

**Alexandre ALOESODE**

# **DOSSIER DE PROJET**

**Concepteur Développeur d'Applications  
2024/2025**

 **La Plateforme**

# TABLE DES MATIÈRES

<b>INTRODUCTION</b>	<b>4</b>
<b>I. Expression des besoins</b>	<b>6</b>
A. Problèmes rencontrés par les étudiants	6
1. Mobilité	6
2. Ergonomie web	6
3. Fragmentation des outils	6
B. Contexte du projet	6
C. Objectifs	7
D. Contraintes rencontrées	7
1. Contraintes techniques	7
a) Technologies orientées mobile	7
b) API et serveur d'Auth La Plateforme	8
c) Authentification Google et La Plateforme	8
d) Utilisation des tokens	8
2. Contraintes fonctionnelles	8
a) Aucune interface admin	8
b) Protection des données étudiants	8
c) Restriction de l'accès aux données	8
d) RGPD	9
3. Contraintes d'UX/UI	9
a) Responsive sur Android et IOS	9
b) Navigation simple	9
c) Respect de charte graphique	9
<b>II. Environnement technique</b>	<b>9</b>
A. Architecture générale du projet	10
B. Technologies et langages utilisés	10
1. Développement front-end	10
a) React Native + Expo	10
b) Typescript	10
2. Développement back-end	10
a) CodeIgniter	11
3. Bases de données	11
C. DevOps	11
1. Versionning (Git & GitHub)	11
a) Gestion des branches	11
b) Pull requests	12
c) Revues de code	12
2. Intégration continue (GitHub Actions)	12
a) Linting automatique (ESLint)	13
b) Tests unitaires (JEST)	13
c) Notifications mail pour erreurs de build et de test	13
D. Outils annexes	14
1. Maquettage	14

a) Figma	14
2. Environnement de développement	14
a) Docker	14
b) Postman	14
c) DBeaver	15
3. Outils de collaboration	15
a) Trello	15
b) Google Chat	15
<b>III. Réalisation du projet</b>	<b>15</b>
A. Conception et Maquettage de l'application	15
1. Userflow - Parcours utilisateur	15
2. Charte graphique	16
3. Réalisation de la maquette	16
a) Modèle Wireframe	17
b) Évolution vers la maquette finale	18
B. Base de données	19
1. Modèle conceptuel de données	20
2. Modèle physique de données	21
C. Normes de développement	21
1. Architecture MVC vs MVP	22
D. Conteneurisation et environnement	23
1. Dockerisation	23
a. Création des dockerfiles	23
b. Création du docker-compose	26
2. Variables d'environnement	27
a. Fichiers de constants pour l'auth et l'api	27
b. Fichier .env pour le front	29
c. Gestion des secrets en développement	29
E. Intégration back-end	30
1. Intégration de l'Authentification La Plateforme	30
2. Intégration de l'API La Plateforme	33
F. Développement front-end	40
1. Architecture du front-end	41
2. Authentification Google OAuth	42
3. Systèmes de requêtes	45
a) Fetch	45
b) Axios	45
c) Fichier utilitaire de requêtes API	48
4. Typage	50
G. Tests et qualité logicielle	52
1. Linting	52
2. Tests des composants	54
3. Automatisation des tests	56
<b>CONCLUSION</b>	<b>58</b>
<b>ANNEXES</b>	<b>60</b>

## INTRODUCTION

Âgé de 36 ans et initialement diplômé d'une école de commerce en 2014, j'ai débuté un projet de reconversion professionnelle en 2022, avec pour objectif de devenir Développeur Web. J'ai donc rejoint l'école La Plateforme en septembre 2022 et démarré un Bachelor, afin d'obtenir le diplôme de Concepteur Développeur d'Applications. Cette formation s'effectuant en alternance, j'ai en parallèle rejoint l'Atelier en tant qu'alternant en mars 2023. L'Atelier est une structure rattachée à La Plateforme, qui réalise des projets informatiques d'envergures diverses pour différentes entreprises. L'un des plus gros clients de l'Atelier n'est autre que l'école La Plateforme; depuis deux années, je m'occupe à temps plein de maintenir les outils de l'école et de développer de nouvelles fonctionnalités.

Cette double casquette d'étudiant et employé indirect de La Plateforme m'a permis d'identifier précisément les besoins des utilisateurs finaux, les limites actuelles de l'infrastructure, et les leviers d'amélioration réalistes.

La Plateforme met à disposition de ses étudiants et employés deux interfaces web principales : un intranet étudiant (consultation des absences, des projets, du profil...) et un intranet administration (création et gestion des promotions, création et gestion des projets, etc...). Ces outils, bien que fonctionnels, souffrent d'une accessibilité limitée depuis un smartphone et d'une ergonomie peu optimisée pour la navigation mobile.

Ces limites d'ergonomie ne sont pas un problème pour les employés de l'école, qui utilisent l'intranet administration sur leurs postes de travail. C'est en revanche le cas pour les étudiants, qui ont une utilisation bien différente de leur intranet. C'est dans ce contexte que m'est venue l'idée du projet SchoolTool.

SchoolTool est une application mobile développée pour répondre à un besoin concret identifié au sein de l'école La Plateforme.

L'objectif est de proposer une interface mobile moderne, accessible et intuitive permettant aux étudiants d'accéder à leurs outils essentiels de suivi de scolarité, actuellement disponibles uniquement sur des interfaces web peu adaptées à un usage quotidien sur smartphone.

L'application SchoolTool vise à :

- Centraliser les fonctionnalités les plus utilisées par les étudiants dans une seule interface mobile.
- Améliorer l'expérience utilisateur via une navigation fluide et intuitive.
- S'appuyer sur les systèmes d'authentification et d'API existants de La Plateforme sans les modifier.
- Offrir une base solide pour de futures évolutions (notification, mise à disposition sur stores, personnalisation).

SchoolTool présente selon moi trois avantages majeurs.

Le premier est d'apporter du confort aux étudiants, par l'intermédiaire d'un outil moderne et conforme aux usages actuels. Le second est d'apporter à l'école un outil supplémentaire, afin d'améliorer sa position sur un marché de plus en plus concurrentiel. Enfin, La Plateforme développe des partenariats avec d'autres centres de formation, dont certains amenés à utiliser les outils développés par l'école. Avoir une application mobile pour leurs étudiants présente également un avantage indéniable.

Dans ce dossier, nous commencerons par définir précisément le cadre dans lequel SchoolTool a été pensé, quels sont ses objectifs, et quelles sont les contraintes à respecter. Suite à cela, nous détaillerons précisément les technologies utilisées, et nous finirons par expliquer les différentes étapes du développement de l'application SchoolTool.

# I. Expression des besoins

## A. Problèmes rencontrés par les étudiants

### 1. Mobilité

Les interfaces actuelles des outils de La Plateforme ne sont pas adaptées à une utilisation sur smartphone. Les étudiants doivent souvent naviguer sur des pages non responsive ou zoomer pour cliquer sur certains boutons.

### 2. Ergonomie web

Les outils ne sont pas conçus pour une interaction tactile fluide. Ce sont en effet des interfaces web conçues initialement pour être consultées via un ordinateur.

### 3. Fragmentation des outils

Plusieurs interfaces sont nécessaires pour accéder aux différentes fonctionnalités. Les étudiants disposent d'un intranet pour la gestion de leur quotidien à l'école, et d'un site différent pour la génération de documents (attestation de scolarité, bulletin de notes, etc...).

## B. Contexte du projet

Le projet a été réalisé dans le cadre du projet de fin d'année de Bachelor 3 à La Plateforme. L'équipe est constituée de :

- **Alexandre ALOESODE**
- **Hervé BEZIAT**

Nous avons notamment comme contrainte de nous appuyer sur l'existant en utilisant l'API et le système d'authentification Google

de La Plateforme, afin de garantir la compatibilité avec l'écosystème de l'école.

## C.Objectifs

Suite à l'analyse des besoins et la prise en compte du contexte, nous nous sommes donné les objectifs suivants:

- Adapter les fonctionnalités de l'intranet web au format mobile
- Rendre les données accessibles en mobilité
- Proposer une UI fluide, moderne et épuré
- Intégrer l'écosystème de La Plateforme sans modifier le backend
- Faciliter l'intégration, le développement et les déploiements futurs
- Intégrer les bonnes pratiques de code, de versionning et de tests

## D.Contraintes rencontrées

### 1. Contraintes techniques

#### a) Technologies orientées mobile

Le projet repose sur Expo Go, un framework basé sur React Native qui simplifie le développement multi-plateformes (iOS, Android, Web). La prise en main nécessite la compréhension de ses limites (notamment sur les builds natifs) et des mécanismes d'authentification sécurisée.

### b) API et serveur d'Auth La Plateforme

L'API principale et le service d'authentification sont codés en Codelgniter. Ces deux systèmes sont des outils fournis par La Plateforme, qu'il a fallu interroger sans possibilité de modifier les routes ou la logique métier.

### c) Authentification Google et La Plateforme

Utilisation d'OAuth2 via **expo-auth-session** avec gestion du code de vérification (PKCE), échange de token et récupération d'un JWT généré par le backend de La Plateforme.

### d) Utilisation des tokens

L'application utilise plusieurs types de jetons (access token Google, id token, et deux JWT La Plateforme). Leur stockage est géré via **expo-secure-store**, et leur validité est contrôlée avant chaque appel protégé.

## 2. Contraintes fonctionnelles

### a) Aucune interface admin

L'application est **réservée aux étudiants**, ce qui implique une gestion des rôles simplifiée côté front.

### b) Protection des données étudiants

Pas d'accès ou de vue dédiée aux rôles non-étudiants (admin, enseignant, etc.)



### c) Restriction de l'accès aux données

Chaque étudiant ne devra pouvoir accéder qu'à ses propres données, et ne pourra par aucun moyen récupérer celles d'autres étudiants de l'école.

### d) RGPD

Les informations affichées (absences, projets, profil) ne doivent pas exposer de données sensibles à des tiers non authentifiés.

## 3. Contraintes d'UX/UI

### a) Responsive sur Android et IOS

L'application doit être agréable à utiliser sur téléphones de différentes tailles.

### b) Navigation simple

L'application doit être conforme aux normes et tendances actuelles de navigation sur des applications mobiles, afin de faciliter son utilisation.

### c) Respect de charte graphique

Ce projet nous imposera de respecter une certaine cohérence dans les couleurs, typos et styles pour ne pas heurter l'identité graphique de La Plateforme.

## II. Environnement technique

### A. Architecture générale du projet

L'application repose sur une architecture en microservices conteneurisés via Docker. Elle est composée de :

- Une application mobile développée avec React Native (Expo Go)
- Un service d'authentification externe basé sur le protocole OAuth Google et CodeIgniter
- Une API principale développée avec CodeIgniter
- Une base de données MariaDB
- Des fichiers de configuration .env, docker-compose.yaml, et Dockerfile assurant la cohérence des environnements

### B. Technologies et langages utilisés

Chaque outil a été choisi pour sa compatibilité avec les technologies du projet et son adéquation avec les pratiques professionnelles modernes.

#### 1. Développement front-end

##### a) React Native + Expo

Choix orienté mobilité, permettant de cibler iOS, Android et Web avec un seul code source. Expo permet un développement rapide, des builds simplifiés et une intégration fluide des plugins natifs.

##### b) Typescript

Utilisé pour améliorer la qualité du code et prévenir les erreurs en développement grâce au typage strict.

## 2. Développement back-end

### a) CodeIgniter

Utilisé car imposé par l'infrastructure existante de La Plateforme. Ce framework PHP très léger offre de nombreux avantages en termes de vitesse d'exécution, de sécurité et d'architecture MVC/MVP.

## 3. Bases de données

Base de données SQL robuste, utilisée ici pour stocker les données exposées par l'API et l'auth.

## C.DevOps

### 1. Versionning (Git & GitHub)

Utilisés pour le versionnement, les revues de code, la gestion des branches et l'intégration continue.

