



DOSSIER PROFESSIONNEL (DP)

Nom de naissance ▶ **ROMDHANI**
Nom d'usage ▶
Prénom ▶ **MEHDI**
Adresse ▶ **33 rue borde, 13008 Marseille**

Titre professionnel visé

RNCP37873 CONCEPTEUR DÉVELOPPEUR D'APPLICATIONS

MODALITÉ D'ACCÈS :

- Parcours de formation
- Validation des Acquis de l'Expérience (VAE)

DOSSIER PROFESSIONNEL (DP)

Présentation du dossier

Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel.
Ce titre est délivré par le Ministère chargé de l'emploi.

Le DP appartient au candidat. Il le conserve, l'actualise durant son parcours et le présente **obligatoirement à chaque session d'examen.**

Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE. Il est consulté par le jury au moment de la session d'examen.

Pour prendre sa décision, le jury dispose :

1. des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l'entretien professionnel ou de l'entretien technique ou du questionnement à partir de productions.
2. du **Dossier Professionnel (DP)** dans lequel le candidat a consigné les preuves de sa pratique professionnelle.
3. des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d'un parcours de formation
4. de l'entretien final (dans le cadre de la session titre).

[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels du ministère chargé de l'Emploi]

Ce dossier comporte :

- pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ;
- un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d'un titre, d'un diplôme, d'un certificat de qualification professionnelle (CQP) ou des attestations de formation ;
- une déclaration sur l'honneur à compléter et à signer ;
- des documents illustrant la pratique professionnelle du candidat (facultatif)
- des annexes, si nécessaire.

DOSSIER PROFESSIONNEL (DP)

Sommaire

Développer une application sécurisée	p. 5
▶ Exemple n°1: Projet SafeBase	p.
▶ Exemple n°2 : Projet Flyradar	p. p. p.
Concevoir et développer une application sécurisée organisée en couches	p.
▶ Exemple n°1 : Projet YouthWallet	p. p.
▶ Exemple n°2: Projet Flyradar	p. p. p. p.
Préparer le déploiement d'une application sécurisée	p.
▶ Exemple n°1 : Projet YouthWallet	p. p.
▶ Exemple n°2 : Projet SafeBase	p. p. p.
Titres, diplômes, CQP, attestations de formation (<i>facultatif</i>)	p.
Déclaration sur l'honneur	p.
Documents illustrant la pratique professionnelle (<i>facultatif</i>)	p.
Annexes (<i>Si le RC le prévoit</i>)	p.

DOSSIER PROFESSIONNEL ^(DP)

**EXEMPLES DE PRATIQUE
PROFESSIONNELLE**

DOSSIER PROFESSIONNEL (DP)

Activité-type 1 Développer une application sécurisée

Exemple n°1 - SAFEBASE

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

INTRODUCTION :

SafeSpace est une application web permettant la gestion sécurisée de sauvegardes et de restaurations de fichiers dans une logique orientée DevOps. Ce projet a été pensé pour centraliser les opérations critiques d'administration (sauvegarde de bases de données, suivi des statuts, notifications, etc.) dans un tableau de bord intuitif et sécurisé.

L'objectif principal était d'intégrer un système d'authentification basé sur des **tokens JWT**, une architecture modulaire avec des **conteneurs Docker**, et un système de **CI/CD** pour simuler un contexte de production.

Le projet met fortement l'accent sur la **sécurité, l'organisation du code et le respect des bonnes pratiques DevOps**. Il a été développé en groupe dans un temps imparti, selon une méthode Agile.

Compétence 1.1 – Installer et configurer son environnement de travail

Dans le cadre du projet **SafeSpace**, j'ai mis en place un **environnement de développement conteneurisé** en utilisant **Docker** et **Docker Compose**

DOSSIER PROFESSIONNEL (DP)

```
services:  
  db:  
    image: postgres:16  
    container_name: postgres_container  
    restart: always  
    environment:  
      POSTGRES_PASSWORD: mysecretpassword  
    ports:  
      - "5432:5432"  
    volumes:  
      - postgres_data:/var/lib/postgresql/data  
  
  pgadmin:  
    image: dpage/pgadmin4  
    container_name: pgadmin_container  
    restart: always  
    environment:  
      PGADMIN_DEFAULT_EMAIL: admin@example.com  
      PGADMIN_DEFAULT_PASSWORD: adminpassword  
    ports:  
      - "8080:80"  
    depends_on:  
      - db
```

```
backend:  
  build:  
    context: ./backend  
    container_name: backend_container  
    restart: always  
  ports:  
    - "3000:3000"  
  depends_on:  
    - db  
  environment:  
    - DB_HOST=db  
    - DB_PORT=5432  
    - DB_USER=postgres  
    - DB_PASSWORD=mysecretpassword  
    - DB_NAME=postgres  
  volumes:  
    - backend_src:/backend  
    - backend_backups:/backend/backups  
  healthcheck:  
    test: ["CMD", "curl", "-f", "http://localhost:3000"]  
    interval: 30s  
    timeout: 10s  
    retries: 5
```

- DOCKERCOMPOSE .YAML

DOSSIER PROFESSIONNEL (DP)

Le projet comprend un **frontend React isolé** avec son propre **Dockerfile**, configuré pour tourner avec **Vite**.

```
FROM node:16

WORKDIR /frontend

COPY package*.json .

RUN npm install

COPY .

RUN npm run build

EXPOSE 5173

CMD ["npm", "run", "dev", "--", "--host", "0.0.0.0"]
```

- DOCKERFILE FRONT

J'ai initialisé un dépôt Git et mis en place une organisation en **GitFlow**, permettant un travail structuré en branches (**main**, **develop**, **feature/***, etc.).

DOSSIER PROFESSIONNEL (DP)

```
...  
ci/cd  
dev  
* feature/DEBUG  
feature/TU  
feature/TUCICD  
feature/connection-database  
feature/docker-end-project  
feature/front-restore  
feature/setup-backend  
feature/setup-cron  
feature/setup-css  
feature/setup-docker  
feature/setup-dump  
feature/setup-frontend  
feature/setup-remove  
feature/setup-rm-restore  
main  
test/github-actions
```

- FLOW GIT

Pour garantir la qualité du code, j'ai intégré des outils comme **ESLint** et **Prettier**, configurés dans le projet, et j'ai ajouté un **.editorconfig** pour harmoniser les pratiques.

Le projet peut être lancé de manière fiable et reproductible avec une simple commande :

```
...  
docker-compose up --build
```

Cela initialise automatiquement tous les services nécessaires.

Ce setup permet d'assurer un **environnement cohérent** pour le développement, les tests, et le déploiement.

DOSSIER PROFESSIONNEL (DP)

Compétence 1.2 – Développer des interfaces utilisateur

J'ai développé une interface utilisateur avec **React** et **Vite**, intégrant des composants tels que des écrans de connexion, confirmation de logout, et des layouts conditionnels en fonction de l'état de connexion.

 **Connexion à la Base de Données**

Configurez votre connexion à la base de données PostgreSQL ici.

