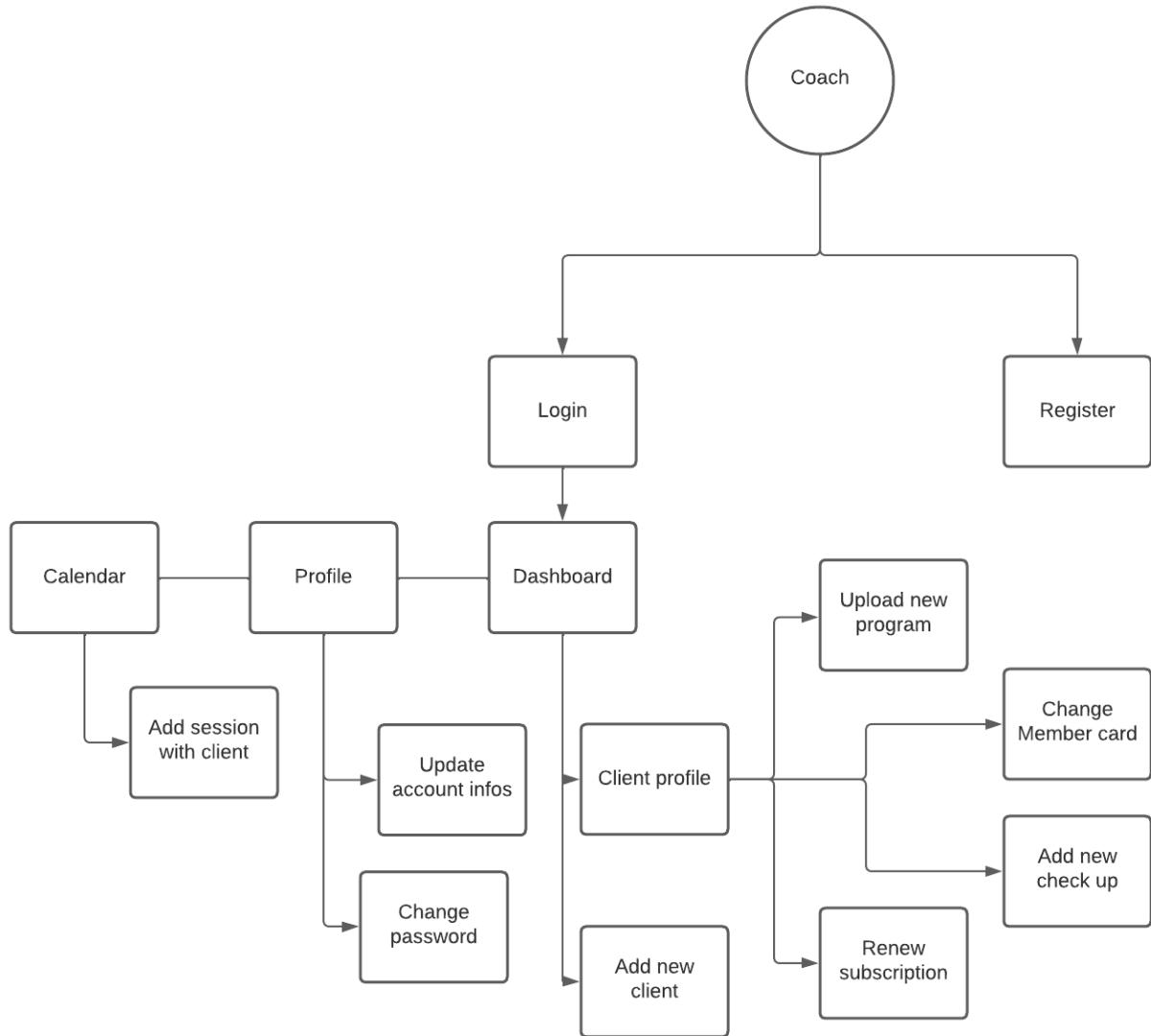
## 2.1.10 Mardi, 26 Avril 2022

---

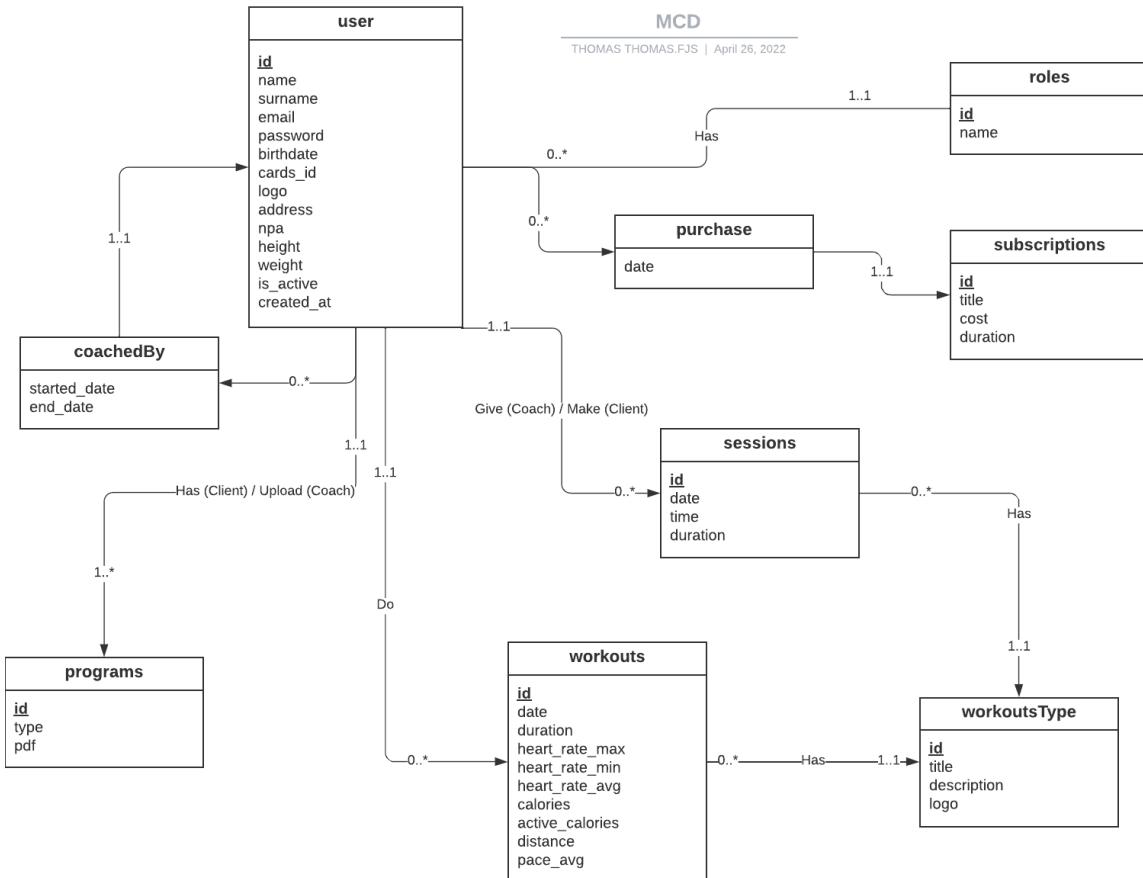
Aujourd'hui je dois compléter la grille pour la première évaluation intermédiaire comme convenu avec M.Jossi.

J'ai pu terminer les 2 sitemaps pour le client et le coach.





Je vais maintenant faire le MCD que j'aurai dû faire avant le MLD pour être sûr de n'avoir rien oublier. Il faudrait que j'avance la documentation sur les diagrammes que je suis en train d'effectuer. J'ai pu terminer le MCD, je le ferai vérifier par M.Jossi car j'ai quelques doutes sur certains points. Je ne sais pas si, dans mon cas, il n'est pas mieux de faire une classe mère (Users) avec 2 enfants (Clients et coachs) pour le MCD.



J'ai ajouté les objets SQLAlchemy en fonction du MCD que j'ai fait au préalable. Je me suis rendu compte que certaines tables devront être remplies au préalable ou je devrais rajouter des écrans pour pouvoir les remplir. Un coach n'a pas moyen d'ajouter un type d'entraînement (la table 'workoutsType') ni d'ajouter un type d'abonnement (table subscriptions). Les relations ont l'air de fonctionner avec SQLAlchemy je vais pouvoir adapter les formulaires de register et login que j'ai fait au préalable pour tester.

J'ai mis également à jour le Trello que je n'avais plus touché depuis quelques jours.

## 2.1.11 Mercredi, 27 Avril 2022

M.Jossi devrait passer aujourd'hui à la pause pour faire un point et faire l'évaluation intermédiaire comme convenu.

Je vais adapter les formulaires de login/register à la nouvelle base de données. Je viens de me rendre compte que le champ 'DateField' dans mon formulaire, que j'utilise pour renseigner la date de naissance de l'utilisateur s'affiche comme un input texte basique. Pour avoir un input de type date j'ai utilisé le champ 'DateField' mais depuis `wtforms.fields.html5`.