### a) Gestion des branches

Nous avons opté pour un fonctionnement avec 3 principaux types de branches:

- La branche *main*, sur laquelle seront toutes les fonctionnalités codées, testées, et prêtes à l'emploi
- Les branches *features*, sur lesquelles chacune des fonctionnalités principales de l'application seront développées dans leur première version
- Les branches *fix/correctif*, sur lesquelles nous viendrons reprendre l'existant dans le but de corriger ou améliorer le code

## b) Pull requests

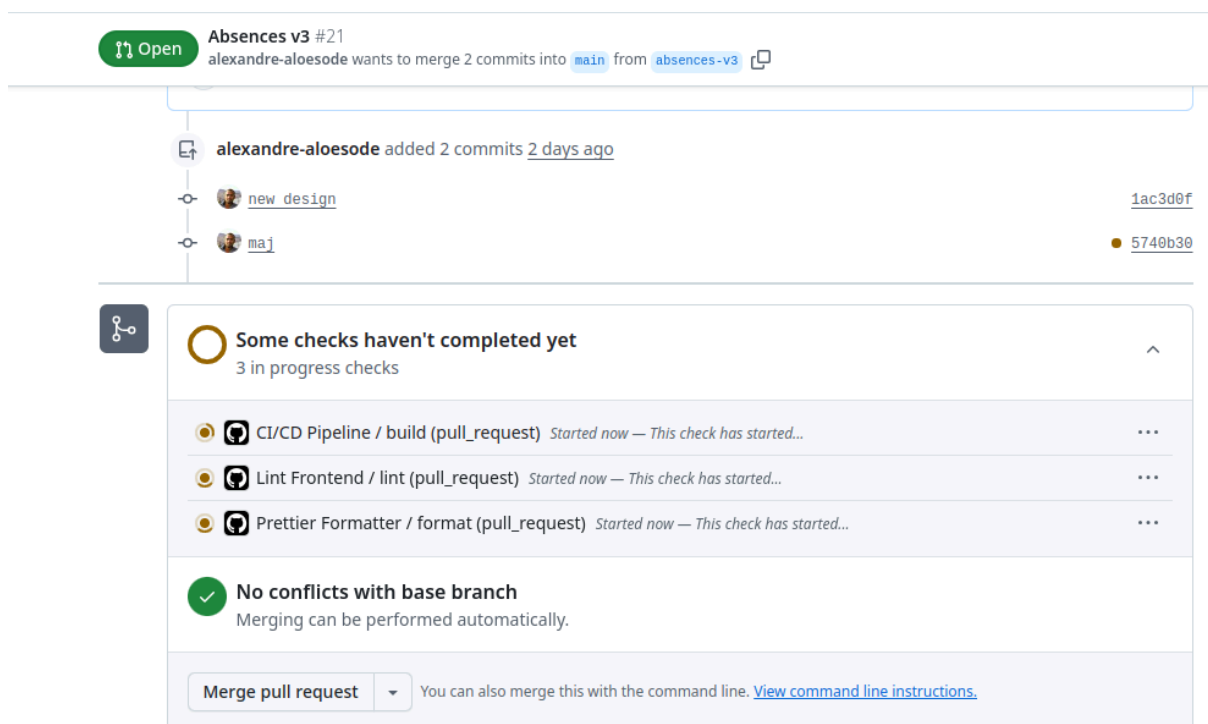
Une fois le travail effectué et testé sur une branche, nous avons utilisé le système de pull request vers la branche main. Ceci nous a permis d'intégrer les fonctionnalités et les améliorations de manière continue, tout en conservant une application fonctionnelle et en évitant les conflits entre branches.

## c) Revue de code

Chaque membre de l'équipe ayant finalisé un ticket réalisait donc une pull request. C'était au reste des deux autres membres de contrôler la pull request et donner leur validation ou non.

## 2. Intégration continue (GitHub Actions)

Nous avons automatisé les tests (jest) et le linting (eslint) à chaque push ou pull request, afin de garantir la qualité du code et la conformité du code.



### a) Linting automatique (ESLint)

Nous avons utilisé le linting automatique avec ESLint, un processus qui analyse le code pour détecter et corriger automatiquement les erreurs de style ou de syntaxe selon des règles prédéfinies. Son but est de :

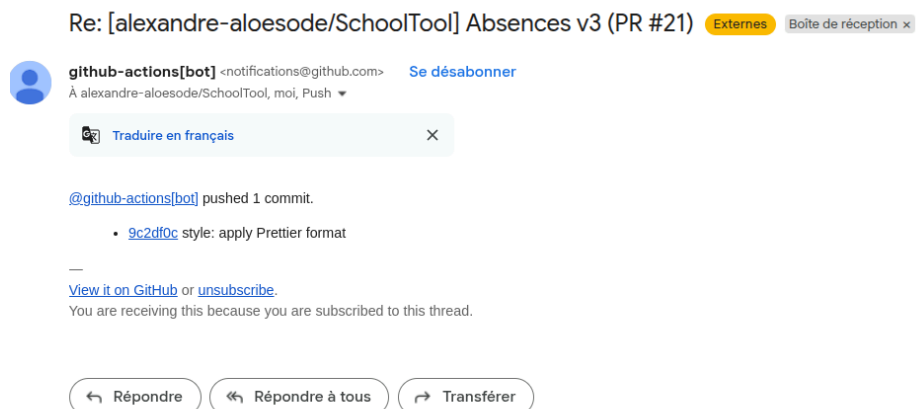
- Améliorer la qualité du code (lisibilité, cohérence).
- Éviter les bugs en signalant des erreurs potentielles.
- Standardiser le style de développement au sein de l'équipe.
- Automatiser la correction de certains problèmes via `eslint --fix`.

### b) Tests unitaires (JEST)

Jest est un framework de test JavaScript utilisé pour tester des applications React et React Native.

### c) Notifications mail pour erreurs de build et de test

Notre projet GitHub a été configuré de telle sorte qu'à chaque pull request, nous soyons notifiés par mail du résultat des différents tests automatisés.



## D. Outils annexes

### 1. Maquettage

#### a) Figma

Pour le maquettage et le prototypage de l'interface mobile.

### 2. Environnement de développement

#### a) Docker

Utilisé pour encapsuler les environnements backend et frontend. Permet une mise en place rapide et reproductible sur n'importe quelle machine.

#### b) Postman

Pour tester manuellement les endpoints de l'API La Plateforme.

#### c) DBeaver

Pour explorer et comprendre la structure de la base de données MariaDB.

### 3. Outils de collaboration

#### a) Trello

Pour le suivi des tâches et la répartition du travail en équipe.

#### b) Google Chat

Pour la communication interne avec les collaborateurs du projet.

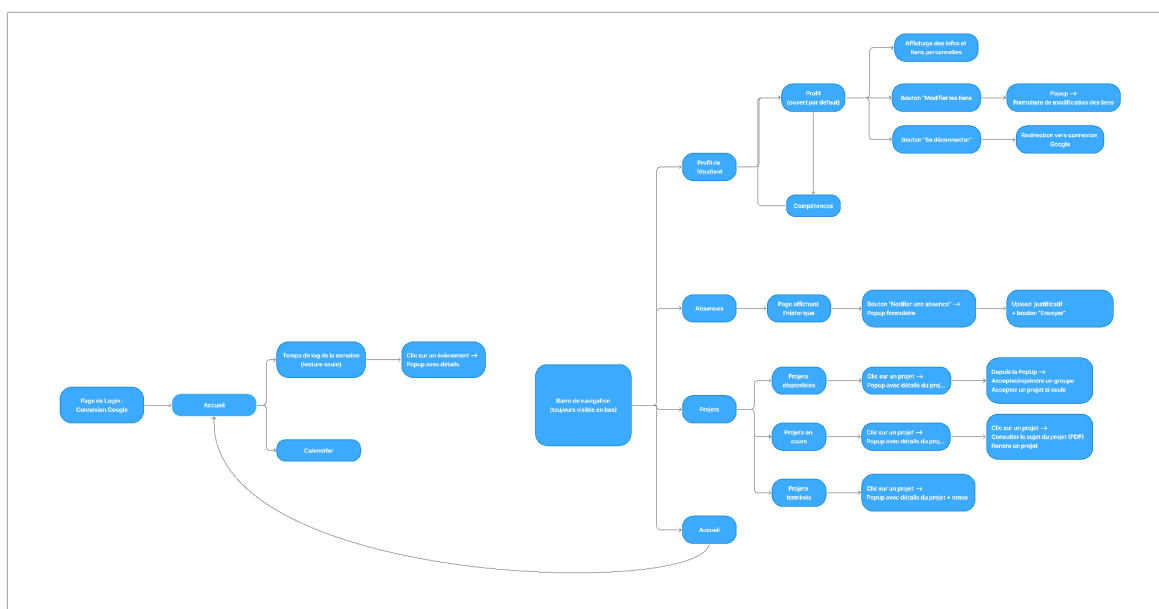
### III. Réalisation du projet

#### A. Conception et Maquettage de l'application

La première étape du projet a consisté à identifier les fonctionnalités essentielles attendues par les étudiants. Une maquette interactive a été réalisée avec Figma, afin de simuler le parcours utilisateur sur mobile. Ce travail a permis d'anticiper les ajustements de navigation, d'ergonomie et de contenus visuels.

#### 1. Userflow - Parcours utilisateur

Ensuite, nous avons commencé à imaginer un parcours utilisateur qui nous permettrait d'afficher les données et les actions disponibles, de manière cohérente et pratique.

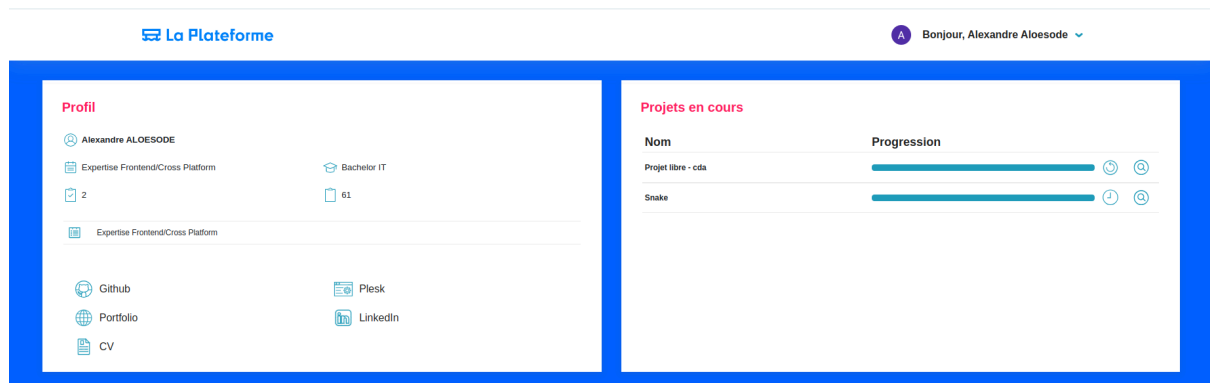


#### 2. Charte graphique

Nous avons pensé à SchoolTool en tant qu'application pouvant répondre aux besoins de toutes les écoles d'informatique. De ce

fait, l'objectif ici n'était pas de créer notre propre charte graphique, mais plutôt d'avoir un outil adaptable à celle du client.

Dans le cas de La Plateforme, nous avons donc par exemple repris le logo de l'école, ainsi que les couleurs principales utilisées sur les outils actuels.



### 3. Réalisation de la maquette

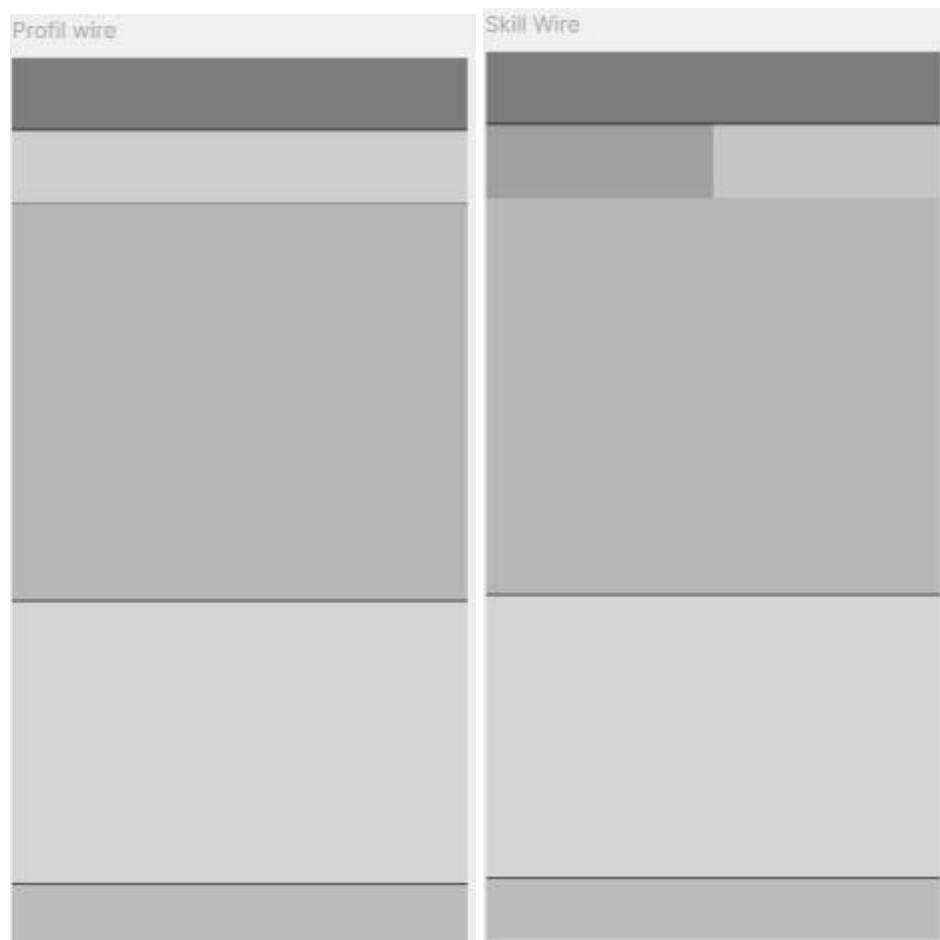
#### a) Modèle Wireframe

Avant de passer à l'aspect graphique, nous avons réalisé plusieurs wireframes de l'application.