Hôte
db

Port
5432

Base de données
SafeBase

Utilisateur
Nom d'utilisateur

Mot de passe
Mot de passe 

 **Se connecter**

- Connexion Screen

Tableau de Bord SafeBase

Bienvenue sur votre centre de contrôle pour la gestion de bases de données et sauvegardes.

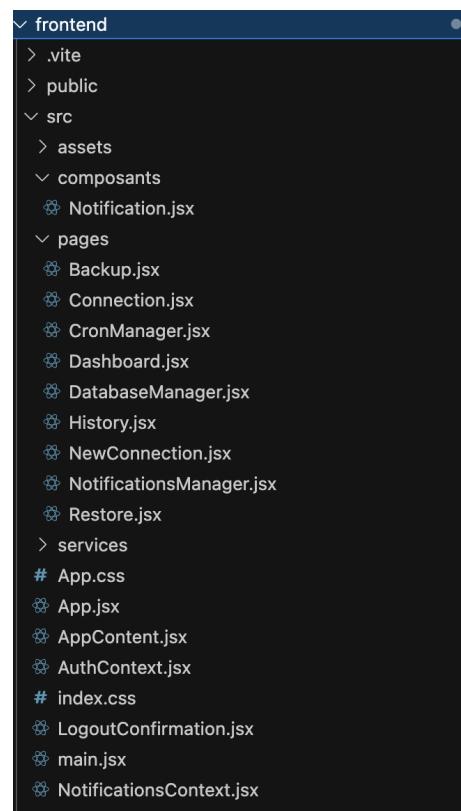
 Sauvegardes Gérez et planifiez vos sauvegardes de bases de données en toute simplicité.	 Restaurations Restaurez rapidement vos bases de données à partir de sauvegardes précédentes.	 Tâches Cron Automatissez vos opérations de sauvegarde et de maintenance.	 Aperçu Rapide Visualisez l'état de vos bases de données et sauvegardes en un coup d'œil.
--	---	---	---

Résumé de l'Activité

Dernière sauvegarde :	Il y a 2 heures
Prochaine sauvegarde planifiée :	Dans 4 heures
Nombre total de bases de données :	12
Espace de stockage utilisé :	45 GB

- Dashboard Screen

DOSSIER PROFESSIONNEL (DP)



- Architecture du Projet

L'interface respecte une charte graphique simple, est responsive, et utilise les bonnes pratiques d'accessibilité (navigation clavier, ARIA). Cela aussi aurait pu être prototype via FIGMA mais vu les délais imposés dans le projet cela n'a pas pu se faire.

Le code de l'interface a été organisé de manière **modulaire**, notamment grâce à un composant **AuthContext** qui centralise la **gestion de session** de l'utilisateur.

Ce contexte permet d'**adapter dynamiquement le rendu des composants UI** (ex. : boutons "Connexion" ou "Déconnexion", pages protégées, redirections).

Le statut de connexion est persisté dans le **localStorage**, ce qui permet à l'application de conserver l'état entre les sessions.

DOSSIER PROFESSIONNEL (DP)

```
import React, { createContext, useState, useEffect } from 'react';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  useEffect(() => {
    const loggedInStatus = localStorage.getItem('isLoggedIn') === 'true';
    setIsLoggedIn(loggedInStatus);
  }, []);

  const login = () => {
    localStorage.setItem('isLoggedIn', 'true');
    setIsLoggedIn(true);
  };

  const logout = () => {
    localStorage.setItem('isLoggedIn', 'false');
    setIsLoggedIn(false);
  };

  return (
    <AuthContext.Provider value={{ isLoggedIn, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

- **AuthContext**

En Conclusion :

*Pour SafeSpace, j'ai conçu une interface utilisateur avec React, organisée de manière modulaire. La gestion de l'état de connexion est centralisée via un **AuthContext**, ce qui permet d'adapter dynamiquement l'affichage de l'interface selon que l'utilisateur est authentifié ou non, tout en respectant les principes de sécurité, de réactivité et de simplicité d'utilisation.*

DOSSIER PROFESSIONNEL (DP)

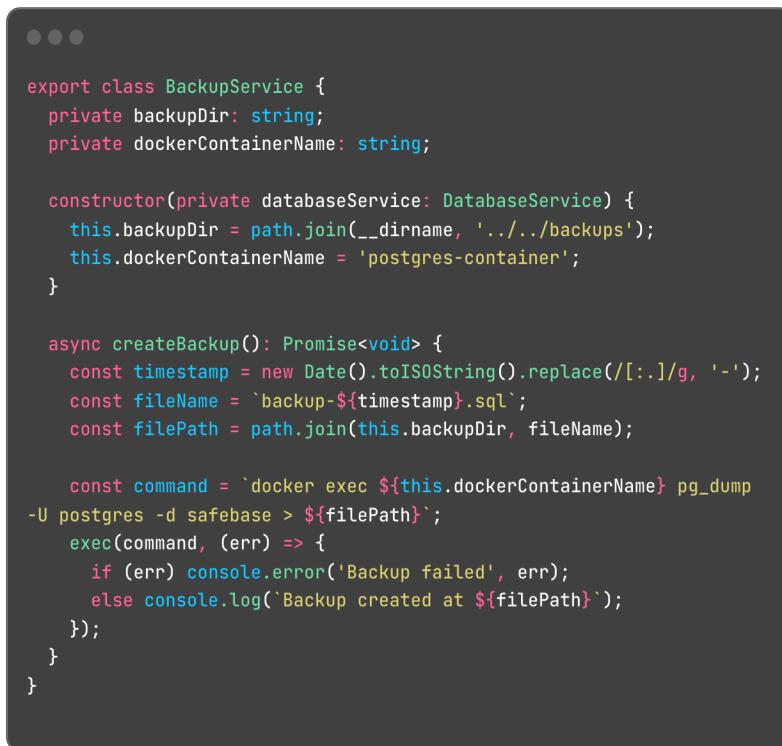
Compétence 1.3 – Développer des composants métier

Dans le projet **Safebase**, j'ai conçu et développé des **composants métier essentiels** en TypeScript, sans framework haut-niveau, en m'appuyant sur Fastify, une base PostgreSQL et une architecture modulaire orientée services.

Parmi les composants développés :

Exemples de composants métier :

- **BackupService.ts** : service de sauvegarde automatique de la base de données via des commandes Docker. Il crée des fichiers de dump horodatés, compressés et chiffrés, stockés dans un répertoire dédié. (voir capture)



```
export class BackupService {
    private backupDir: string;
    private dockerContainerName: string;

    constructor(private databaseService: DatabaseService) {
        this.backupDir = path.join(__dirname, '../../backups');
        this.dockerContainerName = 'postgres-container';
    }