J'ai rencontré un nouveau problème, en premier les validations des champs du formulaire ne s'effectuait pas correctement. J'avais en fait simplement oublier d'utiliser un SubmitField (j'utilisais un input HTML en dur à la place). Les validateurs fonctionnent sauf ceux du mot de passe qui sont sensés vérifier que les 2 mots de passes saisient soit identiques. Le formulaire se valide et prend en compte uniquement le premier mot de passe saisit.

J'ai trouvé pourquoi les mots de passe ne se vérifiaient pas, je passais directement à la création de l'utilisateur lors du clique sur le bouton 'register'. Les champs se vérifiaient "eux-même" par rapport au type de données saisient mais le check si les 2 inputs étaient identiques ne s'effectuaient donc pas. J'ai donc changé ma condition pour que je poursuive la création de l'utilisateur uniquement lors de la validation du formulaire.

J'ai réussi à adapter le login et le register avec les bons champs, il ne manque plus qu'a ajouter l'affichage des erreurs.

M.Jossi est passé pour l'évaluation intermédiaire. Globalement tout est ok, j'ai juste pris un léger retard avec toute la mise en place du système que j'avais légèrement sous-estimé. Pour la documentation, j'ai quelques points a ajouter/modifier :

- Ajouter les sitemaps
- Modifier le MCD (Pas de flèches, revoir les cardinalités, double relations pour coach/client)
- Ajouter Installation pour le projet en général (pas seulement Python Flask)
- Ajouter l'aborescence du dossier source (explication détaillé)
- Revoir l'ordre des maquettes (ordre logique)
- Explication des boutons sur les maquettes
- Use case dans l'analyse fonctionnelle
- Schéma pour montrer comment l'application va intéragir avec l'API

J'ai pu terminé la modification du MCD, je vais l'envoyer à M.Jossi pour qu'il puisse vérifier. Il faut maintenant que j'avance sur la documentation.

## 2.1.12 Jeudi, 28 Avril 2022

---

J'ai commencé à revoir la description de mes maquettes. Je me suis rendu compte de quelques incohérence sur certaines maquettes comme sur le tableau de bord du coach, la prochaine session était affiché tout en bas de la page. Il était donc plus cohérent d'afficher cette information tout en haut et de mettre la liste de client en bas.

J'ai également eu une réflexion sur les bilans. J'avais totalement oublié de penser au bilan qui est un élément essentiel du coaching. Je vais donc revoir les maquettes et je pourrai donc modifier le MCD quand

je retournerai dessus plus tard. Je pense rajouter 2 "types" de bilan, un bilan *général* qui sert à évaluer le ressenti du client par rapport à sa prise en charge, le suivi de manière général. Le deuxième type de bilan serait un bilan par session (qui ne serait pas obligatoire, au début cela peut ne pas déranger mais si le client effectue beaucoup de séance cela peut vite devenir répétitif) où le client note son ressenti sur la séance. Pour le coach, un bilan est effectué de manière soit hebdomadaire, soit mensuelle pour enregistrer les nouvelles mesures (poids, mensu, chrono, cela dépend du type de coaching).

Pour le bilan, les données que le coach devra renseigner à l'aide d'une balance connecté qui permets leurs acquisition sont :

- Poids
- BMI (Body Mass Index / IMC)
- Body Fat %
- Water %
- Muscle Mass %
- Bone Mass
- BMR
- Fat Visceral
- Lean Body Mass
- Body Fat Mass
- Muscle Mass
- Protein
- Body Age

J'ai pu terminé les nouvelles maquettes, j'ai rajouter une page bilan de session pour l'utilisateur ainsi qu'une page bilan "général" pour noter la qualité du coaching. Du côté coach, j'ai rajouté la page pour effectuer le bilan général avec les informations mentionnées au-dessus.

## 2.1.13 Vendredi, 29 Avril 2022

---

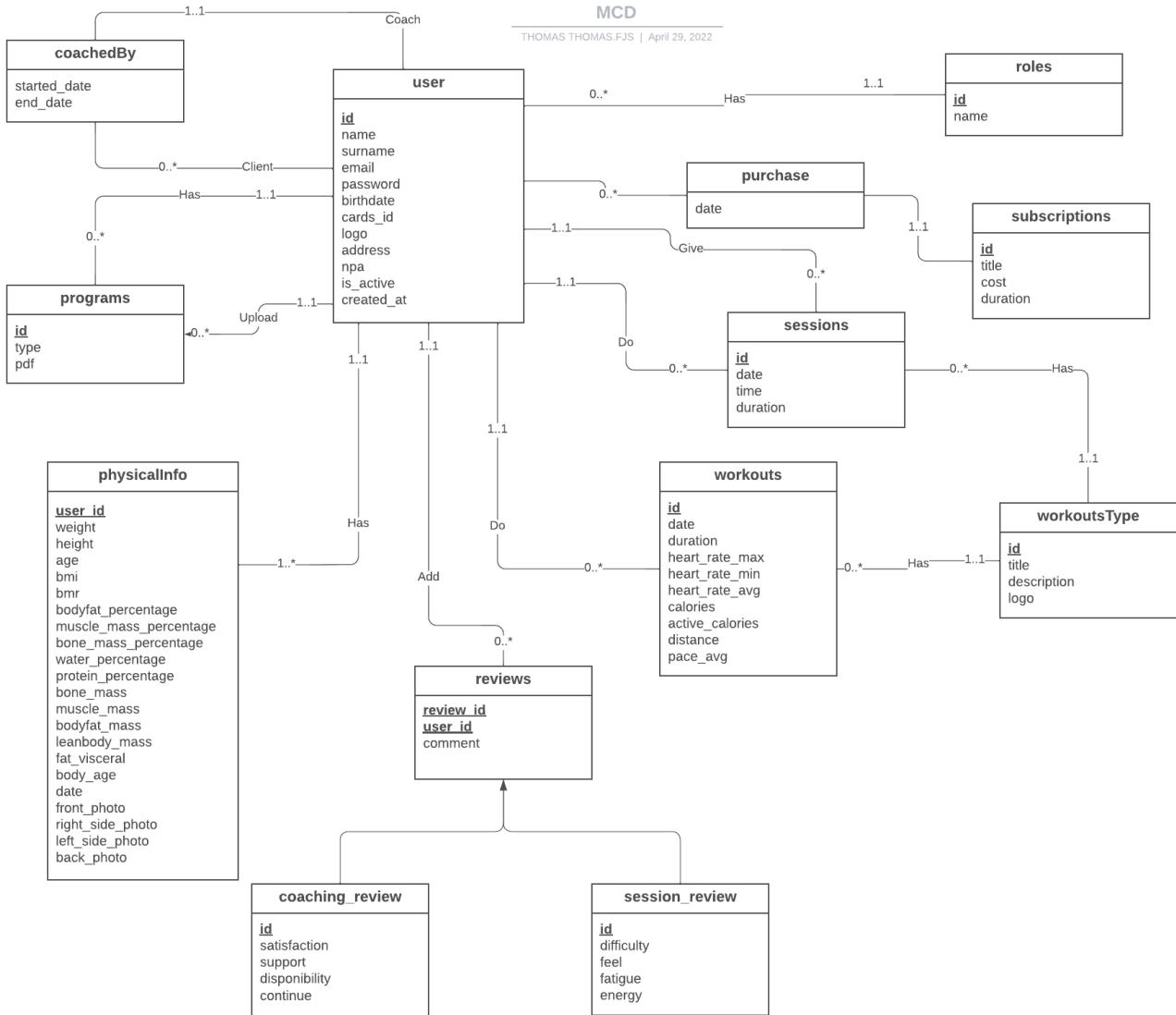
Maintenant que j'ai rajouté les bilans et les avis je vais revoir le MCD. Je décide de séparer les informations physiques de la table utilisateur et d'ajouter une date dans la table information physique pour avoir un historique de l'évolution. Il faut également que j'ajoute 2 tables pour les avis (Coaching et sessions).

Au niveau MCD, je pense que les 2 tables avis doivent être reliées à une table mère "Reviews" qui serait associée à la table *Users*. Je demanderai à M.Jossi lorsque je lui enverrai mon MCD pour qu'il puisse vérifier et me donner son avis.