Ceux-ci ont permis de définir les grands blocs fonctionnels de chaque écran (navigation, contenus, interactions) sans se soucier du style visuel.

L'objectif était de valider les parcours utilisateurs et la structure des pages avant d'appliquer une charte graphique.

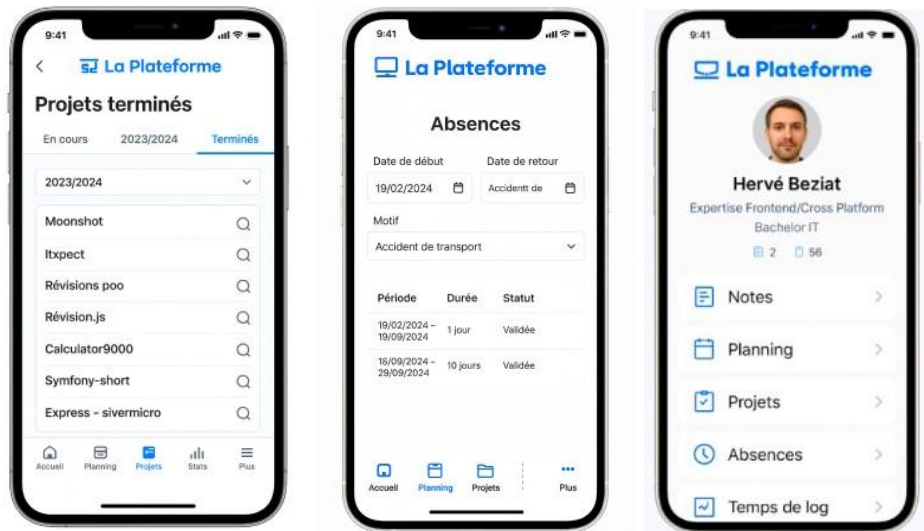




### b) Évolution vers la maquette finale

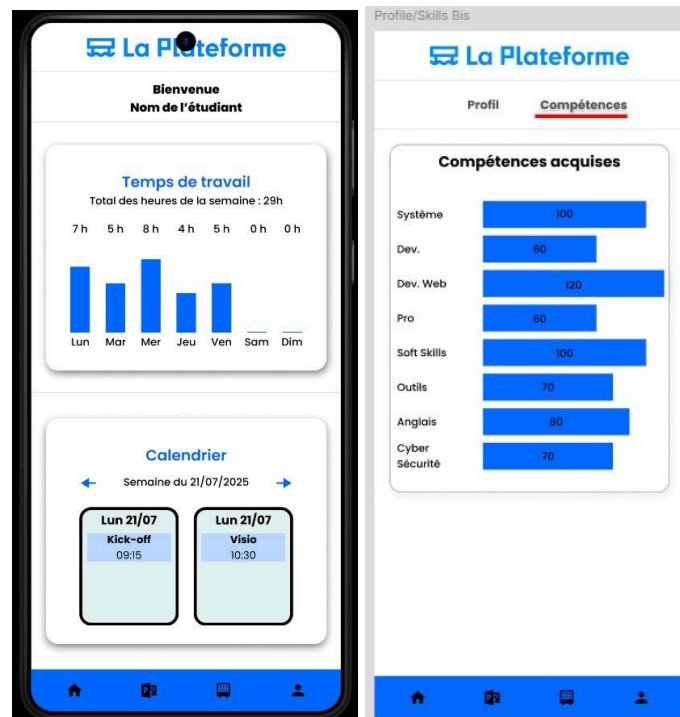
Une fois la structure validée, nous sommes passés à l'étape de maquettage graphique.

Dans un premier temps, nous avons élaboré une maquette générique, pensée pour être adaptable à différentes chartes graphiques.



Après plusieurs itérations, nous avons abouti à une version finale en appliquant la charte graphique de l'école, ce qui a permis d'aligner visuellement l'application avec l'identité visuelle de notre établissement.

Les captures ci-dessous présentent les écrans de l'application dans leur version finale.



## **B. Base de données**

La base de données existante de l'API La Plateforme est constituée de 31 tables, et certaines d'entre elles sont liées à de la logique métier concernant le côté administratif (commentaires internes du staff sur les étudiants, informations sur les promotions, calendriers de formation, etc...). Nous avons donc dû commencer par faire le tri dans les tables, afin de conserver uniquement les données pertinentes à un intranet étudiant.

Nous avons sélectionné 20 tables, que l'on peut classer de la manière suivante:

### **Informations utilisateurs**

On retrouve dans cette section la table applicant, qui stocke les données personnelles des personnes postulant à l'école (date de naissance, adresse, etc...), puis la table student, qui stocke des informations sur les personnes admises, liées à la vie étudiante

### **Parcours étudiant**

Ce sont toutes les tables qui regroupent les informations sur le parcours de l'étudiant, comme par exemple sa promotion, ses absences ou ses temps de présence

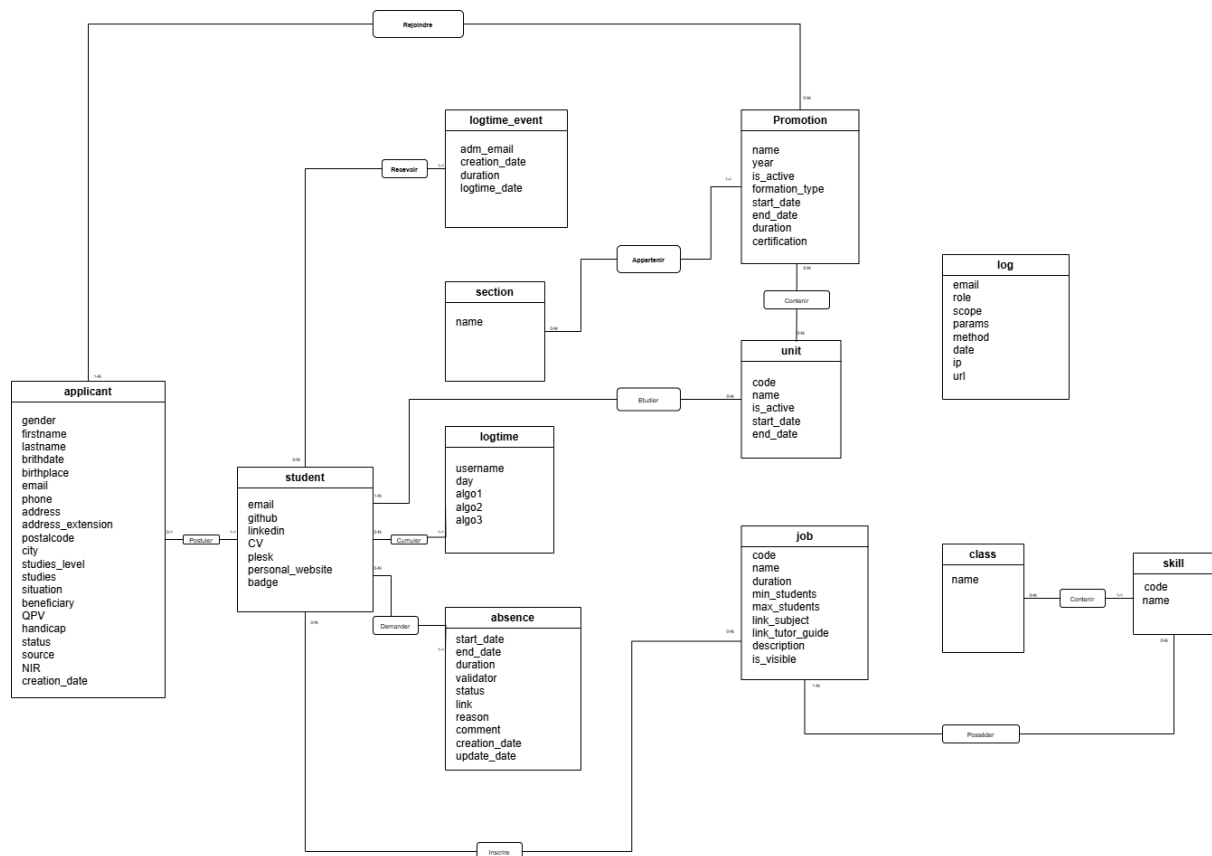
### **Projets**

Ici nous retrouvons les tables stockant les données des projets disponibles pour les étudiants, ainsi que leur notation et les compétences acquises.

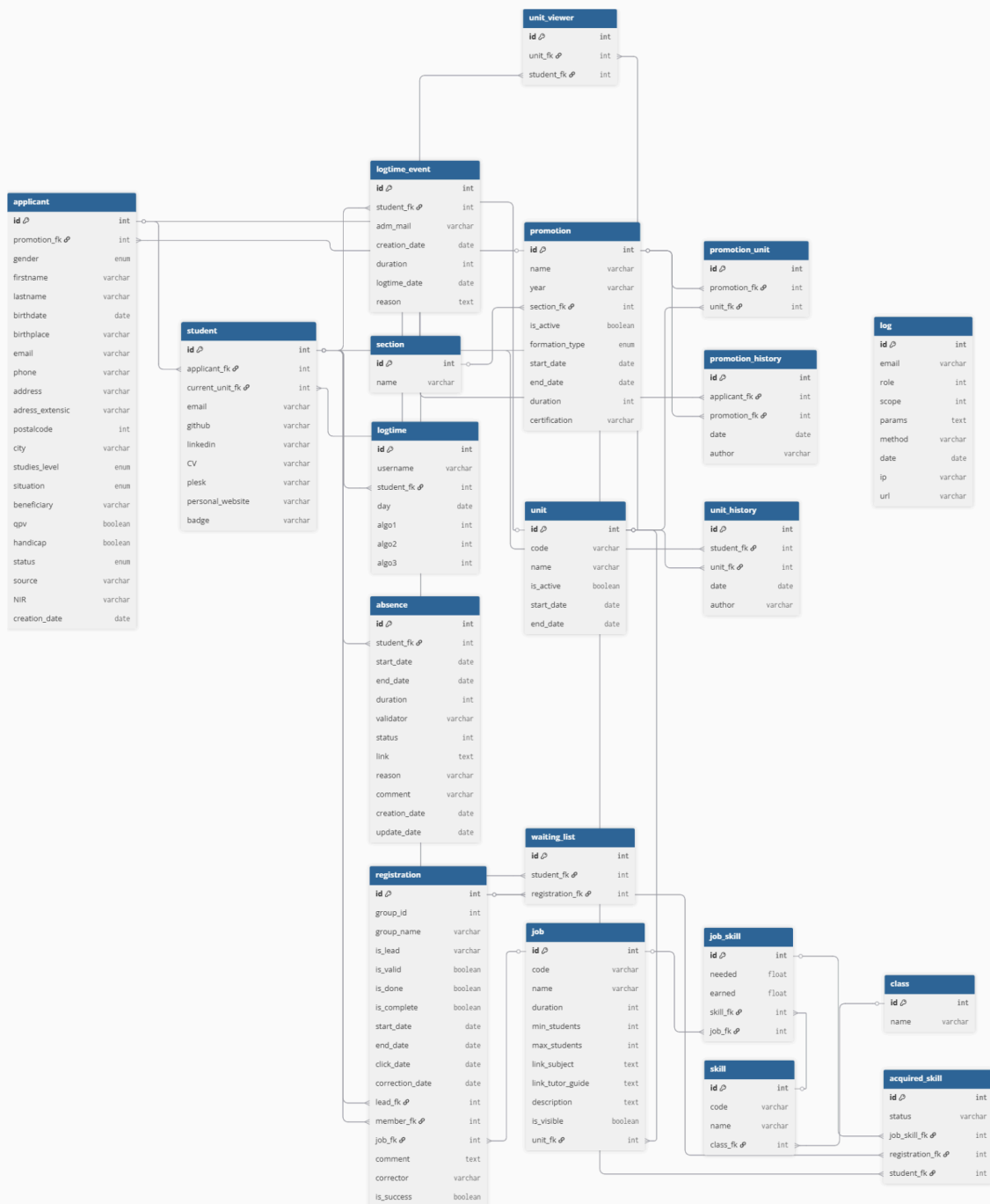
### **Enregistrement**

Enfin, nous avons la table log, qui archive l'ensemble des requêtes reçues par l'API

# 1. Modèle conceptuel de données



## 2. Modèle physique de données



## C. Normes de développement

Le projet est structuré en modules clairs : composants, services, contexts, routes, avec usage de expo-router pour une navigation déclarative et typée. L'intégration de TypeScript assure une robustesse et une maintenabilité du code. Les bonnes pratiques sont suivies : noms explicites, logique découplée, code commenté.