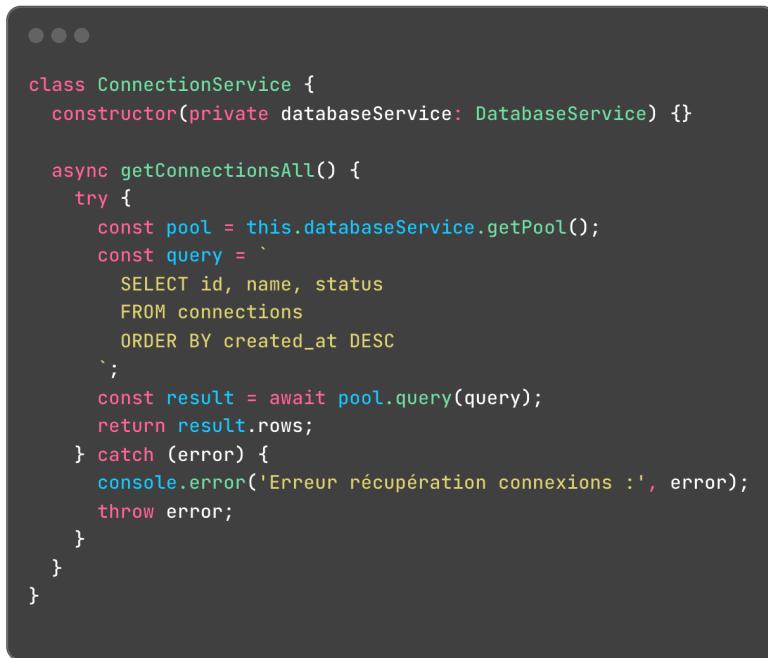
    async createBackup(): Promise<void> {
        const timestamp = new Date().toISOString().replace(/[^.]/g, '-');
        const fileName = `backup-${timestamp}.sql`;
        const filePath = path.join(this.backupDir, fileName);

        const command = `docker exec ${this.dockerContainerName} pg_dump
-U postgres -d safebase > ${filePath}`;
        exec(command, (err) => {
            if (err) console.error('Backup failed', err);
            else console.log(`Backup created at ${filePath}`);
        });
    }
}
```

- Screenshot

DOSSIER PROFESSIONNEL (DP)

- **connectionServiceList.ts** : j'y ai implémenté la logique qui permet de charger dynamiquement la liste des services disponibles, avec vérification de leur état de connexion. Cela permet à l'application de s'adapter automatiquement selon l'environnement ou les dépendances actives.



```
class ConnectionService {
    constructor(private databaseService: DatabaseService) {}

    async getConnectionsAll() {
        try {
            const pool = this.databaseService.getPool();
            const query = `
                SELECT id, name, status
                FROM connections
                ORDER BY created_at DESC
            `;
            const result = await pool.query(query);
            return result.rows;
        } catch (error) {
            console.error('Erreur récupération connexions :', error);
            throw error;
        }
    }
}
```

- SCRENSHOT

- **cronService.ts** : ce service exécute automatiquement des tâches comme les sauvegardes récurrentes. Il repose sur la librairie **node-cron** et inclut des vérifications avant exécution pour éviter les doublons ou erreurs critiques.

DOSSIER PROFESSIONNEL (DP)

```
  export class CronService {
    private cronJobs: Map<string, ScheduledTask>;
    constructor(private io: SocketIOServer) {
      this.cronJobs = new Map();
    }
    startCronJob(jobName: string, schedule: string, callback: () =>
void): void {
      if (this.cronJobs.has(jobName)) {
        console.warn(`Job "${jobName}" already exists.`);
        return;
      }
      const task = cron.schedule(schedule, callback);
      this.cronJobs.set(jobName, task);
      console.log(`Cron job "${jobName}" started with schedule
"${schedule}"`);
    }
  }
```

database.service.ts : j'ai mis en place une couche d'abstraction pour gérer SQLite, avec des requêtes préparées, un système de migrations, et une gestion centralisée des erreurs.

DOSSIER PROFESSIONNEL (DP)

```
export class DatabaseService {
  private pool: Pool | null = null;

  initializePool(config: PoolConfig): void {
    this.pool = new Pool(config);
  }

  async connect(config: PoolConfig): Promise<void> {
    if (!this.pool) this.initializePool(config);
    try {
      const client = await this.pool!.connect();
      console.log('Connected to database');
      client.release();
    } catch (error) {
      console.error('Connection error:', error);
    }
  }

  getPool(): Pool {
    if (!this.pool) throw new Error('Pool not initialized');
    return this.pool;
  }
}
```

En Conclusion :

Tous les composants sont développés dans un style défensif : chaque entrée est vérifiée, les erreurs sont capturées proprement, et les modules sont isolés pour être facilement testables. Le code est intégralement typé et documenté. Cette approche garantit la robustesse et la maintenabilité du système.

Ce travail illustre ma capacité à coder une logique métier complète, sécurisée, et conforme aux bonnes pratiques de développement orienté objet, sans framework lourd, avec un souci constant de fiabilité et de clarté.

DOSSIER PROFESSIONNEL (DP)

Compétence 1.4 – Contribuer à la gestion d'un projet informatique

Méthodologie : Agile Kanban – Outils : Trello, Git,

- Pour ce projet, j'ai utilisé une organisation basique mais efficace. J'ai structuré mes tâches avec un tableau **Kanban** (voir capture ci-jointe), ce qui m'a permis de visualiser les étapes à faire, en cours et terminées.
- Je n'ai pas utilisé d'outil complexe de gestion de projet, mais j'ai quand même pris le temps de **découper mon travail** par fonctionnalités (connexion, sauvegarde, affichage de données, cron, etc.) et de suivre l'avancement à mon rythme.
- J'ai aussi utilisé **Git** pour versionner mon code et faire des sauvegardes régulières. Chaque bloc de code était testé rapidement avant d'être considéré comme terminé.
- Le but était d'avoir un minimum de suivi pour ne pas me perdre et pouvoir avancer étape par étape jusqu'à une version fonctionnelle.

À faire	En cours	Terminé
- Export PDF - Notifications	- Connexion base - Cron automatique	- Sauvegarde BDD - Affichage connexions

DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

Environnement technique utilisé pour le projet

Pour le bon fonctionnement de l'application, j'ai mis en place une infrastructure basée sur Docker avec plusieurs services. Voici les principales technologies utilisées :

- PostgreSQL 16 pour la base de données.
- pgAdmin 4 pour l'administration de la base via une interface web.
- Node.js pour le backend (avec TypeScript).
- Vite pour le frontend (avec React).
- Docker et Docker Compose pour lancer et gérer les conteneurs.
- Volumes Docker pour sauvegarder les données et le code.
- Variables d'environnement pour sécuriser les connexions (base de données, ports...).
- Healthcheck Docker pour s'assurer que le backend est bien lancé avant les autres services.

Chaque service (frontend, backend, base de données) est isolé dans un conteneur dédié et communique avec les autres via un réseau interne géré par Docker

3. Avec qui avez-vous travaillé ?

Romain Chazaut

Maëlle LaGARDE

4. Contexte

Nom de l'entreprise, organisme ou association ▶ **La Plateforme**

Chantier, atelier, service ▶ **SAFEBASE**

Période d'exercice ▶ Du : **03/09/2024** au : **18/09/2024**

DOSSIER PROFESSIONNEL^(DP)

5. Informations complémentaires (facultatif)

Sur un autre projet personnel plus simple (API Express + Vite), j'ai géré les tâches avec un tableau Notion très basique.

Sans méthode formelle, j'ai juste organisé les étapes à suivre pour rester structuré et avancer efficacement en autonomie.

DOSSIER PROFESSIONNEL (DP)

Activité-type 1 | Développer une application sécurisée

Exemple n° 2 - Projet Flyradar

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Compétence 1.1 – Installer et configurer l'environnement de développement

Présentation rapide du projet

Gliradar est une application mobile créée avec Expo (React Native), qui affiche des données en temps réel issues d'un backend **NestJS** relié à une base MongoDB pour le tracking de FLY.

Tout le projet est organisé avec Docker, et l'environnement est automatisé grâce à un Makefile

Ce que j'ai mis en place pour cette compétence :

- Installation de tous les outils : Node.js, Expo CLI, NestJS CLI, Docker, MongoDB, Make, etc.
- Configuration d'un fichier **docker-compose.yml** pour lancer MongoDB et le backend dans des conteneurs séparés.
- Création d'un Dockerfile pour le backend NestJS.
- Utilisation d'un Makefile pour automatiser les commandes (**make dev**, **make install**, etc.).
- Mise en place de fichiers **.env** pour centraliser les variables (ports, URI Mongo, etc.).
- Lancement et test de tout l'environnement avec **make dev** et **docker compose up**, puis vérification que l'app mobile récupère bien les données depuis l'API.

Annexes : Voir ci-dessous

DOSSIER PROFESSIONNEL (DP)