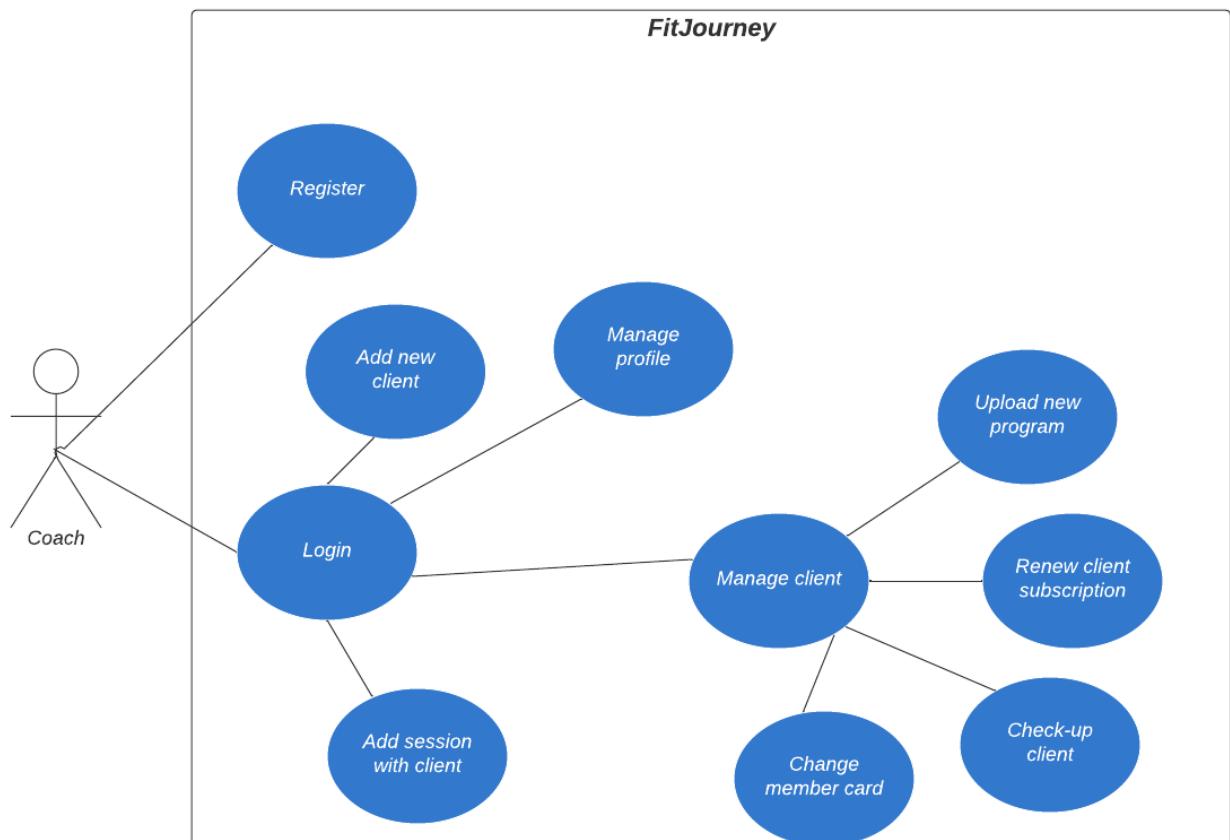
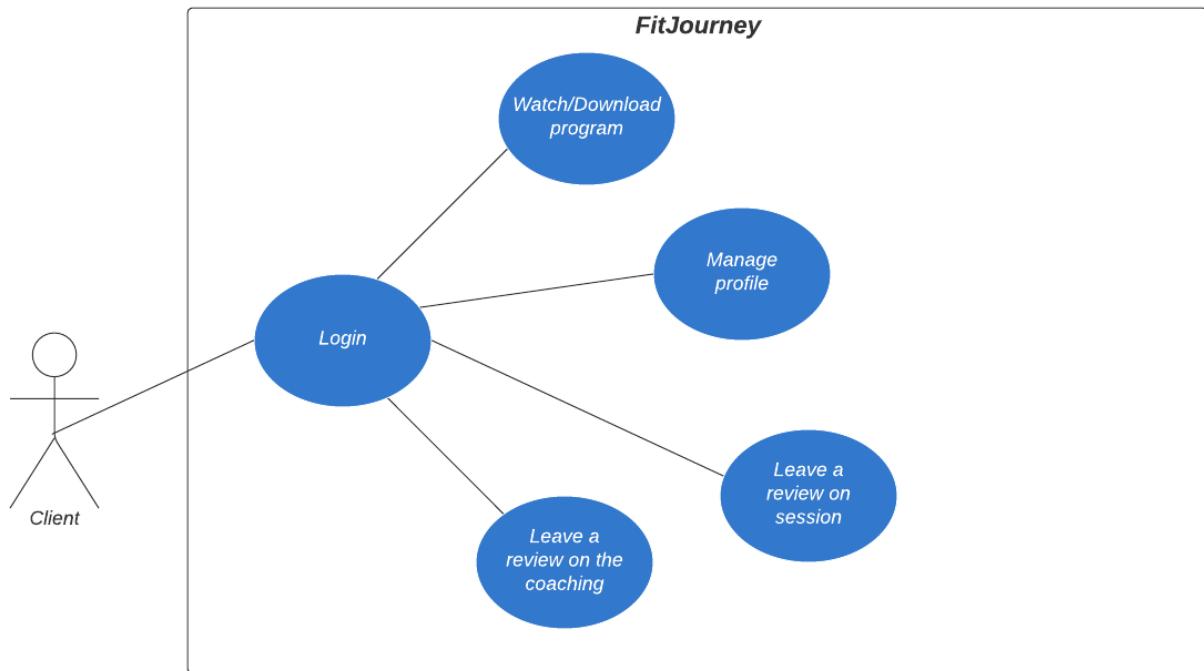
En utilisant une table mère pour les avis, je dois remonter le champ "commentaire" car c'est un champ commun entre les 2 tables avis. (Les champs d'évaluation pour les avis ne pourront donc pas être modifié sachant qu'ils représentent des champs définis dans la base de données)

Je vais encore revoir les cardinalités du MCD et je pourrai ensuite l'envoyer à M.Jossi.

Voici le nouveau MCD :



J'ai revu également les cas d'utilisations que j'avais réalisé, j'ai décidé de faire 2 schéma différent pour le client et le coach.



J'ai pu avancer sur la documentation, il faut encore que :

- J'ajoute les navbars dans la section maquette
- J'explique plus en détail l'arborescence de l'application
- J'ajoute l'explication de l'installation générale du projet
- J'explique les 2 sitemaps
- J'ajoute le MCD
- J'ajoute un schéma pour expliquer l'utilisation de l'API par l'application.

## 2.1.14 Lundi, 2 Mai 2022

---

Aujourd'hui je vais continuer d'avancer sur la documentation et je vais également reprendre le code maintenant que j'ai un MCD plutôt correcte. M.Jossi a pu regarder la nouvelle version du MCD que je lui avais envoyé. Je dois refaire quelques modifications pour respecter les standards Ecole (nom des entités en majuscule singulier) et mettre les verbe d'association à l'indicatif si possible.

M. Jossi est passé me voir, il m'a aidé à éclaircir le passage du MCD au MLD avec la table mère *REVIEW*. Nous avons décidé de ne pas prendre la table mère et de garder uniquement les tables filles car il n'y avait qu'un seul champ en commun.

J'ai du retoucher encore une fois mes maquettes pour revoir les navbars, j'ai également refait le sitemap du côté coach car j'avais oublié quelques chemins. J'ai rajouté la documentation sur les navbars de l'application.

J'ai implémenté le MCD avec SQLAlchemy pour créer les tables.

J'ai rajouté des routes pour afficher une page d'erreur en fonction de l'erreur qui survient :

- 403 pour un accès non autorisé
- 404 si la page n'est pas trouvée
- 500 si c'est un problème avec le serveur

J'ai également créé une route dynamique pour tous les templates dans le dossier templates/home, toutes les pages sont accessibles uniquement si l'utilisateur est connecté (@login\_required)

Je rencontre un problème, je n'arrive pas à accéder aux pages car même une fois le login passé le login manager ne détecte pas la connexion (Il agit comme si l'utilisateur n'était pas connecté)