### 1. Architecture MVC vs MVP

Deux modèles d'architecture logicielle courants sont le MVC (Model-View-Controller) et le MVP (Model-View-Presenter). Bien qu'ils partagent des objectifs similaires – séparer la logique métier de l'interface utilisateur – leur organisation diffère.

#### **MVC (Model-View-Controller)**

- Model : contient les données et la logique métier.
- View : gère l'affichage.
- Controller : fait le lien entre la View et le Model. Il traite les entrées utilisateur et met à jour le modèle ou la vue.

#### **MVP (Model-View-Presenter)**

- Model : identique à MVC.
- View : interface utilisateur passive (n'a pas de logique).
- Presenter : contient toute la logique de présentation. Il interagit avec le Model et met à jour la View via une interface.

Dans SchoolTool, la logique métier (par exemple : appels API, authentification, filtrage des données) est séparée de l'interface graphique. L'application s'approche d'un modèle MVP pour les raisons suivantes:

- Les composants React Native sont principalement des Views : ils affichent les données reçues.
- La logique (appel API, traitement des réponses, gestion des états) est externalisée dans des hooks (useAuth), services (ApiActions) ou fonctions.
- Les composants sont passifs : ils ne savent pas comment les données sont récupérées, seulement comment les afficher.

Cela correspond donc à une architecture MVP, où le Presenter (ici, les hooks et services) s'occupe de tout ce qui ne concerne pas le rendu visuel, laissant les composants se concentrer uniquement sur l'affichage.

## D. Conteneurisation et environnement

Chaque composant de l'application (front, back, auth, db) est encapsulé dans un conteneur Docker. Le fichier docker-compose.yaml permet de lancer l'ensemble du projet en une seule commande, assurant un environnement stable et reproductible pour le développement.

## 1. Dockerisation

### a. Création des dockerfiles

L'API et le service d'authentification utilisent un Dockerfile commun basé sur Apache + PHP (avec CodeIgniter):

```
FROM --platform=linux/x86-64 php:7.3-apache

RUN apt-get update && apt-get upgrade -y \
    && apt-get install -y libzip-dev unzip \
    && docker-php-ext-install mysqli pdo pdo_mysql zip \
    && a2enmod rewrite

RUN chown -R www-data:www-data /var/www/html \
    && chmod -R 755 /var/www/html

EXPOSE 80

CMD ["apache2ctl", "-D", "FOREGROUND"]
```

*FROM php:7.4-apache:*

Utilise une image officielle Docker contenant PHP 7.4 avec Apache préinstallé. Idéale pour héberger une application PHP comme CodeIgniter.

*RUN docker-php-ext-install mysqli pdo pdo\_mysql:*

Installe les extensions PHP nécessaires pour interagir avec une base de données MySQL/MariaDB. Obligatoire pour que CodeIgniter puisse se connecter à la base.

*RUN a2enmod rewrite:*

Active le module mod\_rewrite d'Apache, essentiel pour le routing propre (URLs sans index.php) dans CodeIgniter.

*COPY . /var/www/html:*

Copie le contenu du dossier du projet (API ou Auth) dans le répertoire par défaut d'Apache (/var/www/html).

*WORKDIR /var/www/html:*



Définit le dossier de travail par défaut dans le conteneur. Toutes les commandes suivantes seront exécutées à partir de là.

#### *EXPOSE 80:*

Indique que le conteneur expose le port HTTP standard (80). Cela permet au docker-compose de le lier à un port externe (ex: 8000 ou 8001).

Le Dockerfile de l'application mobile sert à lancer le projet Expo dans un conteneur Node.js :

```
FROM node:18
WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install

COPY . .

COPY .env .env

EXPOSE 8081
EXPOSE 19000 19001 19002

CMD ["npx", "expo", "start", "--tunnel"]
```

#### *FROM node:20:*

Utilise une image officielle Node.js (version 20), adaptée pour les projets JavaScript/TypeScript comme React Native.

#### *WORKDIR /app:*

Crée et positionne le dossier de travail à /app dans le conteneur.

#### *RUN npm install:*

Installe toutes les dépendances définies dans le package.json, nécessaires au bon fonctionnement d'Expo et des librairies utilisées.

#### *EXPOSE 19000 19001 19002:*

Expose les ports utilisés par Expo :

- 19000 : serveur de développement
- 19001 : tunnel LAN
- 19002 : Expo DevTools dans le navigateur

*CMD ["npm", "start"]:*

Démarre le projet Expo avec la commande npm start, qui lance le serveur de développement.

### b. Création du docker-compose

Le fichier docker-compose.yaml centralise la configuration des 4 services principaux du projet :

1. **db** (base de données)
2. **back** (API Codelgniter)
3. **auth** (service d'authentification Codelgniter 4)
4. **front** (application mobile avec Expo Go)

```

services:
  db:
    image: mariadb
    command: --default-authentication-plugin=mysql_native_password
    restart: always
    ports:
      - 3307:3306
    environment:
      MYSQL_ROOT_PASSWORD: root

  back:
    build: ./back
    container_name: schooltool-back
    volumes:
      - ./back:/var/www/html
    ports:
      - '8000:80'
    depends_on:
      - db
    environment:
      TAB_ROLES: '[{"id": 1, "name": "student", "access": "Student_Access"}, {"id": 2, "name": "hors-parcours"}, {"id": 3, "name": "alumni"}, {"id": 4, "name": "admin"}]'
      DB_HOST: db
      DB_USER: root
      DB_PWD: root
      DB_NAME: schooltool_api
      LPTF_JWT_KEY: toto
      GOOGLE_CLIENT_ID: '604347883543-cu73up3fqo5r9gn18tqpkf3tu9ud41s4.apps.googleusercontent.com'
      GOOGLE_DRIVE_ABSENCE_FOLDER: 'toFill'
      IS_DEV: true

  auth:
    build: ./auth
    container_name: schooltool-auth
    volumes:
      - ./auth:/var/www/html
    ports:
      - '8001:80'
    depends_on:
      - db
    environment:
      TAB_ROLES: '[{"id": 1, "name": "student", "access": "Student_Access"}, {"id": 2, "name": "hors-parcours"}, {"id": 3, "name": "alumni"}, {"id": 4, "name": "admin"}]'
      DB_HOST: db
      DB_USER: root
      DB_PWD: root
      DB_NAME: schooltool_auth
      LPTF_JWT_KEY: toto
      GOOGLE_CLIENT_ID: '604347883543-cu73up3fqo5r9gn18tqpkf3tu9ud41s4.apps.googleusercontent.com'
      IS_DEV: true
      OAUTH_URL: 'http://localhost:8001/oauth'

  front:
    build: ./front
    container_name: schooltool-front
    tty: true
    stdin_open: true
    volumes:
      - ./front:/app
    ports:
      - '8002:8001'
      - '19000:19000' # Metro Bundler
      - '19001:19001' # WebSocket (live/hot reload)
      - '19002:19002' # React Native Debugger
    depends_on:
      - back
      - auth
    environment:
      API_URL: 'http://localhost:8000'
      AUTH_URL: 'http://localhost:8001'

  volumes:
    db_data:

```

Chaque service est défini avec ses propres variables d'environnement, ses ports, et ses volumes de développement. Grâce à docker-compose, un seul fichier permet de démarrer l'intégralité de la plateforme, de façon homogène et portable, quel que soit le poste de travail utilisé

## 2. Variables d'environnement

### a. Fichiers de constants pour l'auth et l'api

Dans les projets CodeIgniter utilisés pour l'API et le service d'authentification, les fichiers application/config/constants.php

jouent un rôle central dans la configuration de l'application. Ils permettent de définir des valeurs constantes accessibles globalement, sans avoir à les répéter dans tout le code.

Ces fichiers sont utilisés pour centraliser des paramètres critiques tels que :

- les clés secrètes pour la génération de tokens JWT (LPTF\_JWT\_KEY),
- les identifiants de clients Google (GOOGLE\_CLIENT\_ID),
- les rôles disponibles et leurs droits (TAB\_ROLES sous forme de tableau JSON encodé).

```
/*
 * http://ccept.org/2017/08/14/exit-codes.html
 */
defined('EXIT_SUCCESS') OR define('EXIT_SUCCESS', 0); // no errors
defined('EXIT_ERROR') OR define('EXIT_ERROR', 1); // generic error
defined('EXIT_CONFIG') OR define('EXIT_CONFIG', 3); // configuration error
defined('EXIT_UNKNOWN_FILE') OR define('EXIT_UNKNOWN_FILE', 4); // file not found
defined('EXIT_UNKNOWN_CLASS') OR define('EXIT_UNKNOWN_CLASS', 5); // unknown class
defined('EXIT_UNKNOWN_METHOD') OR define('EXIT_UNKNOWN_METHOD', 6); // unknown class member
defined('EXIT_USER_INPUT') OR define('EXIT_USER_INPUT', 7); // invalid user input
defined('EXIT_DATABASE') OR define('EXIT_DATABASE', 8); // database error
defined('EXIT_AUTO_MIN') OR define('EXIT_AUTO_MIN', 9); // lowest automatically-assigned error code
defined('EXIT_AUTO_MAX') OR define('EXIT_AUTO_MAX', 125); // highest automatically-assigned error code

/* API KEY */
defined('LPTF_JWT_KEY') OR define('LPTF_JWT_KEY', getenv('LPTF_JWT_KEY'));
// no errors
defined('IS_DEV') OR define('IS_DEV', getenv('IS_DEV'));

defined('DB_HOST') OR define('DB_HOST', getenv('DB_HOST'));
defined('DB_USER') OR define('DB_USER', getenv('DB_USER'));
defined('DB_PWD') OR define('DB_PWD', getenv('DB_PWD'));
defined('DB_NAME') OR define('DB_NAME', getenv('DB_NAME'));

defined('TAB_ABANDONS') or define('TAB_ABANDONS', [49, 78]);

defined('TAB_ROLES') or define('TAB_ROLES', json_decode($_ENV['TAB_ROLES'], true));

defined('GOOGLE_CLIENT_ID') or define('GOOGLE_CLIENT_ID', getenv('GOOGLE_CLIENT_ID'));
defined('GOOGLE_CLIENT_EMAIL') or define('GOOGLE_CLIENT_EMAIL', "YOUR CLIENT EMAIL");
defined('GOOGLE_PRIVATE_KEY') or define('GOOGLE_PRIVATE_KEY', "YOUR PRIVATE KEY");
defined('GOOGLE_DRIVE_ABSENCE_FOLDER') or define('GOOGLE_DRIVE_ABSENCE_FOLDER', "YOUR FOLDER ID");
```

Ces constantes sont ensuite utilisées dans le code pour sécuriser les échanges (signature des tokens), filtrer les droits d'accès ou vérifier l'origine des utilisateurs connectés. En les associant à des variables d'environnement via getenv(), elles permettent aussi de préserver la confidentialité des données sensibles et d'adapter l'application à différents environnements sans modifier le code.

## b. Fichier .env pour le front

Le fichier .env situé à la racine du projet frontend permet de définir toutes les variables d'environnement nécessaires au fonctionnement de l'application mobile. Il est utilisé conjointement avec app.config.js pour injecter dynamiquement les paramètres lors du build de l'application avec Expo.

Ce fichier contient notamment :

- les URLs des services API et Auth
- les identifiants Google
- la clé secrète Google si nécessaire

Ce fichier est exclu du dépôt Git (via .gitignore) pour éviter de compromettre les données sensibles. Il peut être facilement adapté à différents environnements (développement, préproduction, production) en changeant simplement les valeurs des variables.

Cette approche assure une séparation claire entre la configuration et le code, une meilleure sécurité et une portabilité accrue du projet.

## c. Gestion des secrets en développement

La gestion des informations sensibles (identifiants, clés secrètes, tokens) est un enjeu fondamental pour la sécurité d'un projet. Dans SchoolTool, cette problématique est traitée en séparant clairement le code source de la configuration confidentielle, à l'aide de fichiers .env ou constants.php et de variables d'environnement.

Les avantages de cette approche sont les suivants:

- Aucune donnée sensible n'est codée en dur dans le projet.

- Il est facile de changer de configuration selon l'environnement : développement local, build Android, production.
- Le système est compatible avec les outils modernes comme EAS Build, GitHub Actions ou Docker.

Grâce à cette méthode, SchoolTool respecte les bonnes pratiques DevOps pour protéger les secrets en développement tout en gardant un déploiement fluide et reproductible.

## E. Intégration back-end

Comme mentionné précédemment, l'une des contraintes de ce projet fût de s'adapter aux outils existants de l'école. Notre application devait donc respecter diverses règles que nous allons tenter de détailler dans cette section.

### 1. Intégration de l'Authentification La Plateforme

L'outil d'Authentification de La Plateforme se base sur le protocole OAuth2 via Google.

OAuth (Open Authorization) est un protocole de délégation d'autorisation. Il permet à une application tierce d'accéder à certaines données d'un utilisateur stockées sur un autre service, sans avoir à connaître ses identifiants. L'utilisateur donne son consentement explicite, et l'application obtient un jeton sécurisé (token) limité à certaines actions (scopes).