```
1 flightApiRadar git:(feature/FEAT-015) ✘ make up
2 docker-compose up -d
3 [+] Running 2/4
4   ⚡ Container mongodb           Waiting
5   ⚡ Container redis             Waiting
6   ⚡ Container flightapiradar-app-1   Created
7   ⚡ Container flightapiradar-frontend-1  Created
8   ⚡ Container flightapiradar-frontend-1  Created
0.0s
0.0s
```

- MAKEFILE

```
1 app:
2   build:
3     context: ./flightradar24-backend
4     dockerfile: Dockerfile
5     target: dev
6   ports:
7     - "3000:3000"
8   volumes:
9     - ./flightradar24-backend:/usr/src/app
10  env_file:
11    - ./flightradar24-backend/.env.docker
12  environment:
13    - NODE_ENV=development
14    - MONGODB_URI=mongodb://root:password@mongodb:27017/admin?
authSource=admin
15    - REDIS_HOST=redis
16    - REDIS_PORT=6379
17
```

- DOCKER COMPOSE

DOSSIER PROFESSIONNEL (DP)

```
1 FROM node:18 AS dev
2
3 WORKDIR /usr/src/app
4
5 # Copier fichiers package
6 COPY package*.json ./
7
8 # Installer dépendances
9 RUN npm install
10
11 # Copier le reste du code (y compris docker-entrypoint.sh)
12 COPY . .
13
14 # S'assurer que le script a les permissions d'exécution
15 RUN chmod +x docker-entrypoint.sh
16
17 EXPOSE 3001
18
19 # Utiliser le script comme point d'entrée
20 CMD ["sh", "./docker-entrypoint.sh"]
```

- DOCKERFILE BACKEND

En Conclusion :

Grâce à cette configuration, j'ai pu travailler dans un environnement stable, reproductible et bien organisé. Le backend, la base de données et le frontend communiquent correctement, ce qui m'a permis de me concentrer ensuite sur le développement des interfaces et des fonctionnalités.

DOSSIER PROFESSIONNEL^(DP)

Compétence 1.2 – Développer des interfaces utilisateur

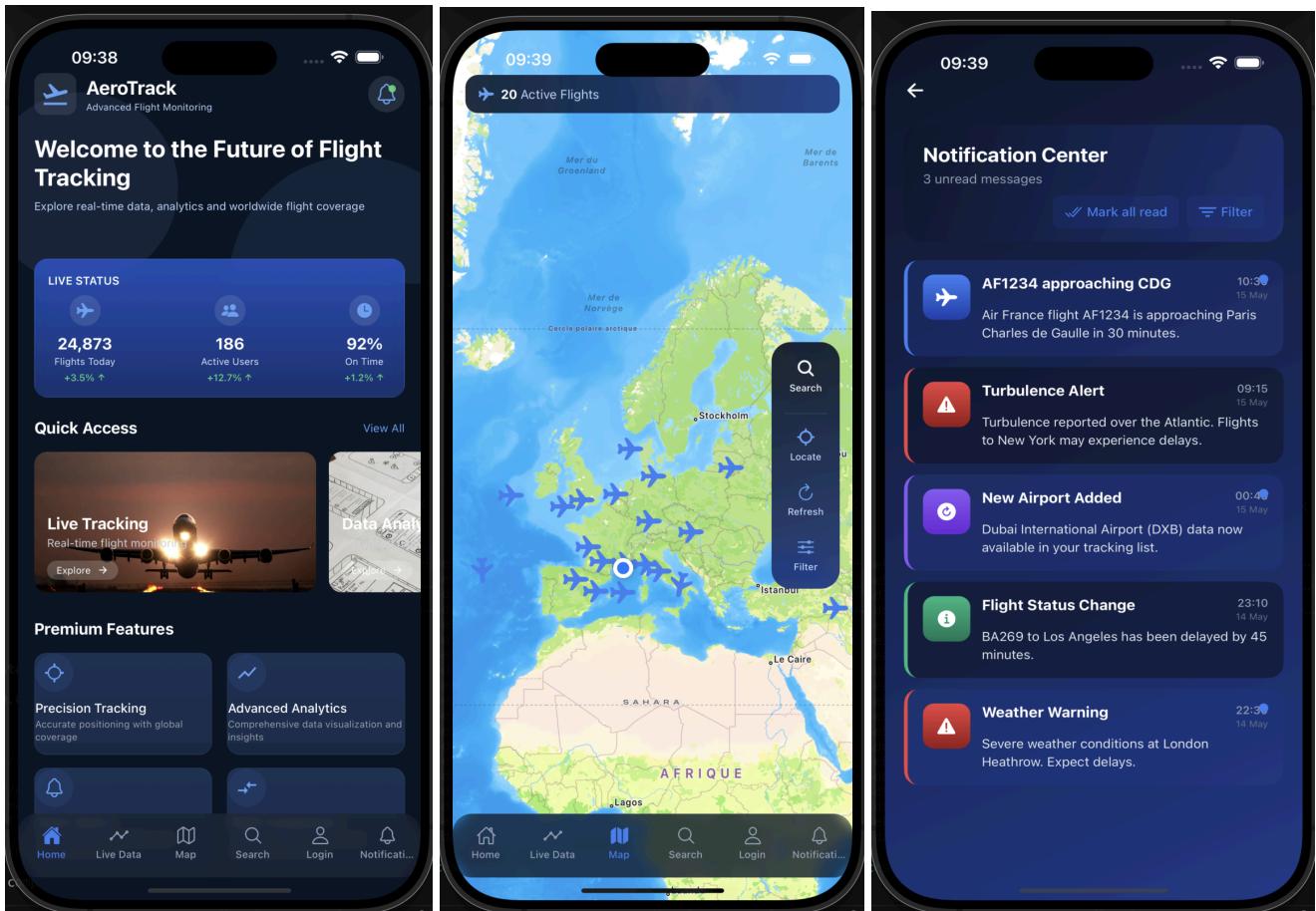
J'ai conçu plusieurs écrans dans l'application mobile, développée avec Expo / React Native, en suivant une logique modulaire et responsive.

- J'ai créé des composants réutilisables avec Zustand pour la gestion d'état local, comme pour les vols suivis (starred flights).
- L'interface est stylisée avec NativeWind, ce qui permet un design responsive et cohérent.
- La navigation est gérée avec expo-router en utilisant une barre d'onglets personnalisée (_layout.tsx) avec indicateurs dynamiques (comme les badges de notifications).
- Chaque écran a un rôle bien défini :
- **homeScreen.tsx** : dashboard principal avec accès rapide et stats animées.
- **liveScreen.tsx, mapsFlights.tsx, searchScreen.tsx** : affichage temps réel, carte, recherche.
- **NotificationsScreen.tsx** : affichage dynamique des vols suivis avec système de validation automatique.
- **LoginScreen.tsx** : page de connexion futuriste avec API GraphQL et animation d'entrée.

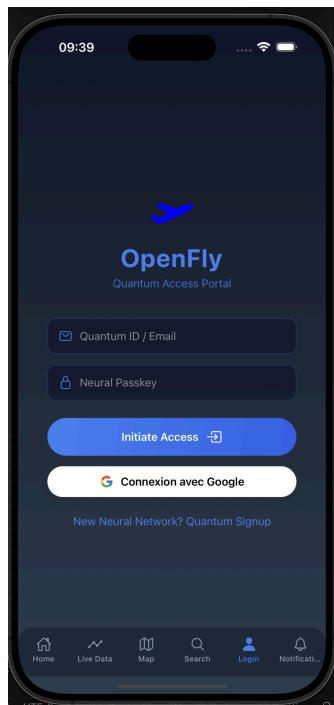
ANNEXES : Voir ci-dessous :

- **Dashboard (homeScreen)**
- **Carte ou suivi (mapsFlights)**
- **Notifications dynamiques**
- **Écran de connexion (formulaire avec effet animé)**

DOSSIER PROFESSIONNEL (DP)



DOSSIER PROFESSIONNEL (DP)



En Conclusion :

L'interface est fluide, cohérente et pensée pour une bonne expérience utilisateur, tout en restant bien structurée et réutilisable. J'ai veillé à séparer les responsabilités dans les fichiers pour un meilleur maintien du code.

////

Compétence 1.3 – Développer une application sécurisée

Pour sécuriser l'application Gliradar, j'ai mis en place un système d'authentification complet côté backend avec NestJS :

- **Mise en place d'une authentification par JWT, protégée avec des guards (jwt-auth.guard.ts).**
- **Intégration de la connexion avec Google OAuth via un guard dédié (google-oauth.guard.ts).**
- **Hash des mots de passe avec bcrypt et création de tokens via le service JWT.**
- **Sécurisation des routes sensibles à l'aide de @UseGuards() dans les controllers.**

DOSSIER PROFESSIONNEL (DP)

Le tout est testé via Postman et connecté à l'interface mobile.

Annexes :

- **Exemple de route protégée dans le contrôleur**