## 2.1.15 Mardi, 3 Mai 2022

---

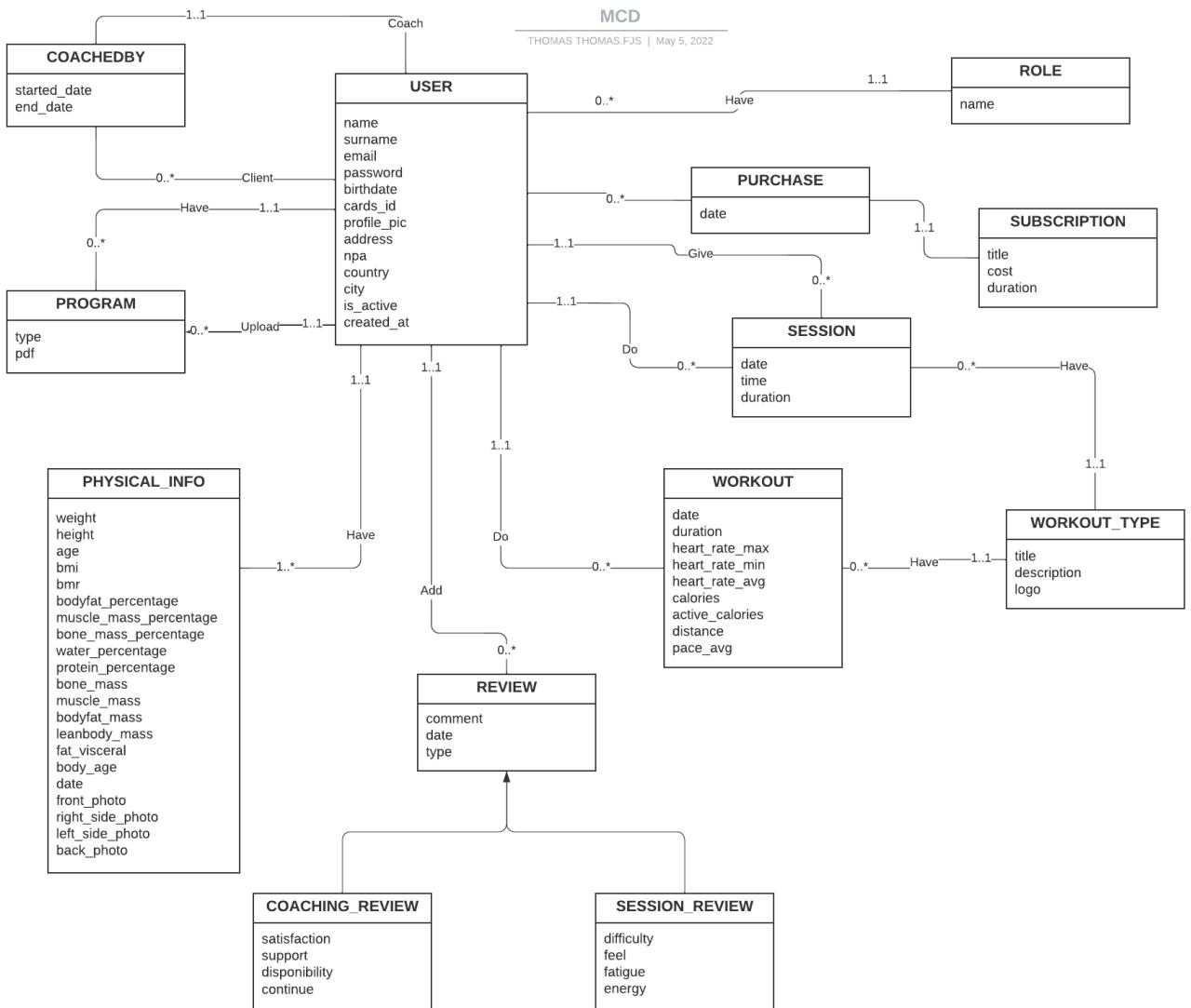
J'ai continué de regarder mon problème avec le login\_manager hier soir, je n'ai toujours pas trouvé de solution.

Ce matin en relisant la documentation de Flask Login, j'ai essayé d'ajouter les méthodes que la classe User doit implémenter pour que le Login manager fonctionne. Cette fois le Login manager fonctionne avec les méthodes implémentées à la main, le problème est que pour éviter de devoir les implémenter à la main j'avais fait hériter ma classe User d'*UserMixin* qui permet l'implémentation par défaut des propriétés et méthodes nécessaires.

Donc je ne sais pas pourquoi en faisant hériter ma classe de *UserMixin* cela ne fonctionne pas mais quand j'implémente les propriétés et méthode à la main le Login manager fonctionne.

[Documentation Flask Login](#)

J'ai très rapidement retouché le MCD pour enlever encore les id qui ne sont pas obligatoire et j'ai juste rajouter un champ *City* et *Country* pour la table USER.



J'ai commencé à avancer sur la page profil du client.

A la base j'avais séparé l'application en 2 gros dossiers :

- authentication/
- home/

`home/` était sensé contenir toute l'application hormis l'authentification, je pense séparer ce dossier en 2 parties :

- client/
- coach/

Je vais alors avoir quelques fichiers en double mais ils seront bien précis par rapport à leurs utilités.

Chaque dossier doit contenir les fichiers : *routes.py*, *forms.py*, *init.py* qui seront bien sûr différents pour chaque dossier.

J'ai pu terminer la base de la page profil, les informations du client sont affichés et peuvent être modifiés. Il faut encore que je rajoute le "widget" pour les retours ainsi que l'option pour changer le mot de passe.

## 2.1.16 Mercredi, 4 Mai 2022

---

J'ai pu ajouter les requêtes SQL pour afficher les données sur la page profil comme la fin de l'abonnement qui a été souscrit en dernier.

Il faut que je commence le Poster car le rendu est fixé à Lundi et je dois également poursuivre l'avancement de ma documentation pour le rendu intermédiaire qui a lieu Mardi.

Les points que je dois encore ajouter :

- Expliquer plus en détail l'arborescence de l'application
- Ajouter l'explication de l'installation générale du projet
- Détails sur les 2 sitemaps
- Remplir la section Base de données
- (Ajouter un schéma pour expliquer l'utilisation de l'API par l'application.)

Je peux éventuellement aussi commencer à documenter quelques fichiers comme les routes ou encore les blueprints que j'utilise.

J'ai pu avancer sur mon poster, M.Bonvin nous a proposé de passer demain pour voir les posters et donner un retour.

Pour revenir à l'application, je dois faire une requête SQL pour récupérer les retours du client. Le problème est que les retours sont dans 2 tables différentes. J'ai cherché pendant un petit moment pour trouver une solution, et je me suis rappelé des UNION en SQL qui me permettent exactement de faire ce dont j'ai besoin. Le seul problème est que les champs gardent le nom du premier SELECT.

Donc lorsque je récupère les retours client d'un client en particulier, je les récupère dans un objet *CoachingReview* même si les données proviennent de la table *SessionReview* (Car le premier SELECT de l'UNION est celui de *CoachingReview*). Il faut que je trouve comment rajouter des AS dans ma requête SQL.

## 2.1.17 Jeudi, 5 Mai 2022

---

J'ai trouvé le moyen de renommer les champs avec la méthode `label()`. J'ai donc renommé les champs du premier SELECT pour avoir des propriétés plus génériques. Je peux maintenant terminer l'affichage des retours client.

Finallement, je vais revoir la base de données pour optimiser la récupération des retours. Actuellement, il n'est déjà pas possible de savoir de quel type est le retour car pas de champs et pour savoir quel coach était assigné au client il faut vérifier dans 2 autres tables pour la date du coaching etc. Je pense finalement garder la table intermédiaire dans le MLD, une table "REVIEW" qui comporte les points communs entre les 2 (client\_id, coach\_id, date, comment, type) et garder les 2 tables fille qui comporteront les champs spécifiques à chacune. J'ai mis l'ID d'une review dans la table `REVIEW` et cet ID sera la clé primaire et étrangère des tables filles.

Une fois la table intermédiaire ajoutée, je pourrai terminer l'affichage des retours ainsi que la l'affichage de leurs détails.

Je suis bloqué sur un problème, je n'arrive pas à joindre une table récupérer à l'aide d'un `UNION` avec SQL Alchemy.

Voici la requête en SQL :

```
SELECT R.id, R.comment, R.date , R.type, I.Field1
FROM REVIEW AS R
JOIN (SELECT C.id AS ID, C.satisfaction AS Field1 FROM COACHING_REVIEW AS C UNION SELECT S.id, S.difficulty
FROM SESSION_REVIEW AS S) AS I ON I.ID = R.id
WHERE id_client = 1
```

J'arrive à faire l'`UNION` avec tous les champs, seulement lorsque je veux join la table à une autre table je n'arrive pas à déterminer la jointure (le `ON`) car je ne peux pas accéder aux colonnes de la table "UNION".

La première requête qui va chercher toutes les reviews de coaching :

```
q1 = db.session.query(CoachingReview.id, CoachingReview.satisfaction.label("Field1"),
CoachingReview.support.label("Field2"), CoachingReview.disponibility.label("Field3"),
CoachingReview.is_continuing.label("Field4"))
```

La deuxième requête qui va chercher toutes les reviews de sessions :

```
q2 = db.session.query(SessionReview.id, SessionReview.difficulty, SessionReview.feel, SessionReview.fatigue,
SessionReview.energy)
```

L'union entre les deux :

```
q3 = q1.union(q2)
```

Et la requête qui relie les infos de la table mère `REVIEW` aux données récupérés dans l'`UNION` :

```
query = db.session.query(Review.id, Review.comment, Review.date,
Review.type).join(q3,Review.id==q3.c.id).filter(Review.client_id==id)
```

La variable *q3* qui contient le résultat de l'UNION est sensé posséder les colonnes de cette table accessible depuis *q3.c* mais j'ai l'erreur :

```
AttributeError : id
```

En debugant les objets SQL Alchemy, j'ai trouvé pourquoi je n'arrivais pas à accéder aux champs. Les champs avaient un alias généré automatiquement. Pour régler ce problème, j'ai rajouter un nouvel alias à chaque champs et désormais tout fonctionne.

Les reviews du clients sont maintenant affichés sur sa page profil, je peux maintenant ajouter la page de détails lorsqu'on clique sur la review.

Demain je vais beaucoup travailler sur la documentation car le rendu intermédiaire à lieu lundi et il faudrait que j'ai documenté toutes les fonctionnalités déjà effectué.

## 2.1.18 Vendredi, 6 Mai 2022

---

Je viens d'ajouter le changement d'image de profil. Sachant que l'application n'est pas destinée à contenir des milliers d'utilisateurs, j'ai créé un dossier "profile/" dans le dossier "img/" de l'application. Toutes les photos de profil seront enregistrées ici avec un UUID comme nom de fichier. Le nom de fichier est enregistré dans la base de données.