#### Fonctionnement de Google OAuth dans SchoolTool:

- L'utilisateur clique sur "Se connecter avec Google".
- Il est redirigé vers une page d'authentification Google.

- Une fois connecté, Google retourne un ID Token à l'application front-end (mobile).
- Ce token est ensuite envoyé au back-end, à un point d'entrée appelé /oauth, pour vérification.

```

class OAuth extends LPTF_Controller
{
    public function index()
    {
        header('Access-Control-Allow-Origin: *');

        $token_id = $this->input->post("token_id");
        $client = new Google_Client();
        try {
            $payload = $client->verifyIdToken($token_id, GOOGLE_CLIENT_ID);
        } catch (Exception $e) {
            echo json_encode(["error" => "You are not one of us."]);
            return (false);
        }
        if ($payload) {
            $google_id = $payload['sub'];
            $email = $payload['email'];
            $firstname = $payload['given_name'];
            $lastname = $payload['family_name'];
            // $locale = $payload['locale'];
            if (isset($payload['hd']))
                $domain = $payload['hd'];
            else
                $domain = "gmail.com";
        } else {
            echo json_encode(["error" => "You are not one of us."]);
            return (false);
        }

        $user = $this->User_model->GetUser(array('email' => $email));
        if (empty($user)) {
            echo json_encode(["error" => "You are not one of us."]);
            return (false);
        }

        // Récupération du scope
        $scope = $this->User_model->GetUserScope(array("scope.user_id" => $user[0]['id']));
        $scopes = array();
        foreach ($scope as $value) {
            array_push($scopes, $value["scope_value"]);
        }

        $jwt = new JWT();
        $token_data = [
            'user_id' => $user[0]['id'],
            'user_email' => $user[0]['email'],
            'role' => $user[0]['role'],
            'scope' => $scopes,
            'exp' => time() + (60 * 15)
            // 'exp' => time() + 15
        ];

        $api_token = $jwt->generate($token_data);

        // generation du authtoken
        $secret = random_string('alnum', 128);
        $this->User_model->GenerateSecret(array("user_id" => $user[0]['id'], "secret" => $secret));

        $authjwt = new JWT();
        $token_data = [
            'secret' => $secret
        ];
        $authtoken = $authjwt->generate($token_data);

        echo json_encode(["authtoken" => $authtoken, "token" => $api_token]);
    }
}

```

Le token ID est vérifié grâce à la bibliothèque officielle Google\_Client. Si le token est invalide, la requête est rejetée. En cas de succès, on récupère l'identifiant Google, l'email, le prénom,



le nom, et le domaine (utile pour filtrer les domaines autorisés). L'utilisateur est ensuite recherché par son email dans la base de données de l'école via le modèle `User_model`. Si l'email ne correspond pas à un compte connu, l'accès est refusé.

Si les étapes précédentes sont validées, Un JSON Web Token (JWT) est généré avec :

- `user_id`
- `email`
- `rôle`
- `scope` (droits associés)
- `exp` (expiration à 15 min)

Ce token est utilisé pour sécuriser les appels API à l'application principale.

Un second JWT est généré avec un secret aléatoire stocké en base, pour servir de jeton d'authentification complémentaire et renforcer la sécurité.

Les deux tokens sont enfin envoyés au front-end. Le premier servira à tous les appels API, et le deuxième sera utilisé pour rafraîchir le premier toutes les quinze minutes.

## [2. Intégration de l'API La Plateforme](#)

Les appels API sont gérés et sécurisés par le biais du premier jeton d'authentification. Le point d'entrée est donc une classe permettant de:

- Vérifier la validité du token reçu
- Vérifier si le token n'est pas expiré

- Enregistrer la requête dans une table de logs
- Rediriger la requête vers la route concernée

You, 3 months ago | 1 author (You)

<?php

You, 3 months ago | 1 author (You)

```
class LPTF_Controller extends CI_Controller
{
    static $Status_Helper;

    public function __construct()
    {
        parent::__construct();

        $this->handleHeader();

        include APPPATH . 'third_party/JWT/JWT.php';
        $this->jwt = new JWT();

        $this->load->database();
        $this->load->model('Log_Model');
        $this->load->helper('Status_Helper');
        $this->load->helper('Token_Helper');

        LPTF_Controller::$Status_Helper = new Status_Helper();

        $this->token_helper = new Token_Helper($this);

        if (!$this->token_helper->verify_token()) {
            $this->Status()->ExpectationFailed();
            $this->logAction();
            die();
        } elseif (!$this->token_helper->token_valid()) {
            $this->Status()->TokenExpired();
            $this->logAction();
            die();
        } else {
            $this->load->helper('Role_Helper');
            $this->role_helper = new Role_Helper($this);
            $this->logAction($this->token_helper->controler->jwt->get_payload());
        }
    }
}
```

Lorsque la requête passe les différents contrôles, CodeIgniter offre un système de routing qui se présente de la manière suivante:

```

routes.php X
back > application > config > routes.php > ...

51  */
52
53  $route['default_controller'] = 'main';
54  $route['404_override'] = '';
55  $route['translate_uri_dashes'] = FALSE;
56
57  /* CLASS */
58  $route['class'] = 'Classe/Class';
59
60  /* UNIT */
61  $route['unit'] = 'Unit/Unit';
62  $route['unit/job'] = 'Unit/UnitJob';
63  $route['unit/duplicate'] = 'Unit/UnitDuplicate';
64  $route['unit/end'] = 'Unit/UnitEnd';
65
66  /* PROMOTION */
67  $route['promotion'] = 'Promotion/Promotion';
68  $route['promotion/end'] = 'Promotion/PromotionEnd';
69  $route['promotion/faithfulness'] = 'Promotion/PromotionFaithfulness';
70
71  /* SECTION */
72  $route['section'] = 'Section/Section';
73
74  /* SKILL */
75  $route['skill'] = 'Skill/Skill';
76
77  /* JOB */
78  $route['job'] = 'Job/Job';
79  $route['student/job/available'] = 'Job/StudentJobAvailable';
80  $route['job/student/available'] = 'Job/JobStudentAvailable';
81
82  /* JOB_SKILL */
83  $route['job/skill'] = 'Job_Skill/JobSkill';
84
85  /* REGISTRATION */
86  $route['registration'] = 'Registration/Registration';
87  $route['job/student'] = 'Registration/JobStudent';
88  $route['job/await'] = 'Registration/getJobAwait';
89  $route['job/progress'] = 'Registration/getJobProgress';
90  $route['job/ready'] = 'Registration/getJobReady';
91  $route['job/checked'] = 'Registration/getJobChecked';
92  $route['job/done'] = 'Registration/getJobDone';
93  $route['group'] = 'Registration/Group';
94  $route['member'] = 'Registration/Member';
95  $route['group/review'] = 'Registration/GroupReview';
96  $route['group/validity'] = 'Registration/GroupValidity';
97  $route['group/available'] = 'Registration/GroupAvailable';
98  $route['group/click'] = 'Registration/GroupClick';
99  $route['job/corrector'] = 'Registration/JobCorrector';
100 $route['job/review'] = 'Registration/JobReview';
101 $route['unit/review'] = 'Registration/UnitReview';
102
103 /* STUDENT */
104 $route['student'] = 'Student/Student';
105 $route['student/activity'] = 'Student/StudentActivity';
106 $route['student/inactive'] = 'Student/StudentInactive';
107 $route['student/attendance'] = 'Student/StudentAttendance';
108 $route['group/attendance'] = 'Student/GroupAttendance';
109 $route['student/last_log'] = 'Student/LastLog';

```

Dans ce fichier, CodeIgniter permet tout d'abord de récupérer l'endpoint de la requête. Prenons l'exemple d'une requête `getStudent` faite pour récupérer les informations d'un étudiant. L'endpoint `student` est donc retrouvé via la ligne

```
$route['student'] = 'Student/Student';
```

La syntaxe `'Student/Student'` permet ensuite de renvoyer à un nom de Contrôleur, et à une fonction interne à ce dernier.

```
back > application > controllers > Student.php > Student > putStudent
You, 5 months ago | 1 author (You)
1 <?php
2 defined('BASEPATH') or exit('No direct script access allowed');
3
You, 5 months ago | 1 author (You)
4 class Student extends LPTF_Controller
5 {
6
7     public function __construct()
8     {
9         parent::__construct();
10
11         $this->load->model('Student_Model');
12     }
13
14     public function index()
15     {
16         echo json_encode(["whoami" => "api"]);
17     }
18
19     public function Student()
20     {
21         $method = $_SERVER['REQUEST_METHOD'];
22         $actions = [
23             'GET' => 'getStudent',
24             'POST' => 'postStudent',
25             'PUT' => 'putStudent',
26             'DELETE' => 'deleteStudent'
27         ];
28
29         if (array_key_exists($method, $actions)) {
30             $call = $actions[$method];
31             $response = $this->$call();
32         } else {
33             $response = $this->Status()->BadMethod();
34         }
35         echo json_encode($response);
36     }
37
38     private function getStudent()
39     {
40         $params = $this->input->get();
41         if ($this->role_helper->Access($params) == true) {
42             return ($this->Student_Model->getStudent($params));
43         } else {
44             return ($this->Status()->Denied());
45         }
46     }
47
48     private function postStudent()
49     {
50         $params = $this->input->input_stream();
51         if ($this->role_helper->Access($params) == true) {
52             return ($this->Student_Model->postStudent($params));
53         } else {
54             return ($this->Status()->Denied());
55         }
56     }
57 }
```

Nous retrouvons ici la fonction Student, contenue par un Contrôleur nommé Student. Cette fonction commence par définir les types de requêtes accessibles via le Modèle Student (GET, POST, PUT, DELETE).

Sont présentes par la suite les fonctions inhérentes aux méthodes disponibles. C'est dans celles-ci que, avant de faire appel au Modèle concerné, l'API utilise un système d'accès permettant de sécuriser et restreindre si besoin l'accès aux données.

En effet, le token reçu par l'api contient plusieurs informations, notamment:

- le rôle de l'utilisateur
- son scope (en l'occurrence, dans le cas d'un rôle student, son Id (identifiant dans la table Student))

Chaque rôle dispose donc d'un fichier d'accès, qui définit l'accessibilité aux Modèles.

```

...
<?php
...
class Student_Access
{
    private $controller;

    public function __construct(&$controller)
    {
        $this->controller = &$controller;
    }

    public function access($call, $payload, &$params)
    {
        $granted = [
            'getClass',
            'getPromotion',
        ];

        $limited = [
            'getAbsence',
            'postAbsence',
            'getActivityAttendance',
            'getAlert',
            'getApplicant',
            'getStudentCalendar',
            'getCalendarDay',
            'getJob',
            'getJobAwait',
            'getJobDone',
            'getJobProgress',
            'getJobReady',
            'getJobSkill',
            'getLogtime',
            'getLogtime_Event',
            'getPromotion',
            'getPromotionHistory',
            'getRegistration',
            'getStudent',
            'putStudent',
            'getStudentUnit',
            'getUnitJob',
            'getUnitGoal',
            'getPromotionUnit',
        ];

        if (in_array($call, $granted)) {
            return (true);
        }

        if (in_array($call, $limited)) {
            return ($this->$call($payload, $params));
        }

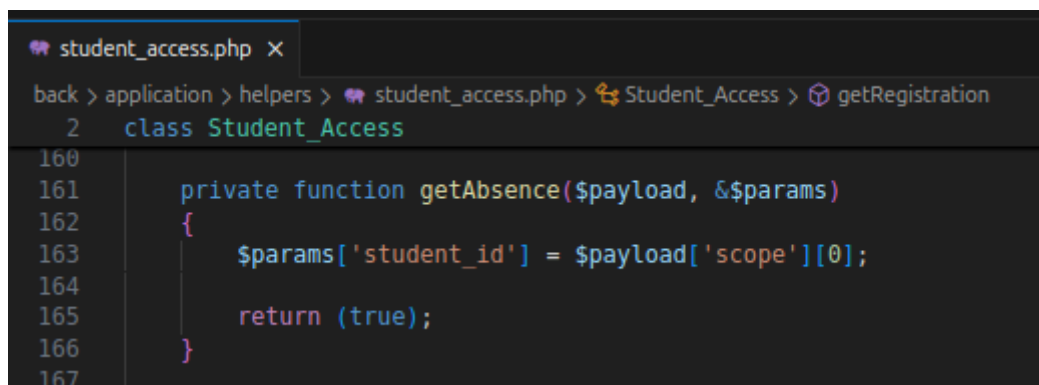
        return (false);
    }
}

```

Dans la fonction access, nous retrouvons donc deux tableaux:

- granted: les méthodes autorisées sans aucune restriction
- limited: les méthodes pour lesquelles une restriction sera effectuée

Dans le cas par exemple d'un appel à GetAbsence, l'API appellera ensuite une fonction permettant d'ajuster les paramètres de la requête.



```
student_access.php X
back > application > helpers > student_access.php > Student_Access > getRegistration
2 class Student_Access
160
161     private function getAbsence($payload, &$params)
162     {
163         $params['student_id'] = $payload['scope'][0];
164
165         return (true);
166     }
167
```

On voit ici que par défaut, en cas d'appel à GetAbsence par un utilisateur de type "student", l'API ajoutera automatiquement à la requête un paramètre "student\_id". Ainsi, les données sont sécurisées et l'API ne permet pas à un étudiant de récupérer les absences d'autres personnes de l'école.