```
 1  @UseGuards(GoogleAuthGuard) // Redirection vers url de google
 2  @Get('google/login')
 3  async googleLogin() {}
 4
 5  @UseGuards(GoogleAuthGuard)
 6  @Get('google/callback')
 7  async googleCallback(@Req() req, @Res() res) {
 8    try {
 9      console.log('Google Callback:', req.user);
10
11      if (!req.user) {
12        return res.redirect(
13          `exp://192.168.1.40:8081/---callback?
error=authentication_failed`,
14        );
15    }
}
```

DOSSIER PROFESSIONNEL (DP)

```
1  @Injectable()
2  export class JwtStrategy extends PassportStrategy(Strategy) {
3    constructor(private usersService: UsersService) {
4      console.log('JWT_SECRET in JwtStrategy:',
5      process.env.JWT_SECRET);
6
7      super({
8        jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
9        ignoreExpiration: false,
10       secretOrKey: process.env.JWT_SECRET || 'secret', // Make
11       sure this matches your JwtModule
12     });
13    }
14
15    async validate(payload: any) {
16      const user = await this.usersService.findById(payload.sub);
17      if (!user) {
18        throw new UnauthorizedException();
19      }
20      return user;
21    }
22  }
```

- **JWT**

- **Test d'appel API avec token valide/expiré**

DOSSIER PROFESSIONNEL (DP)

```
● ● ●

1  @UseGuards(GoogleAuthGuard) // Redirection vers url de google
2  @Get('google/login')
3  async googleLogin() {}
4
5  @UseGuards(GoogleAuthGuard)
6  @Get('google/callback')
7  async googleCallback(@Req() req, @Res() res) {
8    try {
9      console.log('Google Callback:', req.user);
10
11     if (!req.user) {
12       return res.redirect(
13         `exp://192.168.1.40:8081/---/callback?
14         error=authentication_failed`;
15     }
16   }
```