J'ai avancé un peu sur la page Workouts, il faut que je modifie la création des champs de la table 'WORKOUT' car les valeurs sont arrondies et n'ont pas de décimales.

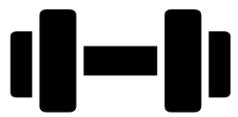
J'ai pu terminer l'affichage des workouts effectués par le client. Cet après-midi, je vais avancer sur la documentation technique et le poster.

Pour le moment, il me manque encore l'ajout des reviews par le client ainsi que l'affichage détaillé des reviews et des workouts et l'affichage des graphiques avec Chart.JS. Une fois cela terminé, j'aurai terminé le côté client.

La partie des graphiques avec Chart.JS sera je pense que la partie la plus "compliquée". Pour le reste de l'application, tous le côté coach est uniquement composé de formulaire et des mêmes pages que le côté client mais avec des options supplémentaires.

J'ai pris la décision de laisser un peu de côté l'aspect "esthétique" de l'application car n'étant vraiment en avance je veux me concentrer principalement sur les fonctionnalités de l'application.

J'ai pu avancé un peu sur la documentation de la base de données. Je vais devoir revoir mon poster ce week-end, M.Bonvin m'a fait un retour : Je n'explique pas ce que mon application fait et je fais plus de pub pour Polar qu'autre chose.



FITJOURNEY

**POLAR**<sup>®</sup>  
WATCHES

START TODAY!

TAKE YOUR COACHING TO  
THE NEXT LEVEL



## 2.1.19 Lundi, 9 Mai 2022

---

Aujourd'hui à 17h, nous avons le rendu intermédiaire. Nous devons rendre le poster, le résumé et l'abstract ainsi que l'état actuel de la documentation technique.

J'ai pu refaire un nouveau Poster plus explicit de ce qu'est mon application :



## COACHING APP | FITNESS TRACKER

THOMAS FUJISE



POLAR®



Je l'ai montré à M.Garcia qui m'a fait la remarque qu'il n'y avait pas vraiment d'ordre de procéder pour l'application (Trop d'images misent un peu partout sans sens particulier). Je vais donc reordrer les images et je devrai être pas mal.

J'ai pu avancer ma documentation technique un peu, il faut encore que j'ajoute :

- L'arborescence
- L'installation

et que je commence à expliquer les différentes routes de l'application.

Je viens de rencontrer un problème, lors du rendu en PDF avec Mkdocs, j'ai plusieurs images qui s'affichaient sur le site statique mais qui ne sont pas visible sur le PDF.

M.Jossi est passé pour voir l'avancement du projet, nous ferons un point pour l'évaluation intermédiaire Mercredi. Il m'a également fait la remarque que mon poster contenait trop d'images un peu partout. Je vais épurer mon poster et continuer d'avancer sur la documentation.

J'ai pu ajouter l'arborescence (qui n'est pas terminée vu que le projet ne l'est pas encore).

Il faudrait encore que j'ajoute des tests utilisateurs dans la documentation pour les parties déjà terminées.

J'ai pu modifier et alléger mon poster :



## COACHING APP | FITNESS TRACKER

THOMAS FUJISE



POLAR®



## 2.1.20 Mardi, 10 Mai 2022

---

Aujourd'hui je vais avancer sur mon projet. Il faut que j'ajoute l'affichage des détails d'une review lorsque l'utilisateur clique dessus.

Je rencontre un problème, je voulais à la base passer la review (objet) en paramètre GET pour pouvoir récupérer les infos sur la page review. Mais lorsque je passe l'objet en paramètre il arrive de l'autre côté sous forme de chaîne (string). Je ne peux donc pas récupérer les données comme je le souhaite car seul les valeurs sont dans la chaîne, il n'y a pas les indexs.

Je vois deux possibilité :

Soit je passe l'id en paramètre et je vais refaire une requête pour chercher les infos (je vais devoir en faire deux car je ne sais pas savoir le type de la review à l'avance et en fonction du type la requête est différente)

Soit j'arrive à récupérer un objet en paramètre.

Je vais essayer de trouver un moyen de passer l'objet mais si je perds trop de temps je ferai l'option avec les deux requêtes.

J'ai pu passer les valeurs en paramètre, au lieu de passer l'objet j'ai passé toutes les valeurs séparément. En avançant je viens de me rendre qu'il faut que j'ajoute un champ dans les 2 tables "COACHING REVIEW" et "SESSION REVIEW". Il faut que j'ai l'id de la session/coach concerné par la review.

J'ai rajouté les champs "target\_id" aux deux tables de reviews. J'ai mis à jour les diagrammes.

J'ai pu terminer l'affichage des 2 types de reviews, je pourrai revoir encore le style éventuellement plus tard si j'ai le temps.

## 2.1.21 Mercredi, 11 Mai 2022

---

Nous allons faire le point intermédiaire aujourd'hui avec M.Jossi, je pense toujours avoir un peu de retard sur le projet de manière générale.

Je rencontre un problème lors de l'ajout des charts sur la page profil. J'ai fait une requête pour obtenir les types d'entrainements qui ont été effectués un client ainsi que le nombre de séance effectué pour chaque type. Je sépare donc ces valeurs en 2 tableaux (Pour pouvoir ensuite les passer dans le Javascript et utiliser les valeurs avec Chart.JS). Seulement une fois passer du côté HTML j'ai un problème, pour tableau qui contient les valeurs numériques (int) tout va bien mais pour le tableau qui contient le titre des exercices (string) les apostrophes contenues dans le tableau pour déterminer les strings sont remplacées par : "

Tableau avant (backend python)

```
['Weightlifting', 'Cycling']
```

Tableau après (frontend html)

```
["Weightlifting", "Cycling"]
```

Le problème est que je ne peux même pas remplacer les caractères indésirables car ce n'est pas une chaîne mais un tableau.

Pour résoudre ce problème, j'ai fait une boucle qui parcourt le tableau reçu sur le côté template et j'ai rempli un tableau javascript avec les valeurs. J'ai ensuite remplacé les caractères indésirables des valeurs et tout fonctionnait.

En me faisant une réflexion sur l'application, je me suis rendu compte que je vais devoir modifier une maquette. Sur la maquette d'interface pour la page détail d'un entraînement, j'ai affiché un graphique qui représentait l'évolution des battements de cœur durant la séance. Malheureusement l'API Polar ne permet pas de récolter les données de toute la séance, on obtient uniquement le bpm max, min et moyen (pas de quoi afficher un graphique représentant la séance entière).

J'ai fait le point intermédiaire avec M.Jossi, dans l'ensemble je suis sur le bon chemin. Il ne faut pas que je me relâche mais j'avance dans la bonne direction. J'ai toujours un peu de retard mais je ne me fais pas trop de soucis pour l'instant. Il faut que je commence à découper le code python (séparer en fonction pour chaque requête effectuée) et il faut également que j'ajoute le guide d'installation dans la documentation. M.Jossi m'a également fait la remarque qu'il faudrait que je précise les entrées obligatoires dans les formulaires présents sur les maquettes.

J'ai pu commencer l'ajout du deuxième graphique de la page profil avec Chart.JS qui représentera le nombre d'entraînements effectués par mois.

## 2.1.22 Jeudi, 12 Mai 2022

---

J'ai pu terminer le passage des valeurs pour le deuxième graphique (pour le nombre d'entraînements par mois). Je n'ai pas pu avancer l'après-midi car il y a eu l'après-midi "poster" où tous les élèves CFC ont venu voir les projets.

## 2.1.23 Vendredi, 13 Mai 2022

---

J'ai reçu un mail inquiétant de la part de Polar. Ils annoncent une maintenance le 18 Juin, je ne sais pas pour combien de temps. Sachant que la défense du diplôme a lieu le 20, je vais devoir prendre des précautions et faire une sorte de "Mock" de l'API Polar et simuler la récupération des données au cas où. Je vais quand même leurs envoyer un mail, pour connaître la durée de la maintenance et savoir si des changements majeurs auront lieu (des changements qui risquent de rendre obsolète mon application).

J'ai remarqué que les données utilisées pour le graphique qui affiche le nombre d'entraînements par mois ne prends pas en compte l'année (si j'ai fait des entraînements en Mai 2021 et Mai 2022 il prendra la somme des 2). Je vais rajouter le critère de l'année actuelle dans la requête SQL.

J'ai pu ajouter les statistiques de la semaine à côté des deux graphiques. Il faut encore que j'affiche le total d'heures d'entraînements de la semaine, la difficulté est qu'il faut que j'utilise la fonction SQL SUM() sur des champs du type TIME (ce qui n'est pas faisable en SQL). Pour solutionner ce léger problème j'ai utilisé la fonction SQL extract qui m'a permis d'extraire les heures, minutes et secondes de chaque durée et ainsi les additionner.