Pour finir, en cas de validation par le fichier d'accès, le Modèle concerné est appelé.

## F. Développement front-end

### 1. Architecture du front-end

Le front-end repose sur une architecture claire, modulaire et orientée métier, respectant les principes de bases d'un projet React Native + Expo. Tout a été pensé dans le but de faciliter la maintenance, la réutilisabilité des composants et l'évolutivité de l'application.

## **Structure générale du projet**

Le code source de l'application est structuré selon une logique de domaines fonctionnels et de responsabilités séparées. Voici les principaux répertoires :

### *components/*

Ce dossier regroupe les composants d'interface utilisateur. Il est organisé par contexte métier :

- `auth/`, `absences/`, `dashboard/`, `jobs/`, etc. : chaque sous-dossier contient les composants spécifiques à une fonctionnalité.
- `global/` : composants transverses, comme `Header.tsx`.
- `modals/` : modales réutilisables (ex. pour afficher un projet).

### *hooks/*

Contient les hooks personnalisés facilitant l'accès à l'état d'authentification (`useAuth.ts`).

### *services/*

Implémente les services métier, notamment `ApiServices.ts`, qui encapsule les appels à l'API REST avec `axios` et gère automatiquement l'authentification (token, rafraîchissement, redirection, etc.).

### *types/*

Contient les définitions de types TypeScript pour structurer les données échangées (authentification, jobs, etc.).



## *utils/*

Fonctions utilitaires génériques utilisées dans toute l'application (décodage JWT, gestion de l'environnement, manipulation de sessions...).

### **Points techniques notables**

- Navigation basée sur expo-router : permet une organisation simple des routes et une gestion dynamique des pages.
- Sécurité : la logique de session (via JWT) est sécurisée localement, avec vérification d'expiration, rafraîchissement automatique et stockage sécurisé.
- Composants modulaires : chaque domaine fonctionnel est isolé, ce qui facilite les tests, le debug et l'évolution future.
- Tests unitaires : dossier `__tests__` dans lequel sont stockés les tests que nous avons développés.

### **Avantages de l'architecture**

- Clarté et lisibilité du code.
- Séparation des responsabilités facilitant la maintenance.
- Adaptabilité à de nouvelles fonctionnalités sans dette technique importante.
- Préparation au scaling : gestion centralisée des API, des contextes et des types.

## 2. Authentication Google OAuth

Nous avons précédemment évoqué le système d'authentification dans la partie back-end. Notre tâche pour SchoolTool fût donc de développer un système de connexion qui puisse fonctionner avec l'authentification La Plateforme.

La première étape était donc d'envoyer à l'API Google une requête comprenant l'identifiant client La Laplateforme.

```
const [request, response, promptAsync] = AuthSession.useAuthRequest(  
  {  
    clientId: googleClientId,  
    redirectUri,  
    usePKCE: true,  
    scopes: [  
      'openid',  
      'profile',  
      'email',  
      'https://www.googleapis.com/auth/calendar.readonly',  
    ],  
    extraParams: {  
      access_type: 'offline',  
      prompt: 'consent',  
      response_type: 'code',  
    },  
  },  
  { authorizationEndpoint: 'https://accounts.google.com/o/oauth2/v2/auth' },  
);
```

Nous avons également intégré l'API Google Calendar, puisque c'est l'outil utilisé pour la gestion des calendriers de formation.

Cette étape permet de récupérer un code Google nécessaire à l'étape suivante, qui est de récupérer une token\_id requise par l'authentification La Plateforme.

```

const exchangeCodeForToken = async (code: string) => {
  try {
    const response = await fetch('https://oauth2.googleapis.com/token', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      },
      body: new URLSearchParams({
        code,
        client_id: googleClientId,
        client_secret: isExpoGo ? undefined : googleSecret,
        redirect_uri: redirectUri,
        grant_type: 'authorization_code',
        code_verifier: request?.codeVerifier || '',
      }).toString(),
    });

    const tokenData = await response.json();

    if (!tokenData.access_token) {
      Alert.alert('Erreur', 'Token Google non reçu');
      return;
    }
  }
}

```

Une fois la token\_id Google récupérée, nous avons pu mettre en place une requête vers l'authentification La Plateforme.

```

const authToken = await fetch(`${authUrl}/oauth`, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded',
  },
  body: `token_id=${tokenData.id_token}`,
});

const apiToken = await authToken.json();

const decoded = decodeJWT(apiToken.token);
if (!decoded) {
  Alert.alert('Erreur', 'Impossible de décoder le token utilisateur');
  return;
}

```

Enfin, une fois le token d'authentification et le token d'accès à l'API reçus, nous avons pu stocker:

- Les tokens La Plateforme et Google, nécessaires pour les requêtes vers l'API et le refresh des tokens
- Les informations utilisateur

```
const userSession = {
  accessToken: apiToken.token,
  authToken: apiToken.authToken,
  googleAccessToken: tokenData.access_token,
  googleExpiresIn: tokenData.expires_in,
  googleExpiresAt: Date.now() + tokenData.expires_in * 1000,
  googleScope: tokenData.scope,
  googleRefreshToken: tokenData.refresh_token,
};

const userData = {
  id: decoded.user_id,
  email: decoded.user_email,
  role: decoded.role,
  scope: decoded.scope,
};
```

### 3. Systèmes de requêtes

#### a) Fetch

Comme vu précédemment, nous avons utilisé *fetch*, un système de requête interne à Javascript, pour faire les requêtes vers Google et l'authentification La Plateforme.

#### b) Axios

Axios est une bibliothèque JavaScript populaire qui permet de faire des requêtes HTTP davantage poussées que fetch, qui présente les principales fonctionnalités suivantes:

- Léger et basé sur les Promesses (comme fetch, mais avec plus de fonctionnalités).
- Utilisé pour interagir facilement avec des API REST (GET, POST, PUT, DELETE...).
- Supporte la configuration des en-têtes, l'interception des requêtes/réponses, le timeout, l'annulation, etc.

- Gère automatiquement la conversion des données JSON.

Axios peut être utilisé pour créer un service personnalisé, ce qui est exactement ce dont nous avons besoin dans Schooltool, afin de respecter les contraintes de l'API.

### c) Fichier utilitaire de requêtes API

Dans l'architecture de SchoolTool, le module ApiActions agit comme un intermédiaire universel entre le frontend React Native et l'API backend CodeIgniter. Il centralise la logique des appels API HTTP (GET, POST, PUT, DELETE) tout en respectant les contraintes spécifiques de sécurité, de structure, et de gestion de session imposées par le backend.

### **Gestion du token d'authentification**

Avant chaque requête :

- Le module tente d'obtenir le token accessToken via la session stockée (Session.getSession()).
- Il vérifie si ce token est expiré via un décodage local JWT.
- Si le token est expiré, une requête de rafraîchissement est envoyée (/refresh), exploitant le authToken.
- En l'absence de token valide, l'utilisateur est automatiquement redirigé vers la page d'accueil (router.push('/')).

```

async function getApiToken() {
  try {
    const session = await Session.getSession();

    if (!session) {
      router.push('/');
      return null;
    }

    if (isTokenExpired(session.accessToken)) {
      const newToken = await refreshToken(session);
      if (newToken) {
        await Session.updateAccessToken(newToken);
        return newToken;
      } else {
        await Session.clear();
        router.push('/');
        return null;
      }
    }

    return session.accessToken;
  } catch (error) {
    console.error('Error fetching session: ', error);
    return null;
  }
}

```

## **Adaptation aux contraintes de CodeIgniter**

L'API centralise toutes les méthodes dans une fonction du Contrôleur, qui agit dynamiquement selon le verbe HTTP (`$_SERVER['REQUEST_METHOD']`). Pour se conformer à cela, ApiActions :

- Inclut toujours les données dans data (même pour les DELETE), car CodeIgniter utilise `$this->request->getRawInput()` qui lit le corps brut de la requête.
- Utilise multipart/form-data pour le rafraîchissement (comme l'exige le endpoint /refresh), et application/x-www-form-urlencoded pour les autres requêtes.
- Construit dynamiquement l'URL en fonction de la méthode fournie et enchaîne les paramètres dans l'URL (GET) ou dans

le corps (POST, PUT, DELETE) selon les besoins.

- Ajoute manuellement un header Token (non standard), attendu par l'API pour la vérification d'accès.

### **Structure de la requête**

Chaque méthode (get, post, put, delete) suit le même schéma :

- Vérification/rafraîchissement du token
- Construction de l'URL finale
- Encodage des paramètres (URLSearchParams)
- Requête via `axios.request(...)`

```

export const ApiActions = {
  async get(payload: ApiPayload): Promise<AxiosResponse | null> {
    let route = payload.route;
    let url = buildUrl(payload.params);

    const token = await getApiToken();
    if (!token) {
      router.push('/');
      return null;
    }

    let params = {
      method: 'get',
      maxBodyLength: Infinity,
      url: `${ENV.LPTF_API_URL}/${route}?${url}`,
      headers: {
        Token: token || '',
      },
    };
    try {
      const response = await axios.request(params);
      return response;
    } catch (error) {
      console.error('GET request error: ', error);
      throw error;
    }
  },

  async post(payload: ApiPayload): Promise<AxiosResponse | null> {
    let route = payload.route;
    const body = payload.params;

    const token = await getApiToken();
    if (!token) {
      router.push('/');
      return null;
    }

    let params = {
      method: 'post',
      maxBodyLength: Infinity,
      url: `${ENV.LPTF_API_URL}/${route}?`,
      data: new URLSearchParams(body).toString(),
      headers: {
        Token: token || '',
      },
    };
    try {
      const response = await axios.request(params);
      return response;
    } catch (error) {
      console.error('POST request error: ', error);
      throw error;
    }
  },
};

```



Les requêtes GET suivent un cheminement légèrement différent, puisque l'API, pour cette méthode, lit les paramètres directement dans l'url.

```
function buildUrl(  
  params: Record<string, string | number | (string | number)[]>,  
) : string {  
  let url = '';  
  Object.keys(params).forEach((key) => {  
    if (Array.isArray(params[key])) {  
      params[key].forEach((element) => {  
        url += `${key}[]={element}&`;  
      });  
    } else {  
      url += `${key}=${params[key]}&`;  
    }  
  });  
  return url;  
}
```

Par exemple, la requête suivante:

```
ApiActions.get({  
  route: 'absence/Absence',  
  params: { student_id: '123' }  
});
```

génèrera : [http://localhost:8000/absence/Absence?student\\_id=123](http://localhost:8000/absence/Absence?student_id=123)

## 4. Typage

Le projet SchoolTool est entièrement développé en TypeScript, un sur-ensemble typé de JavaScript. Ce choix technique apporte des avantages majeurs à plusieurs niveaux, notamment en termes de robustesse, scalabilité et compatibilité multi-plateformes (Android & iOS).

## Objectifs du typage

L'usage de TypeScript permet de :

- Sécuriser les échanges entre le frontend et le backend grâce à des types explicites.
- Réduire les bugs au moment de la compilation plutôt qu'à l'exécution.
- Améliorer la lisibilité et la documentation du code.
- Faciliter l'auto-complétion dans les IDE (VS Code, WebStorm).
- Garantir une portabilité fluide sur Android et iOS via Expo en assurant que toutes les données sont bien formatées et typées avant le build.