```
[LOG] GraphQL Response: {"data": {"login": {"__typename": "AuthResponse", "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmtZSI6Ikx1aUBsdWkuZnIiLCJzdWIi0iI20DdhMDgxN2E4NDkzM2FiMDU4ZWFLYzkilCJyb2xIjoiVVNFUiIsImhdCI6MTc1MjgyNzk0OCwiZXhwIjoxNzUyODI40DQ4fQ.FDEeqWIQnohEhLM4RwCKfiZ8qhNUZEW7RVDYScJoX0"}, "errors": undefined, "extensions": undefined}
[LOG] 🌐 Zustand - setToken appelé avec : eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmtZSI6Ikx1aUBsdWkuZnIiLCJzdWIi0iI20DdhMDgxN2E4NDkzM2FiMDU4ZWFLYzkilCJyb2xIjoiVVNFUiIsImhdCI6MTc1MjgyNzk0OCwiZXhwIjoxNzUyODI40DQ4fQ.FDEeqWIQnohEhLM4RwCKfiZ8qhNUZEW7RVDYScJoX0
```

NOTE : Côté frontend, l'accès aux pages est contrôlé selon l'état de connexion. Les tokens sont stockés côté client et utilisés dans les appels API sécurisés.

En Conclusion :

Ce système d'authentification me permet de sécuriser l'accès aux données sensibles, tout en offrant une bonne expérience utilisateur avec la connexion Google. La structure est claire, réutilisable, et prête à évoluer si besoin.

DOSSIER PROFESSIONNEL (DP)

Compétence 1.4 – Contribuer à la gestion d'un projet informatique

Pour le projet Gliradar, j'ai travaillé en méthode Agile Scrum, avec des daily meetings chaque matin pour faire le point sur l'avancement.

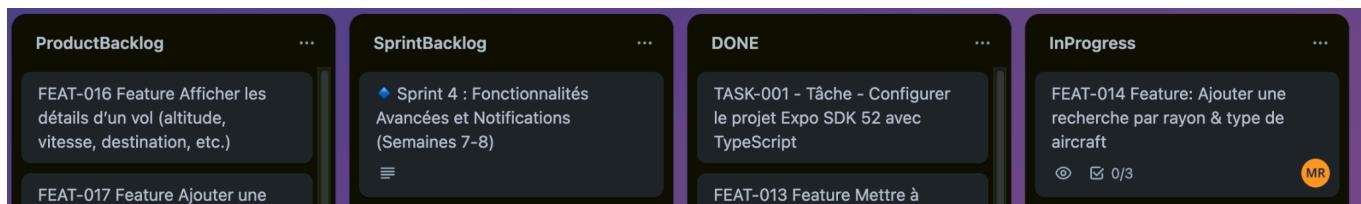
J'ai utilisé Trello pour suivre les tâches et organiser le travail par sprint.

Le projet était en collaboration avec un tech lead, qui validait mes pull requests. Une petite pipeline CI/CD était en place, et m'envoyait des mails à chaque action Git importante (push, merge, etc.).

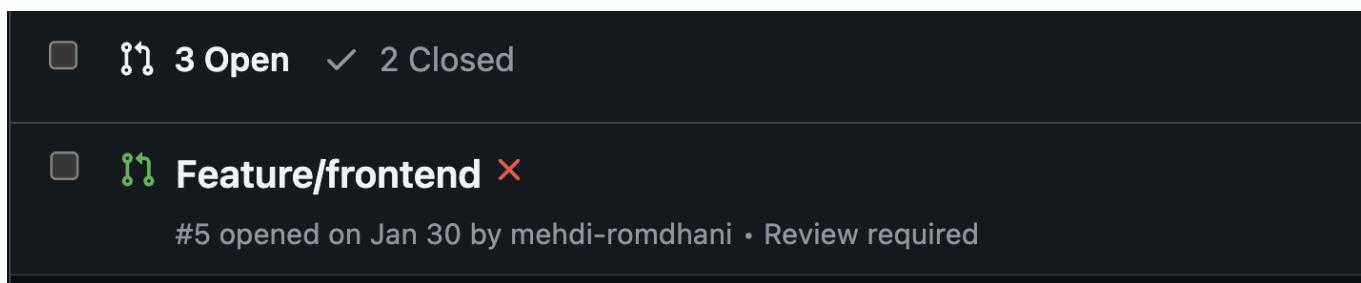
Un fichier README bien structuré permettait aussi de suivre l'installation, les scripts utiles et l'architecture du projet.

ANNEXES : Voir ci-dessous :

- Trello (scrum/agile)



- PR git



- Pipeline CI/CD

DOSSIER PROFESSIONNEL (DP)

```
1     - name: Send Gmail Notification
2       uses: dawidd6/action-send-mail@v2
3       with:
4         server_address: smtp.gmail.com
5         server_port: 465
6         username: ${secrets.EMAIL_USER}
7         password: ${secrets.EMAIL_PASS}
8         subject: Nouvelle Pull Request ouverte
9         body: Une nouvelle pull request a été ouverte et le
10            linting a été effectué.
11           to: mehdi.romdhani1@gmail.com
12           from: GitHub Actions <${secrets.EMAIL_USER}>
```

- Aperçu du README du projet

```
1      # Application Mobile - Projet d'Alternance
2
3  ## 📄 Description du Projet
4
5 Ce projet est une application mobile développée dans le cadre de
mon alternance. Elle utilise **React Native avec Expo** pour le
frontend et **NestJS** pour le backend. L'objectif est de créer
une application performante, évolutive et bien structurée, avec un
backend entièrement dockerisé.
6
7 ---
```

DOSSIER PROFESSIONNEL (DP)

En Conclusion :

Ce mode de gestion m'a permis d'avoir une bonne visibilité sur les tâches à faire, de garder un rythme de travail régulier, et de collaborer efficacement avec le tech lead. La CI/CD légère et les retours quotidiens m'ont aidé à améliorer la qualité du code tout au long du projet

2. Précisez les moyens utilisés :

Technologies et outils utilisés

- **Frontend** : Expo (React Native), Zustand, NativeWind, Expo Router
- **Backend** : NestJS (TypeScript), JWT, OAuth (Google), WebSockets
- **Base de données** : MongoDB
- **Outils de gestion** : Trello, GitHub (PR), Daily Scrum
- **CI/CD** : Notifications Git, pipeline de déploiement simple
- **Docker** : Docker Compose, Dockerfile
- **Automatisation** : Makefile
- **Docs** : README structuré

3. Avec qui avez-vous travaillé ?

Entre collaborateur

4. Contexte

Nom de l'entreprise, organisme ou association ▶Op

Opskov

Chantier, atelier, service ▶

App Mobile Interne

DOSSIER PROFESSIONNEL^(DP)

Période d'exercice ▶ Du : **11/05/2024** au : **30/06/2025**

5. Informations complémentaires (*facultatif*)

En dehors de ce projet, j'ai aussi travaillé sur des projets plus simples (en solo), où je gérais les tâches avec Notion ou en TODO dans le code. Même sans méthode formelle, j'appliquais une logique d'organisation en étapes claires pour rester efficace.

DOSSIER PROFESSIONNEL^(DP)

Activité-type 2 Concevoir et développer une application sécurisée organisée en couches

Exemple n° 1 - YouthWallet

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

2.1 – Analyser les besoins et maquetter une application

📌 YouthWallet

Pour YouthWallet, l'objectif était de créer une application de gestion financière destinée aux adolescents, pilotée par les parents.

Les besoins ont été définis sous forme de user stories :

"En tant qu'ado, je veux visualiser mon solde et mes transactions."

"En tant que parent, je veux attribuer des tâches rémunérées et valider les demandes de retrait."

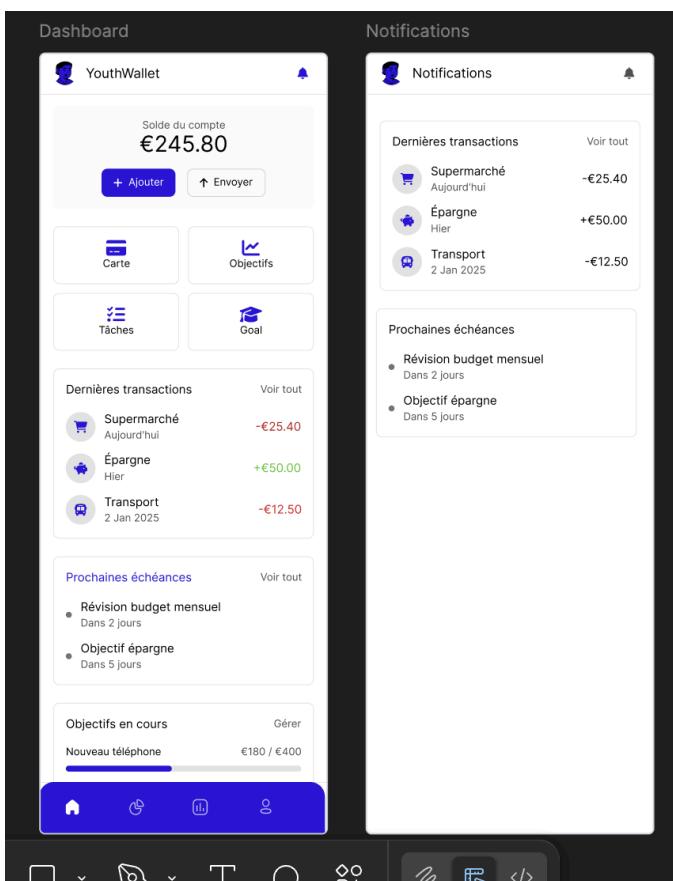
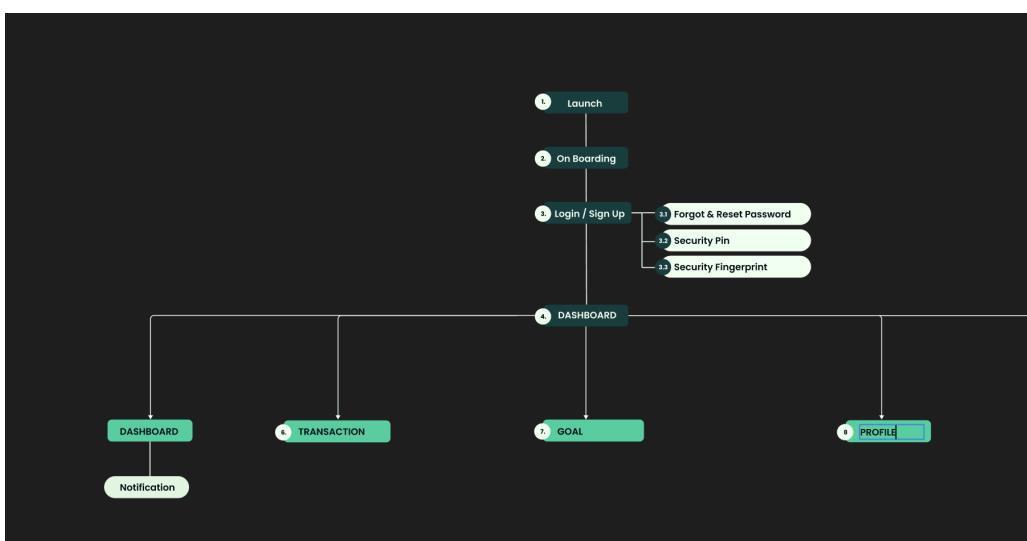
Les maquettes ont été réalisées sur Figma, avec une interface claire, inspirée des apps fintech (dashboard, solde, tâches, historique).

L'accent a été mis sur l'expérience utilisateur pour chaque rôle (ado/parent), avec une UI propre, intuitive et responsive.

Annexes : Voir ci-dessous

- Userstories
- maquette figmaa

DOSSIER PROFESSIONNEL (DP)



DOSSIER PROFESSIONNEL (DP)

2.2 – Définir l'architecture logicielle d'une application

- YouthWallet

Le backend repose sur NestJS avec une architecture modulaire :

- **controllers** → réception des requêtes HTTP,
- **services** → logique métier,
- **prisma** → couche d'accès aux données PostgreSQL.

Le frontend (Expo/React Native) est structuré par écrans, avec Zustand pour la gestion d'état, des modals animés, et une logique claire pour chaque interaction.

Le tout est conteneurisé avec Docker, et automatisé via GitHub Actions.

Annexes : Voir ci-dessous

- Arborescence du backend NESTJS/
- Controller / Service
- Docker Compose

The screenshot shows a terminal window with two panes. The left pane displays the directory structure of a NestJS project:

```
tree src
└── src
    ├── account
    ├── admin
    ├── auth
    ├── badge
    ├── category
    ├── common
    ├── config
    ├── goal
    ├── notification
    ├── task
    ├── tests
    ├── transaction
    └── user
        ├── app.controller.spec.ts
        ├── app.controller.ts
        ├── app.module.ts
        ├── app.service.spec.ts
        ├── app.service.ts
        ├── main.spec.ts
        └── main.ts
    └── test
```

The right pane shows a portion of the `app.controller.ts` file with some code snippets:

```
getHello(): string {
  return 'Hello World!';
}

// Endpoint GET "/check-env" pour vérifier que la variable d'environnement
// est bien configurée
@Get('/check-env')
checkEnv(): string {
  const dbHost = this.configService.get<string>('DB_HOST');
  const dbUser = this.configService.get<string>('DB_USER');
  const dbName = this.configService.get<string>('DB_NAME');
  return `DB_HOST from env: ${dbHost}, \nDB_USER from env: ${dbUser}, \nDB_NAME from env: ${dbName}`;
}

// Endpoint GET "/test-db" pour exécuter une requête SQL simple (SELECT * FROM users)
```

DOSSIER PROFESSIONNEL (DP)

```
@Injectable()
export class AppService implements OnModuleInit {
  constructor(private configService: ConfigService) {}

  onModuleInit() {
    // Exemple : lire et afficher la variable DB_HOST
    const dbHost = this.configService.get<string>('DB_HOST');
    console.log(`[AppService] DB_HOST from .env is: ${dbHost}`);
  }