J'ai également pu ajouter la page "Prochaine sessions" qui affichent la liste des prochaines sessions avec un coach. Pour ressembler le plus possible à la maquette j'ai ajouté dans le dossier static/assets/img/sports des logos de beaucoup de sports qui serviront d'illustrations pour le type de séance qui est prévu pour la session.

Je vais pouvoir maintenant commencer à séparer en différentes fonctions les requêtes SQL pour rendre plus lisible le fichier de routes.

Pour le côté client, il ne manque plus que l'affichage détaillé des entraînements et l'option changement de mot de passe. Je pourrai ensuite commencer le côté coach. Pour la documentation je vais commencer à documenter les endpoints du côté client.

## 2.1.24 Lundi, 16 Mai 2022

---

Aujourd'hui, j'ai pu terminer la page détails d'entraînements, je ne me suis pas trop attardé sur le style mais la page est fonctionnelle. J'ai pu ensuite poursuivre avec la page ajout de review, je vais en faire une seule qui s'adaptera en fonction des champs qui lui sont envoyés.

Je remarque qu'il faut que je modifie la maquette d'ajout de review sur le coaching car j'avais modifié un champ et sur la maquette l'ancien champ est toujours présent.

J'ai eu une réflexion lors de l'élaboration de la page d'ajout de review, ne faut-il pas ajouter une review sur les entraînements plutôt que les sessions ?

Car les sessions ne sont pas toujours des entraînements et cela ne sert à rien d'ajouter une review sur un rendez-vous pour un bilan. Il faut donc que je modifie les noms de table, les relations ainsi que les clés étrangères.

J'ai pu terminer la page d'ajout de review à 90%, il ne manque plus qu'à modifier la base de données pour que les revies soient sur les entraînements et plus les sessions et la page sera fonctionnelle pour l'ajout de review sur les entraînements. J'ajouterai les reviews coaching demain.

## 2.1.25 Mardi, 17 Mai 2022

---

Je vais commencer par modifier la base de données pour passer de "SESSION REVIEW" à "WORKOUT REVIEW" car comme réfléchi hier, il est plus intéressant de pouvoir mettre une review sur un entraînement que sur une session car certains workouts ne sont pas une session et le retour sur ces entraînements peuvent intéresser le coach.

J'ai pu tout modifier, il faut maintenant que je fasse en sorte qu'un entraînement ne puisse avoir qu'une seule review pour éviter que l'utilisateur en publie plusieurs.

J'ai également fait la modification de la maquette d'ajout de review sur le coaching les 4 champs n'étant plus un select mais un slider comme les autres champs.

J'ai ajouté la possibilité d'ajouter les review du coaching. Il faudrait que j'ajoute en tout cas pour l'ajout de review sur le coaching un petit carré qui montre le coach en question. Pour l'ajout de review sur un entraînement cela n'est pas forcément nécessaire sachant qu'il faut être sur la page de détail de l'entraînement pour ajouter une review.

Il faut que j'ajoute une redirection automatique lorsque la review ajouté.

Je ne sais pas si c'est un problème mais vu qu'avant de vérifier si la review d'entraînement existe je remplis la table *REVIEW* cela augmente l'id de 1 et si la review existe tout est rollback mais l'auto incrément garde l'incrément de 1.

Lors de la création de la page pour changer le mot de passe, je rencontre un problème. J'avais ajouté dans le modèle *User* une fonction qui permet de hash le password qui est défini sur la propriété "password" mais lorsque je change la propriété le mot de passe n'est pas hashé et si je le hash manuellement le programme plante.

La modification de mot de passe est désormais fonctionnel mais je rencontre maintenant un autre problème le *logout* ne fonctionne plus

Je n'arrive plus à logout l'utilisateur connecté, dans la route du login il y a une condition :

```
if not current_user.is_authenticated
```

sauf que même après avoir utilisé la fonction *logout\_user* de Flask Login, je ne passe jamais dans cette condition.

## 2.1.26 Mercredi, 18 Mai 2022

---

Je suis resté bloqué sur mon problème de logout presque toute la matinée, je ne trouvais absolument aucune solution logique. M. Zanardi est venu voir mon problème et nous avons debugué le logout. Nous

avons découvert qu'une fois logout Flask Login se connectait automatiquement à un utilisateur "test" que j'avais créé dans la base de données sans mot de passe et sans email.

Ces champs devront être non null car ils sont obligatoires à l'inscription.

Je ne sais pas ni pourquoi ni comment mais Flask Login se connectait par défaut à l'utilisateur sans email. Je pouvais donc après m'être déconnecté accéder aux pages sous `@login_required`. En supprimant cet utilisateur `test`, j'ai remarqué que tout fonctionnait comme il faut car Flask login ne pouvait plus se connecter automatiquement à cet utilisateur sans email.

La partie client est donc terminé, il va falloir que j'effectue quelques tests mais dans l'ensemble tout à l'air fonctionnel. Je vais débuter la partie coach en reprenant déjà les pages clients qui ont des options `coach` et j'ajouterai les nouvelles pages après.

J'ai commencé à séparer un peu le fichier `client/routes.py` j'ai créer un autre fichier `util.py` qui contiendra toutes les fonctions qui permettront de récupérer des données de la base de données. Cela rendra mon fichier `routes.py` beaucoup plus lisible.

## 2.1.27 Jeudi, 19 Mai 2022

---

Aujourd'hui j'ai bien avancé sur l'épuration de mon code, il y avait plein de requête que je faisais en double ou qui était inutile. Exemple pour la plupart des pages je récupérais un objet `user` pour avoir toutes les propriétés alors qu'avec Flask login je détenais déjà un objet `User (current_user)`. Il ne reste qu'une petite partie du fichier `routes.py` à corriger et ça sera bon.

Demain, je vais essayer de beaucoup avancer sur la documentation technique car je ne l'ai plus touchée depuis un moment. J'ai pas mal de points à ajouter comme notamment les endpoints de l'application ou encore les différentes fonctions qui sont appelées pour récupérer des données de la base.

Je pourrai ensuite ajouter la partie coach de l'application qui maintenant que le code est propre, ne devrait normalement pas me poser de gros problèmes.

En croisant M.Jossi, je lui ai demandé si cela pouvait être pertinent d'ajouter une sorte de business plan à la documentation. Pour faire en sorte d'avoir une brève explication des coûts qu'il faudrait prendre en compte si on voulait mettre en place l'application chez un coach. (En comptant les montres, les cartes, les lecteurs, etc..). M.Jossi m'a confirmé que cela pouvait être une bonne idée mais cela n'est vraiment pas prioritaire. Je vais donc garder cette idée et si il me reste du temps sur la fin j'ajouterai ce point à la documentation.

## 2.1.28 Vendredi, 20 Mai 2022

---

Aujourd'hui j'ai pu terminer de refaire le fichier de route, toutes les fonctions utilisant la base de données sont maintenant dans le fichier *util.py*. J'ai pu avancer sur la documentation en ajoutant le routage de l'application et en commençant à expliquer les différents endpoints. Je pense travailler un peu ce week-end pour avancer un peu la documentation et je pourrai également commencer à ajouter les pages pour coach.

Pour la lecture des cartes RFID et la récupération des données via l'API, je possède déjà des scripts me permettant de le faire. Je n'aurai qu'à les adapter pour les faire fonctionner.

Avec l'annonce de maintenance de l'API le 18 Juin, je vais quand même créer un mock de l'API pour me permettre de faire quand même tourner l'application au cas où l'API ne fonctionnerait pas pour la présentation.

## 2.1.29 Lundi, 23 Mai 2022

---

J'ai découvert et voulus implémenter le *UserManager* qui m'aurait permis de gérer l'accès à mes pages avec le décorateur : *@roles\_required*. Malheureusement, il est nécessaire d'utiliser une certaine structure de données et cela impliquerait trop de changement au stade actuel. Je vais donc m'en passer et faire des vérifications plus traditionnelles pour gérer les accès aux pages.

J'ai eu un petit problème avec le formatage des datetimes récupérer depuis la base. J'ai simplement utilisé *strftime()* qui me permet de les afficher avec le format que je veux.

J'ai pu terminer le tableau de bord coach, la prochaine session est affichée avec la date, durée et type de session. Le client concerné est également affiché. En dessous, on peut retrouver une liste de tous les clients que le coach suit avec quelques infos comme la date de fin de leurs abonnements ou encore le dernier entraînement qu'ils ont effectué.

Le dernier rendu intermédiaire a lieu demain, je vais essayer d'avancer un petit peu la documentation encore. En comptant cette semaine il reste un peu moins que 3 semaines avant la fin de ce travail. Pour être dans les temps, il faudrait que je termine le côté coach cette semaine, j'aurai ensuite une semaine pour implémenter l'API et les cartes RFID et je pourrai pour la dernière semaine effectuer les derniers tests et arranger la documentation.