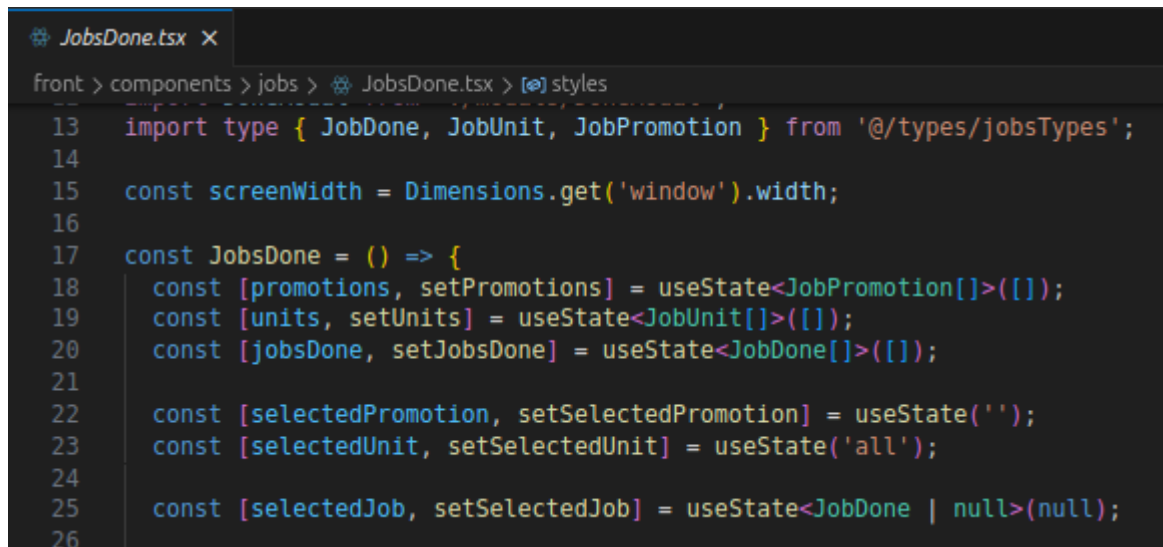
## Exemple typique d'une interface :

```
export interface JobDone {  
  job_name: string;  
  registration_id: number | string;  
  job_unit_id: number | string;  
  job_unit_name: string;  
  job_description?: string;  
  job_is_complete?: boolean | string;  
  start_date?: string;  
  end_date?: string;  
  group_name?: string;  
  lead_email?: string;  
  click_date?: string;  
  job_id?: number | string;  
  group_id?: number | string;  
}
```

Ici, chaque champ est précisément défini, y compris les champs optionnels (?) et les unions de types (number | string) pour couvrir les cas issus de l'API.

## Intégration avec les composants

Ces types sont ensuite utilisés dans l'ensemble du code :



```
front > components > jobs > JobsDone.tsx > styles
13 import type { JobDone, JobUnit, JobPromotion } from '@types/jobsTypes';
14
15 const screenWidth = Dimensions.get('window').width;
16
17 const JobsDone = () => {
18   const [promotions, setPromotions] = useState<JobPromotion[]>([]);
19   const [units, setUnits] = useState<JobUnit[]>([]);
20   const [jobsDone, setJobsDone] = useState<JobDone[]>([]);
21
22   const [selectedPromotion, setSelectedPromotion] = useState('');
23   const [selectedUnit, setSelectedUnit] = useState('all');
24
25   const [selectedJob, setSelectedJob] = useState<JobDone | null>(null);
26 }
```

## Avantage pour les builds Expo (Android & iOS)

Dans un contexte multi-plateforme :

- TypeScript évite les erreurs liées à des types imprévus ou mal formatés, notamment sur Android (où les erreurs JS sont moins tolérées).
- Il assure un build plus stable avec Expo, en détectant en amont les incohérences (ex. undefined non géré dans des données critiques).
- Il rend le déploiement en production plus fiable, sans surprises au runtime.

## G. Tests et qualité logicielle

### 1. Linting

```
root: true,
env: {
  browser: true,
  es2021: true,
},
extends: [
  'eslint:recommended',
  'plugin:@typescript-eslint/recommended',
  'plugin:react/recommended',
  'plugin:react-native/all',
  'plugin:prettier/recommended',
],
parser: '@typescript-eslint/parser',
parserOptions: {
  ecmaFeatures: {
    jsx: true,
  },
  ecmaVersion: 'latest',
  sourceType: 'module',
},
plugins: ['react', 'react-native', '@typescript-eslint', 'prettier'],
rules: {
  'react/prop-types': 'off',
  'react/react-in-jsx-scope': 'off',
  '@typescript-eslint/no-var-requires': 'off',
  'prettier/prettier': [
    'error',
    {
      singleQuote: true,
      trailingComma: 'all',
      semi: true,
      tabWidth: 2,
      printWidth: 80,
    },
  ],
},
settings: {
  react: {
    version: 'detect',
  },
},
};
```

Dans ce fichier de configuration d'ESLint, on retrouve notamment dans la partie *plugins*, l'utilisation de prettier pour formater le code selon des règles prédéfinies.

## 2. Tests des composants

Des tests de composants ont été mis en place afin de valider le bon fonctionnement des interfaces utilisateurs. Ces tests s'appuient sur la bibliothèque Jest associée à React Native Testing Library, ce qui permet de simuler l'affichage, les interactions utilisateur et les effets déclenchés par le cycle de vie du composant (useEffect, useState, etc.).

Ces tests permettent de s'assurer que :

- Les composants réagissent correctement aux données entrantes.
- Les interactions utilisateur fonctionnent comme prévu.
- Les éventuelles régressions soient détectées automatiquement.

## Exemple détaillé: JobsAvailable.test.tsx

```
1 import React from 'react';
2 import { render, waitFor, fireEvent } from '@testing-library/react-native';
3 import JobsAvailable from '../jobs/JobsAvailable';
4 import { ApiActions } from '@services/ApiServices';
5
6 jest.mock('@services/ApiServices', () => ({
7   ApiActions: {
8     get: jest.fn(),
9   },
10 }));
11
12 const mockJobs = [
13   {
14     job_id: '1',
15     job_name: 'Développeur Frontend',
16     job_unit_name: 'Unit 1',
17   },
18   {
19     job_id: '2',
20     job_name: 'Développeur Backend',
21     job_unit_name: 'Unit 2',
22   },
23 ];
24
25 describe('JobsAvailable', () => {
26   beforeEach(() => {
27     (ApiActions.get as jest.Mock).mockResolvedValue({
28       status: 200,
29       data: mockJobs,
30     });
31   });
32
33   it('affiche les jobs disponibles après chargement', async () => {
34     const { getByText } = render(<JobsAvailable />);
35
36     await waitFor(() => {
37       expect(getByText('Développeur Frontend')).toBeTruthy();
38       expect(getByText('Développeur Backend')).toBeTruthy();
39     });
40   });
41
42   it('filtre les jobs par unité via le picker', async () => {
43     const { getByText, getByDisplayValue, queryByText } = render(
44       <JobsAvailable />,
45     );
46
47     await waitFor(() => {
48       expect(getByText('Développeur Frontend')).toBeTruthy();
49     });
50
51     const picker = getByDisplayValue('Toutes les Units');
52     fireEvent(picker, 'valueChange', 'Unit 1');
53
54     await waitFor(() => {
55       expect(getByText('Développeur Frontend')).toBeTruthy();
56       expect(queryByText('Développeur Backend')).toBeNull();
57     });
58   });
59
60   it('ouvre la modale en cliquant sur un job', async () => {
61     const { getByText, queryByText } = render(<JobsAvailable />);
62
63     await waitFor(() => {
64       expect(getByText('Développeur Frontend')).toBeTruthy();
65     });
66
67     fireEvent.press(getByText('Développeur Frontend'));
68
69     await waitFor(() => {
70       expect(queryByText(/Groupe/i)).toBeTruthy();
71     });
72   });
73 });
```

### Affichage des données après chargement

Ce test vérifie que les projets sont correctement affichés après que la requête API (mockée ici) a renvoyé les données. L'utilisation de `waitFor` permet de s'assurer que le rendu s'effectue après la fin du chargement.

### Filtrage dynamique via le picker

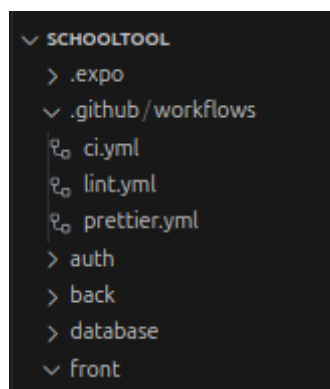
Ce test simule l'interaction avec le menu déroulant (Picker) permettant de filtrer les projets. En sélectionnant "Unit 1", le composant ne doit plus afficher les jobs associés à "Unit 2". Cela vérifie que la logique de filtrage est correctement implémentée côté client.

### Ouverture de la modale au click sur un projet

Ce test valide l'interaction utilisateur : lorsqu'on appuie sur un projet (Pressable), une modale d'information doit s'ouvrir. On vérifie sa présence à l'aide d'un mot-clé contenu dans la modale (Groupe).

## 3. Automatisation des tests

A la racine de notre projet, nous avons ajouté un dossier `.github`, dans lequel sont stockés nos fichiers permettant de paramétrer les github actions.



C'est notamment ici que l'on retrouve le fichier permettant de lancer le test de nos composants à chaque pull request effectuée.

```

name: CI/CD Pipeline

on:
  push:
    branches:
      - main
  pull_request:

jobs:
  build:
    runs-on: ubuntu-latest

    services:
      docker:
        image: docker:24.0.2
        options: --privileged

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 18

      - name: Install frontend dependencies
        run: |
          cd front
          npm install

      - name: Run frontend tests
        run: |
          cd front
          npm test -- --ci

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Build Docker Compose
        run: docker compose -f docker-compose.yml build

      - name: Send email notification
        uses: dawidd6/action-send-mail@v3
        with:
          server_address: smtp.gmail.com
          server_port: 465
          username: ${ secrets.EMAIL_USERNAME }
          password: ${ secrets.EMAIL_PASSWORD }
          subject: Nouvelle action sur le repo!
          body: "Une action a été déclenchée sur la branche main."
          to: ${ secrets.EMAIL_USERNAME }
          from: ${ secrets.EMAIL_USERNAME }
          secure: true

```



# CONCLUSION

## **Bilan personnel**

Le développement de l'application SchoolTool a constitué une expérience particulièrement formatrice, tant sur le plan technique que sur le plan humain. Ce projet m'a permis de renforcer mes compétences en architecture logicielle, gestion d'état, typage TypeScript, et intégration d'API sécurisées, tout en consolidant ma maîtrise de l'écosystème React Native / Expo.

J'ai également développé une réelle autonomie dans la résolution de problèmes complexes, la prise de décision technique, et l'organisation du travail en équipe. L'utilisation d'outils comme GitHub Projects, Docker, ou encore GitHub Actions m'a permis d'aborder des sujets proches des pratiques professionnelles actuelles (CI/CD, versioning, modularisation...). Je pense donc avoir couvert l'ensemble des compétences nécessaires à l'obtention du diplôme de Concepteur Développeur d'Applications.

## **Pertinence de l'application**

Comme précisé au début de ce rapport, ce projet est l'aboutissement de deux années et demi passées à la fois en tant qu'étudiant et sous-traitant de La Plateforme.

SchoolTool répond à un besoin réel : celui de centraliser la vie étudiante dans une application mobile fluide, moderne et accessible. Grâce à son design pensé pour la mobilité et ses fonctionnalités ciblées (gestion des projets, absences, emplois du temps via Google Calendar, etc.), l'application propose une expérience utilisateur cohérente et utile pour les étudiants.

Le projet démontre ainsi une réelle pertinence pédagogique, tant pour les utilisateurs finaux (étudiants, formateurs) que pour ses contributeurs techniques.

### **Pistes d'amélioration**

Afin de faire évoluer l'application vers un produit déployable à grande échelle, plusieurs améliorations peuvent être envisagées :

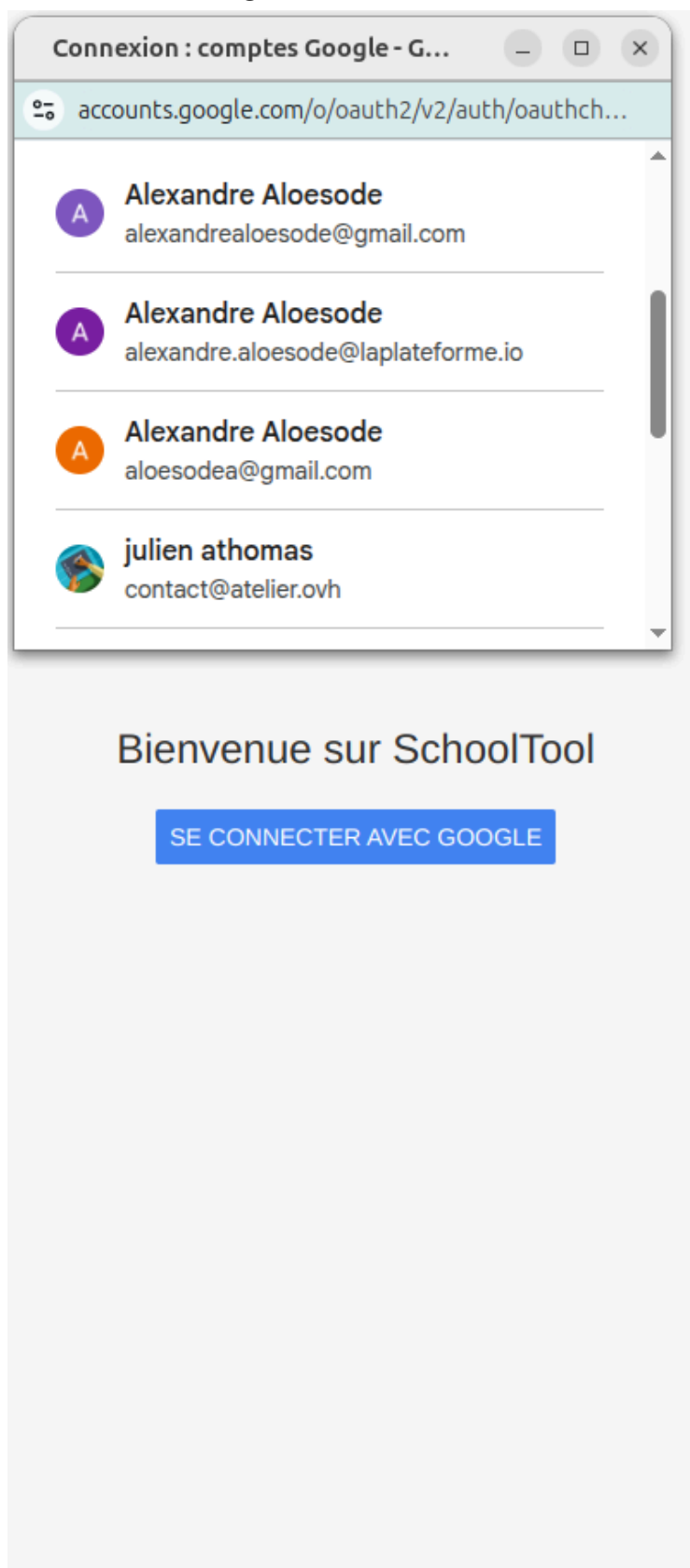
- Build final de l'application mobile, afin de générer les APK/IPA pour Android et iOS via Expo.
- Déploiement sur les stores (Google Play, App Store) avec signature, icônes et fiche produit.
- Thématisation adaptable, permettant de modifier la charte graphique selon l'école ou l'organisme d'accueil.
- Ajout d'un système de notifications push, pour renforcer l'engagement et améliorer la communication en temps réel.
- Optimisation des performances et tests plus poussés pour garantir stabilité et scalabilité.