## 2.1.30 Mardi, 24 Mai 2022

---

Nous avons le dernier rendu intermédiaire aujourd'hui. J'ai commencé à travailler sur la page calendrier qui est une des dernières pages "compliquées" à réaliser. J'ai en premier temps, utilisé un vieux calendrier que j'avais effectué lors de ma formation ici. En faisant quelques recherches j'ai fait la découverte de [FullCalendar.io](#) qui permet d'utiliser un calendrier javascript avec beaucoup de fonctionnalités. J'ai décidé

d'essayer de l'implémenter dans mon application car cela va me permettre d'avoir différents affichages du calendrier (Semaines, mois, jour, liste). J'ai réussi à implémenter le calendrier FullCalendar.

Pour les événements, je dois donner un tableau contenant plusieurs objets représentant un événement. Chaque événement possède au minimum 1 titre, 1 date de début.

Exemple :

```
{
  title: 'Thomas Fuji',
  start: '2022-05-22T16:00:00',
  end: '2022-05-22T17:00:00'
}
```

Je vais donc faire une fonction qui va me retourner toutes les sessions enregistrées sous ce format et je le passerai ensuite en javascript pour renseigner les événements à afficher sur le calendrier.

J'ai également avancé sur le formulaire d'ajout de session, j'ai eu plusieurs problèmes d'affichage à cause de quelques erreurs d'inattention. Pour le formulaire d'ajout, il y a un champ *Date* et un champ *Time* pour sélectionner la date et l'heure de la session. On peut ensuite sélectionner une valeur de 1 à 3 (représentant le nombre d'heure prévu pour la séance). L'heure de fin est générée automatiquement lorsque les champs mentionnés ci-dessus sont renseignés.

Avec le rendu d'aujourd'hui, j'ai toujours du retard, néanmoins je pense quand même réussir à terminer dans les temps, j'avance beaucoup sur les dernières pages à faire et une fois cela terminer, j'ai énormément de point à ajouter à la documentation.

J'ai ajouté les tâches qu'il reste à effectuer dans le Trello.

## 2.1.31 Mercredi, 25 Mai 2022

---

J'ai refait le modèle de la table Session, à la place d'avoir qu'une date, j'ai un *start\_time* et un *end\_time*. Cela me facilitera la conversion en événement pour le calendrier.

J'ai pu terminer le formulaire d'ajout de session. On peut maintenant ajouter une session en sélectionnant :

- Client
- Type d'entraînement
- Date
- Heure
- Durée

Pour l'affichage j'ai récupéré toutes les sessions et je les ai insérés dans un tableau pour que ça respecte le format de FullCalendar. Je passe ensuite mon tableau directement au javascript pour qu'il l'utilise comme source.

J'ai eu le dernier point avec M.Jossi, nous avons regarder les tâches qu'il me reste à effectuer. Dans l'ensemble, j'ai encore beaucoup de travail mais cela devrait être faisable dans les temps.

L'objectif est de terminer toutes les pages (Ajout client, paiement, bilan, ajout de programme et affichage de profil client) avant la fin de cette semaine. Je pense que c'est faisable car la plus part des pages ne sont qu'un formulaire pour insérer des éléments dans la base de données.

## 2.1.32 Vendredi, 26 Mai 2022

---

Aujourd'hui j'ai poursuivi l'avancement des dernières pages qu'il me reste à créer. J'ai pu terminer la page d'ajout de nouveau client, le coach peut désormais créer un client lui-même.

J'ai également fait la page de profil client, on y retrouve plusieurs infos du client comme les reviews qu'il a posté, ses programmes ainsi que des graphiques représentant les entraînements qu'il effectue.

Je vais cette fois vraiment devoir travailler un peu ce week-end, pour m'assurer que toutes les pages soient terminées et fonctionnent comme je le souhaite. Je pourrai ainsi débuter la récupération des données avec l'API et la détection de carte RFID. J'espère au plus tard terminer mon projet en fin de semaine prochaine, pour garder une semaine complète de documentation.

## 2.1.33 Lundi, 30 Mai 2022

---

J'ai pu terminer toutes les pages que je devais encore créer, le coach peut maintenant ajouter un "checkup"(bilan) à un utilisateur et peut renouveler l'abonnement d'un client. J'ai dû modifier légèrement la base de données, j'ai ajouté un champ *date* à la table *PROGRAM* pour garder les anciens programmes. J'ai également modifié le champ *pdf* en *LONGBLOB* car *BLOB* était trop court dans certains cas.

Le client peut maintenant télécharger son programme directement depuis son profil dans la section *Programs*.

J'ai encore quelques détails à revoir et je vais tester toutes les fonctionnalités/pages. Si je ne rencontre aucun problème et que tout est ok, je vais ajouter la lecture de carte RFID pour assigner les cartes de membres.

J'ai pu régler quelques détails comme :

- Affichage d'un message d'erreur *Password must match* lors de la création d'un nouveau compte
- Définir le rôle coach lors de l'enregistrement de la page register (Les clients sont créés avec le formulaire coach)
- Affichage des messages *flash* avec le bon CSS
- Détails d'erreur pour l'ajout au calendrier si le coach ne possède aucun client.
- Affichage d'un message à la place des graphiques si l'utilisateur n'a pas encore de données d'entraînement
- Ajouter le total de temps d'entraînement cette semaine

J'ai pu ajouter la mise à jour de la carte de membre, lorsque le coach click sur le bouton *change* sur le profil du client il peut changer la carte de membre en scannant une nouvelle carte.

Pour l'instant, je n'ai pas mis de timeout, c'est-à-dire que si le coach ne scanne pas de carte alors qu'il a cliqué sur le bouton le site restera bloqué à attendre la carte. Je pense donc mettre un timeout de 30s

## 2.1.34 Mardi, 31 Mai 2022

---

J'ai mis à jour quelques détails comme l'affichage d'icônes cohérentes à côté des statistiques.

J'ai commencé à implémenter le mailer avec *FlaskMail*. J'ai ajouté la configuration dans mon fichier de config.

Pour envoyer les mails, j'ai créer le mail : *fitjourney.cfpt@gmail.com*.

Après quelques tests, le mailer fonctionne correctement, je vais maintenant ajouter l'envoie d'un mail de notification sur quelques actions. (Pas trop car cela prend du temps donc je ne vais pas trop charger l'application avec des envois de mail)

J'ai ajouté l'envoie d'une notification par mail lorsque le coach ajoute un nouveau programme, lors du renouvellement d'un abonnement , lors de l'enregistrement d'une session et lors de la création du compte client également.

REMARQUE : Les actions qui envoient un mail, sont remarquablement ralenties.

Il me reste quelques point à effectuer avant de commencer le script qui tournera à côté pour gérer les entrées/sorties et l'appel à l'API :

- Ajouter un graphique représentant l'évolution du poid (si possible changement des valeurs avec *Masse graisseuse, Masse musculaire, etc*)
- (Ajouter la possibilité au coach de modifier les informations du compte client)
- Vérifier que le compte connecté est bien coach pour toutes les pages coach
- Ajouter l'annulation d'un abonnement

## 2.1.35 Mercredi, 01 Juin 2022

---

J'ai envoyé un mail à M.Jossi pour faire une mise au point avant de commencer la dernière semaine.

J'ai pu ajouter l'annulation d'abonnement, lorsque le coach clique sur le bouton *Cancel*.

J'ai également ajouté un graphique qui affiche l'évolution du poids d'un client. Je récupère la moyenne des pesées enregistrée pour chaque mois (dans la plupart des cas 1 bilan par mois). Je n'aurai pas le temps de le faire maintenant, mais pour une amélioration future cela serait intéressant d'ajouter une "navbar" qui permettent de changer les valeurs du graphique entre toutes les données récupérées lors d'un bilan (Taux de masse musculaire, graisseuse, etc..)

J'ai également ajouté le check pour les pages coach, elles ne sont plus accessibles par un client.

Nous avons fait une mise au point avec M.Jossi, dans l'ensemble je commence à arriver au bout. Je dois encore rajouter un historique des bilans ajoutés pour avoir quand même un petit aperçu des bilans ajoutés. M.Jossi m'a également fait remarquer que je n'avais pas mis l'échelle à côté des valeurs des reviews. Je vais donc les rajouter.

J'ai pu terminer d'ajouter l'historique de bilan, sur la page client pour le coach et sur la page profil pour le client.

## 2.1.36 Jeudi, 02 Juin 2022

---

Pour aujourd'hui, il me reste à valider l'insert dans la DB des données d'entraînement récupérés avec l'API Polar. Je vais également déjà préparé une méthode que j'utilisera pour la démo si l'API n'est pas disponible.

Je pourrai ensuite établir mon scénario de démo et commencer à effectuer tous les tests utilisateurs que j'ajouterais directement dans la documentation.

J'ai pu terminer l'insert dans la base de données, je ne peux pas encore tester car l'API Polar semble avoir un problème (Impossible de faire une requête), j'obtiens une *Error 500 Internal server error* à chaque requête.

Je testerai demain pour voir si mon script fonctionne comme il faut, je vais préparer ma méthode pour récupérer des données dans le même format que les données Polar.

En relisant les termes et conditions d'utilisations de l'API, j'ai vu que tout en bas de leur pdf ils précisent qu'ils ne garantissent **ABSOLUMENT** pas la disponibilité de leurs API. Je vais donc bien utiliser ma méthode car je veux vraiment éviter les problèmes lors de la démo.

## 2.1.37 Vendredi, 03 Juin 2022

---

L'API Polar semble être à nouveau disponible, je vais tester mon script pour vérifier que l'insert dans la base de données est effectué correctement. Une fois cette étape validée, je commencerai à effectuer tous les tests utilisateurs pour vérifier qu'ils ne restent aucun "bug". L'objectif est qu'à la fin de la journée je ne touche plus au code et qu'il me reste uniquement la documentation à terminer.

J'ai dû faire quelques ajustements pour certains formats de valeurs lors de l'insert. J'utilise le champ *duration* que l'API me donne sauf qu'il est rendu avec un format très spécifique qui change en fonction de la durée exemple :

Pour une séance de 1h30 : *PT1H30M*

Pour une séance de 50 min et 30s : *PT50M30S*

j'ai donc fait un try avec plusieurs exceptions pour permettre de couvrir les différents cas.

L'insert est désormais fonctionnel, les données d'entraînement sont correctement insérées dans la base de données. Je vais donc commencer mon tableau de test utilisateur et vérifier que toutes mes fonctionnalités fonctionnent comme il faut.

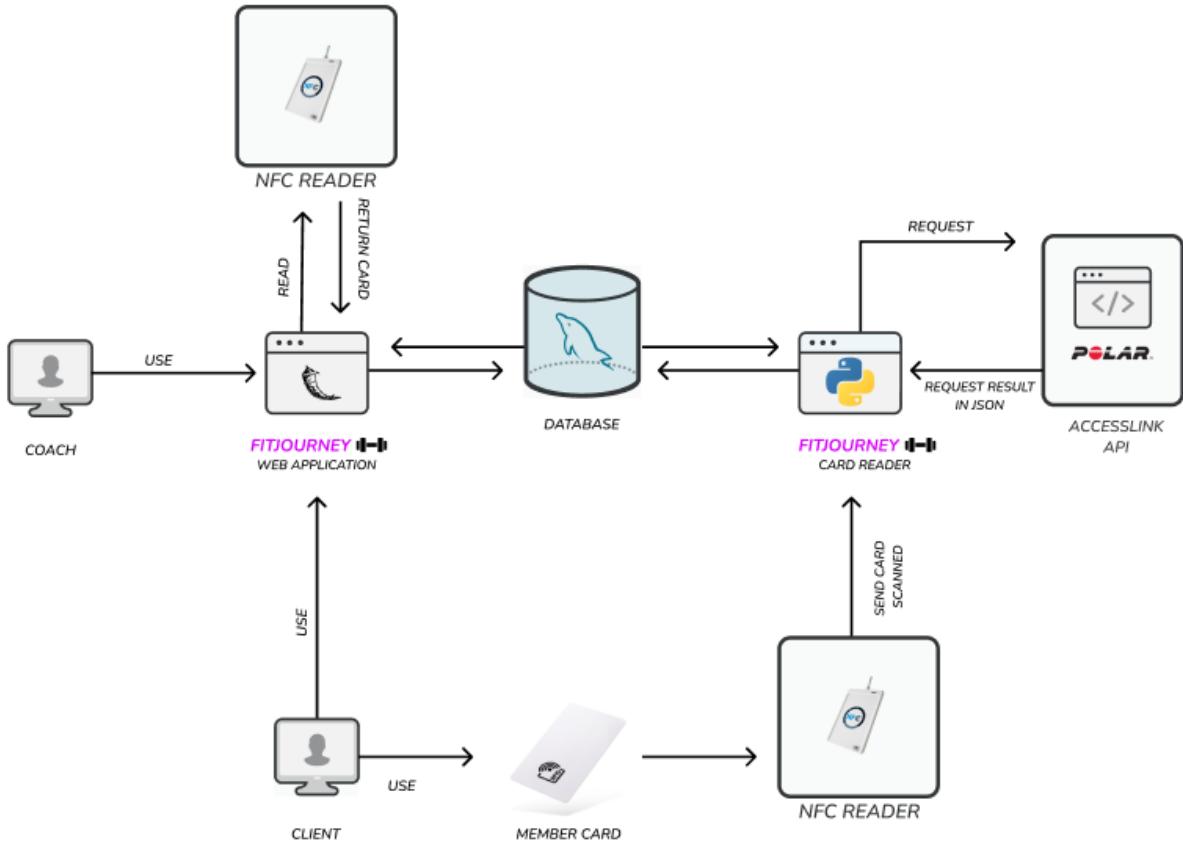
J'ai pu réaliser une bonne partie des tests tout est fonctionnel pour l'instant, je vais terminer les tests et avancer grandement dans la documentation. J'aimerais avoir une version "très avancée" à montrer à M.Jossi d'ici Lundi soir pour avoir le temps d'avoir un retour avant la fin.

Pour la documentation, je vais devoir refaire les diagrammes de USECASES ainsi que l'architecture du projet qui ont bien changé depuis la dernière fois.

## 2.1.38 Mardi, 07 Juin 2022

---

Dernière semaine de ce travail de diplôme, je dois encore pas mal avancer sur la documentation. J'ai ajouté un schéma pour expliquer le fonctionnement interne de l'application :



En ajoutant les plannings, je me suis bien rendu compte que le planning prévisionnel n'a vraiment pas été respecté. Beaucoup de tâches sont venues s'ajouter et certaines ont pris plus de temps que prévu. J'ai pu décrire toutes les technologies que j'ai utilisé et terminé une grosse partie de l'analyse organique. Il faut encore que j'ajoute les tests utilisateurs que j'ai effectués, que je souligne les améliorations possibles ainsi que mon bilan personnel.

## 2.1.39 Mercredi, 08 Juin 2022

---

Aujourd'hui, je vais continuer d'avancer sur la documentation l'idéal serait de la terminer d'ici demain matin. Cela me laissera le temps de faire un manuel utilisateur pour le client et le coach.

En voulant montrer application, je me rends compte que le mailer ne fonctionne plus. Le compte google fitjourney-cfpt@gmail.com a été désactivé pour "Accès suspect" alors que j'avais au préalable bien autorisé l'accès à des applications externes. J'ai réactivé le compte et vais attendre un peu voir si cela refonctionne. En attendant je vais continuer la documentation en ajoutant les tests utilisateurs qui ont été effectué.

En retestant le mailer je me rends compte qu'il ne fonctionne toujours pas. Je vais donc voir sur le compte google voir s'il y a un problème avec le compte et je vois que depuis le 30 mai 2022 Google ne prend plus en charge les applications "moins sécurisée"

Pour vous aider à sécuriser votre compte, à compter du **30 mai 2022**, Google ne prendra plus en charge l'utilisation d'applications ou d'appareils tiers qui vous demandent uniquement votre nom d'utilisateur et votre mot de passe pour vous connecter à votre compte Google.

Ce qui veut dire que je ne peux plus utiliser de compte google pour envoyer les mails. Je vais devoir trouver une solution sinon le mailer ne sera pas fonctionnel. J'ai essayé d'utiliser un compte Yahoo à la place, mais cela n'a pas l'air de fonctionner. Le problème est que le 90% de la documentation que l'on peut trouver sur le net utilise les comptes google. Je n'ai toujours pas trouvé de solution, alors je pense utiliser [MailTrap](#). MailTrap est un service "sandbox" pour email, il capture le trafic SMTP et permet d'analyser les mails. Le mailer n'étant plus fonctionnel, je peux utiliser mailtrap pour démontrer que le mail s'envoie correctement (le mail ne s'enverra pas réellement au destinataire, il restera dans la boîte mailtrap)

## 2.1.40 Jeudi, 09 Juin 2022

---

Avant-dernier jour avant le rendu, je vais refaire un tour sur mon application vérifier que tout fonctionne normalement et que la réception avec MailTrap est fonctionnelle.

Je vais garder MailTrap car je n'ai plus le temps d'implémenter l'API Google qui me permettrait d'utiliser les comptes google pour le mailer comme je le faisais initialement. J'ai changé les morceaux de code que j'avais placé dans ma documentation, je les ai générés avec [Ray.so](#) pour avoir un rendu visuel meilleur.

Exemple de code avec Ray.so :

```
try:
    newWorkoutReview = WorkoutReview(id= newReview.id,difficulty=request.form['field1'],
                                      feel=request.form['field2'], fatigue=request.form['field3'],
                                      energy=request.form['field4'], target_id=request.form['target'])
    db.session.add(newWorkoutReview)
    db.session.commit()
    flash("Your workout review is added", 'success')
except :
    db.session.rollback()
    flash("Error, please try again", 'danger')
```

Il faut que j'ajoute encore le retour sur l'expérience qui comprendra : \* Points positifs \* Points négatifs \* Améliorations \* Problèmes rencontrés \* Bilan personnel

Je peux ensuite commencer les manuels utilisateurs pour détailler l'utilisation de mon application.

J'ai eu le temps de terminer les deux manuels utilisateur. Il faut encore que je corrige l'orthographe de ma documentation et que je génère le listing du code source et je serai bon.