Je tiens, pour conclure, à remercier mon collaborateur sur ce projet, Hervé BEZIAT, pour son apport notamment sur les parties maquettage, design et parcours utilisateur de l'application.

Je remercie également le CEO de l'Atelier, Julien ATHOMAS, pour ses conseils tout au long du développement, et enfin Terry CRISTINELLI, Directeur Informatique de l'école, pour ses encouragements dans la poursuite de ce projet.

# ANNEXES

## Page de connexion



## Index

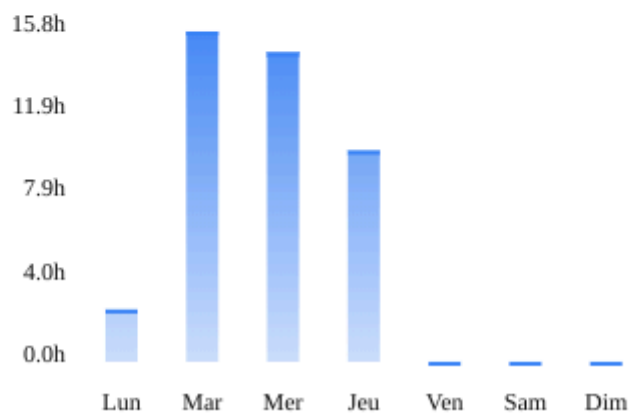


**Bienvenue  
Alexandre Aloesode**

## Temps de présence

Semaine du 17/03/2025

Total des heures : 43h



## Calendrier

Semaine du 28/07/2025

Mon 28/07

Daily  
09:15

Tue 29/07

Daily  
09:15

Wed 30/07

Daily  
09:15



## Projets en cours

# La Plateforme

**En cours**

**Disponibles**

**Terminés**

### Projets en cours

| Nom                    | Progression  |
|------------------------|--|
| RunTrack 1<br>- Jour 6 | <div><div></div></div> <a href="#">Retard de 6 jours</a>   |
| Sokoban                | <div><div></div></div> <a href="#">Retard de 10 jours</a>  |
| Snake                  | <div><div></div></div> <a href="#">Retard de 131 jours</a> |
| Projet libre -<br>cda  | <div><div></div></div> <a href="#">Retard de 210 jours</a> |



## Modale projets en cours



## Projets disponibles

# La Plateforme

En cours

Disponibles

Terminés

### Projets disponibles

Nom

Toutes les Units

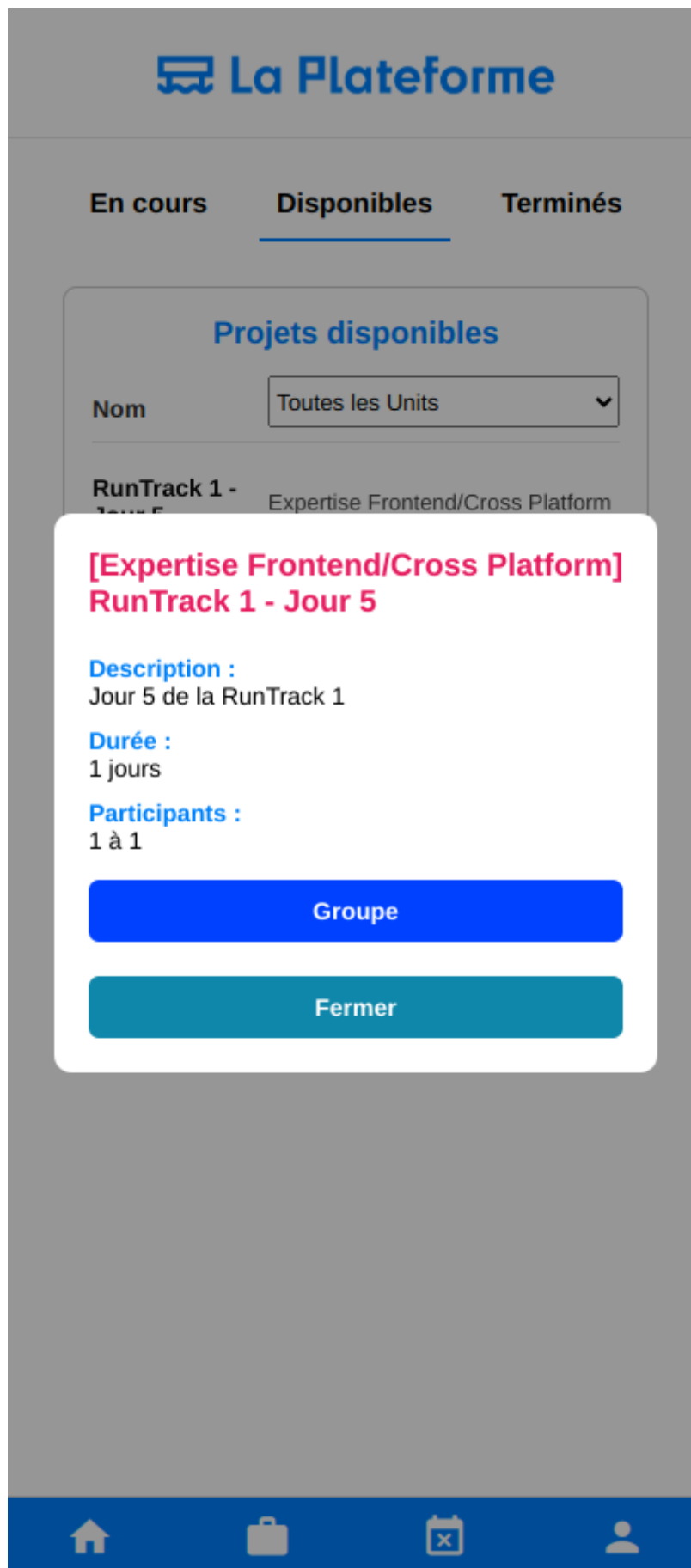


RunTrack 1 -  
Jour 5

Expertise Frontend/Cross Platform



## Modale projets disponibles





## Projets finis (avec click sur le filtre promotion)

### La Plateforme

En cours

Disponibles

Terminés

#### Projets finis

Promotion

Start 2022



Start 2022

Start Web

Marseille - B3 Web - 2023/24

Runtrack  
réseau - projet  
1

Starter

Runtrack php -  
jour 10

Starter

Runtrack php -  
jour 9

Starter

Runtrack php -  
jour 8

Starter

Runtrack php -  
jour 7

Starter

Runtrack php -  
jour 6

Starter

Runtrack php -  
jour 4&5

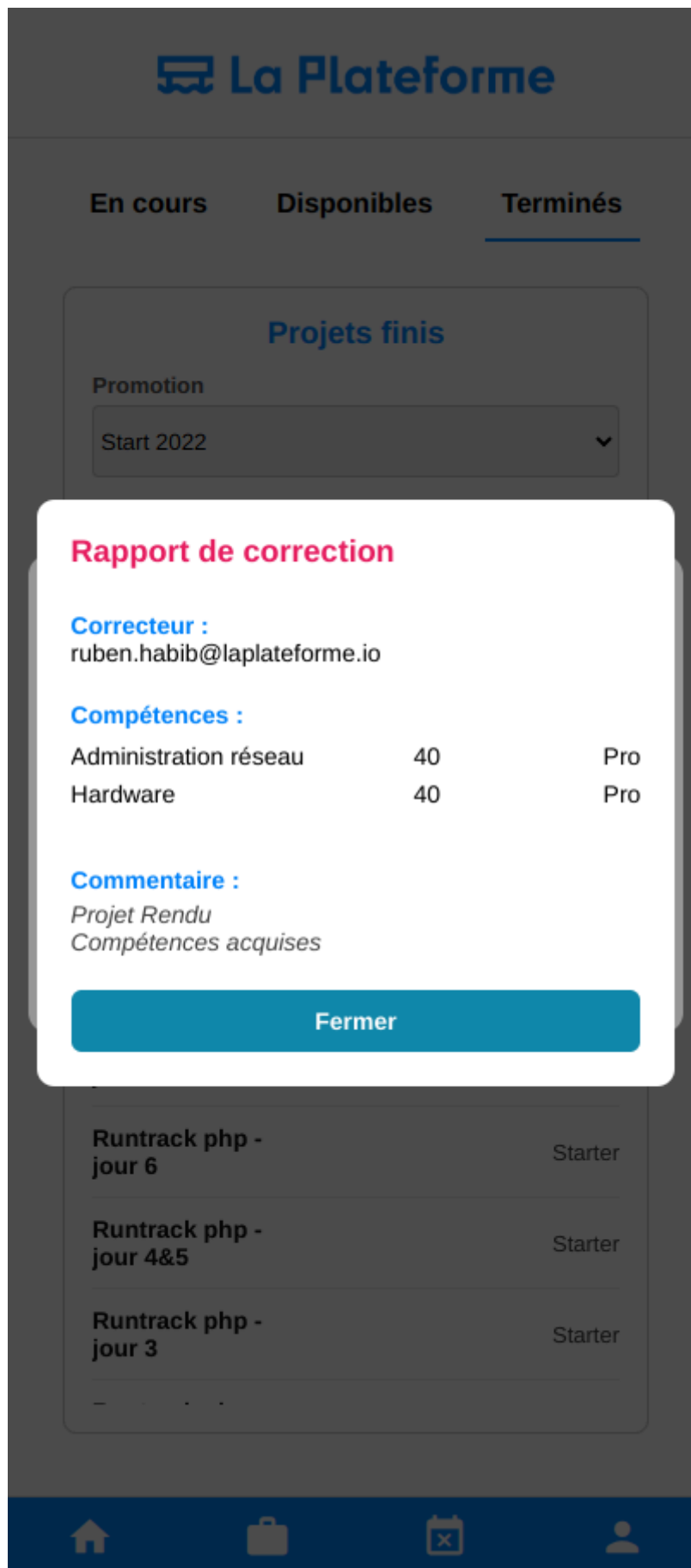
Starter

Runtrack php -  
jour 3

Starter



## Modale rapport de projet fini



## Consultation des absences




+ Nouvelle absence

### Absences précédentes

| Période                    | Durée   | Statut     |
|----------------------------|---------|------------|
| 26/08/2024 -<br>29/08/2024 | 3 jours | En attente |
| 02/09/2024 -<br>06/09/2024 | 4 jours | En attente |
| 04/12/2024 -<br>12/12/2024 | 6 jours | En attente |




## Modale de demande d'absence




### Nouvelle absence


**Date de début**







**Date de retour**



**Motif**

 Joindre un justificatif



## Compétences acquises



Profil

Compétences

### Compétences acquises

Anglais

100.0%

Cyber Sécurité

100.0%

Dev.

83.5%

Dev. Web

80.5%

Expression E&O

100.0%

Outils

88.0%


Pro

82.1%

Soft Skills



## Modale détails de la compétence



Profil

Compétences

### Compétences acquises

Anglais

100.0%

Cyber Sécurité

100.0%

**Compétences [Cyber Sécurité]**

| Nom           | Niveau | Acquis | En cours |
|---------------|--------|--------|----------|
| Network       | Pro    | 32     | 0        |
| Packet Tracer | Pro    | 32     | 0        |

Fermer

Expression E&O

100.0%

Outils

88.0%

Pro

82.1%

Soft Skills