  getHello(): string {
    return 'Hello World!';
  }
}
```

- ServiceNESTJS

```
services:
  > Run Service
  app:
    build:
      context: ./flightradar24-backend
      dockerfile: Dockerfile
      target: dev
    ports:
      - "3000:3000"
    volumes:
      - ./flightradar24-backend:/usr/src/app
    env_file:
      - ./flightradar24-backend/.env.docker
    environment:
      NODE_ENV=development
```

- DOCKERCOMPOSE

DOSSIER PROFESSIONNEL (DP)

2.3 – Concevoir et mettre en place une base de données relationnelle

- YouthWallet

J'ai modélisé la base PostgreSQL à partir d'un MCD puis MLD/MPD.

Les entités principales sont : **User, Transaction, Task, Goal, Notification**.

La structure respecte les règles de nommage, les relations sont bien typées, les données sensibles (mots de passe) sont sécurisées via hash.

J'ai également mis en place une base de tests pour simuler des cas réels, et un script de reset en cas d'incident.

ANNEXES :

- **MCD, MLD, MPD**
- **.PRISMA**
- **PGAMIN**

- **.PRISMA**

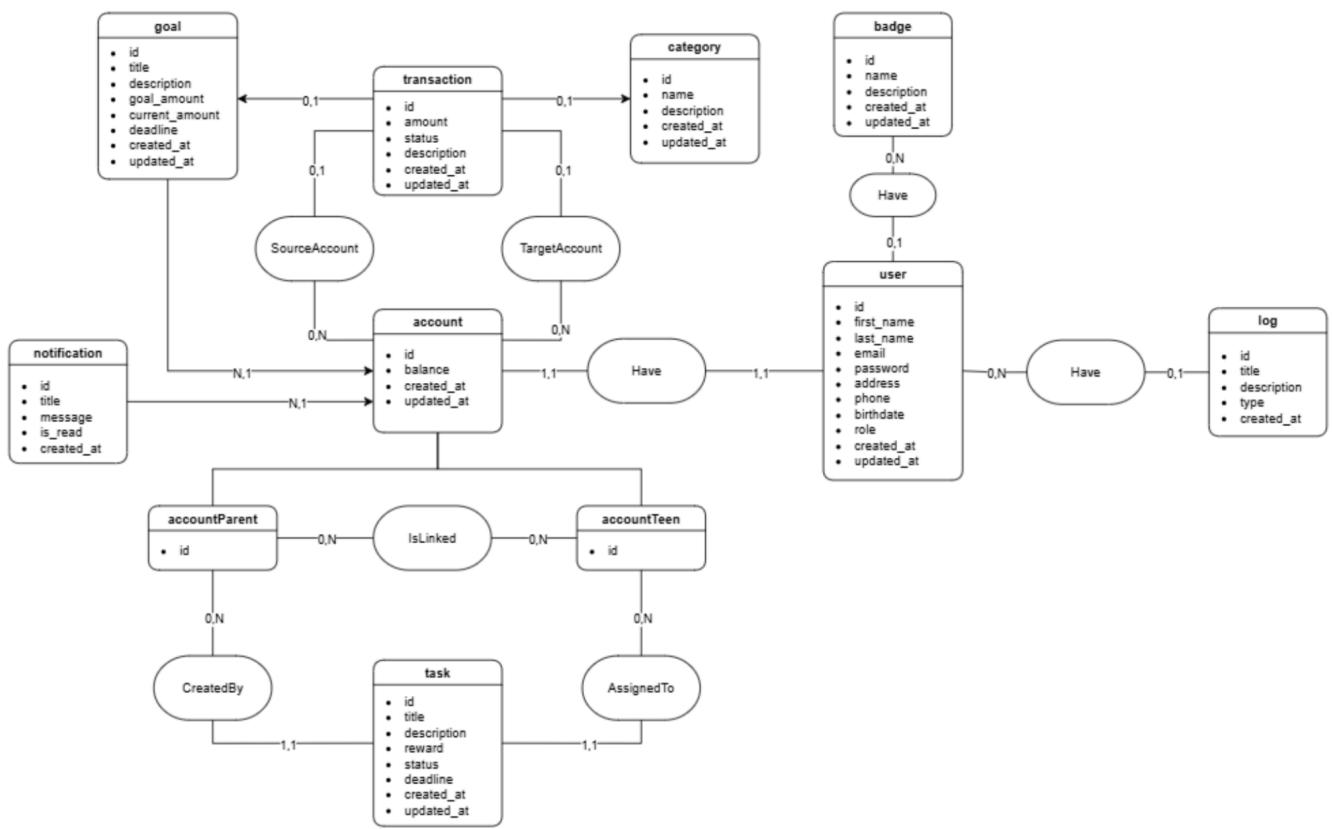
```
model Account
  id          Int      @id @default(autoincrement())
  balance     Decimal  @default(0.00)
  createdAt   DateTime @default(now()) @map("created_at")
  updatedAt   DateTime? @updatedAt @map("updated_at")

  userId      Int      @unique @map("user_id")
  user        User    @relation(fields: [userId], references: [id], onDelete: Cascade)
```

DOSSIER PROFESSIONNEL (DP)

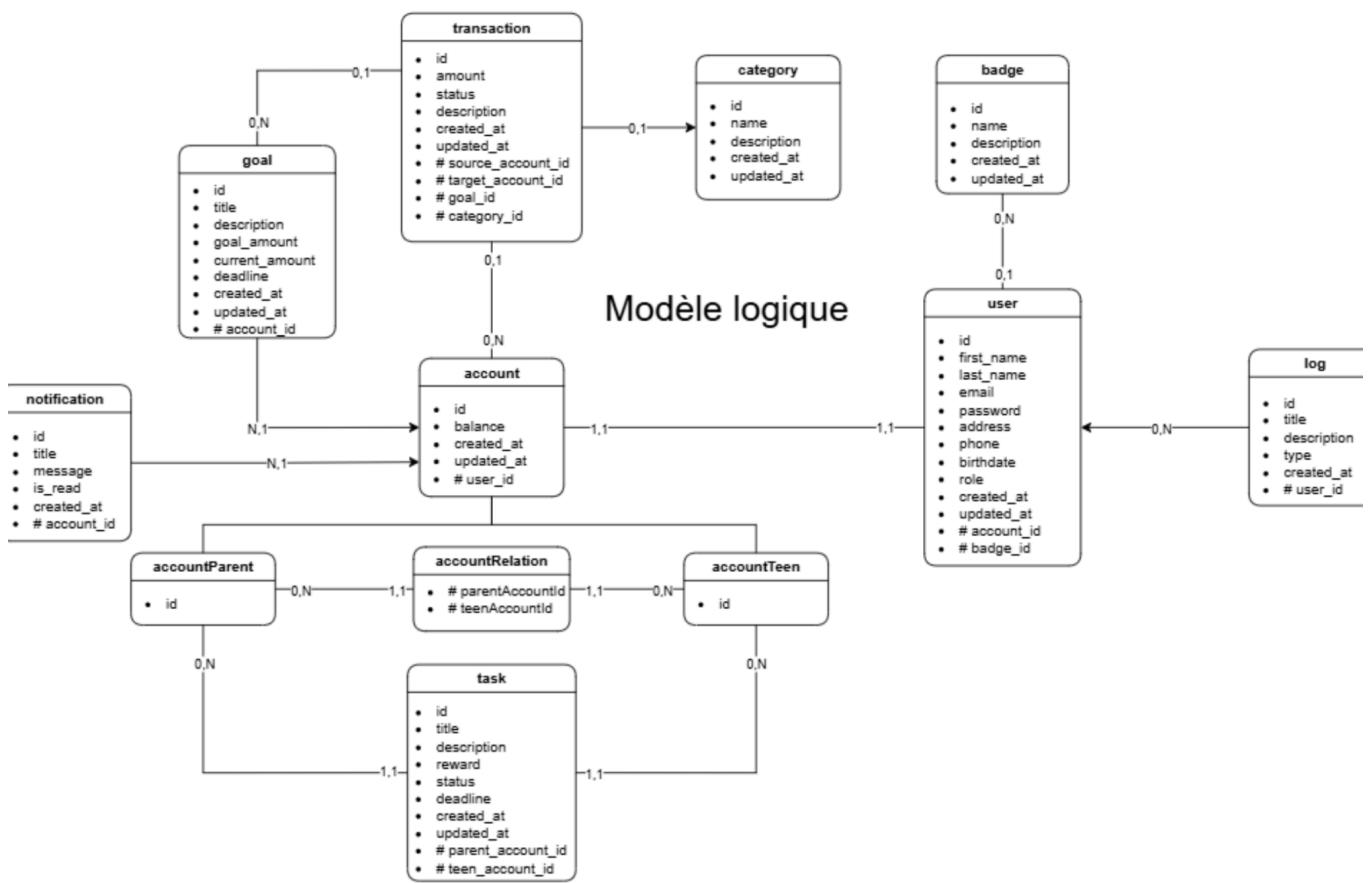
- MCD

Modèle conceptuel



DOSSIER PROFESSIONNEL (DP)

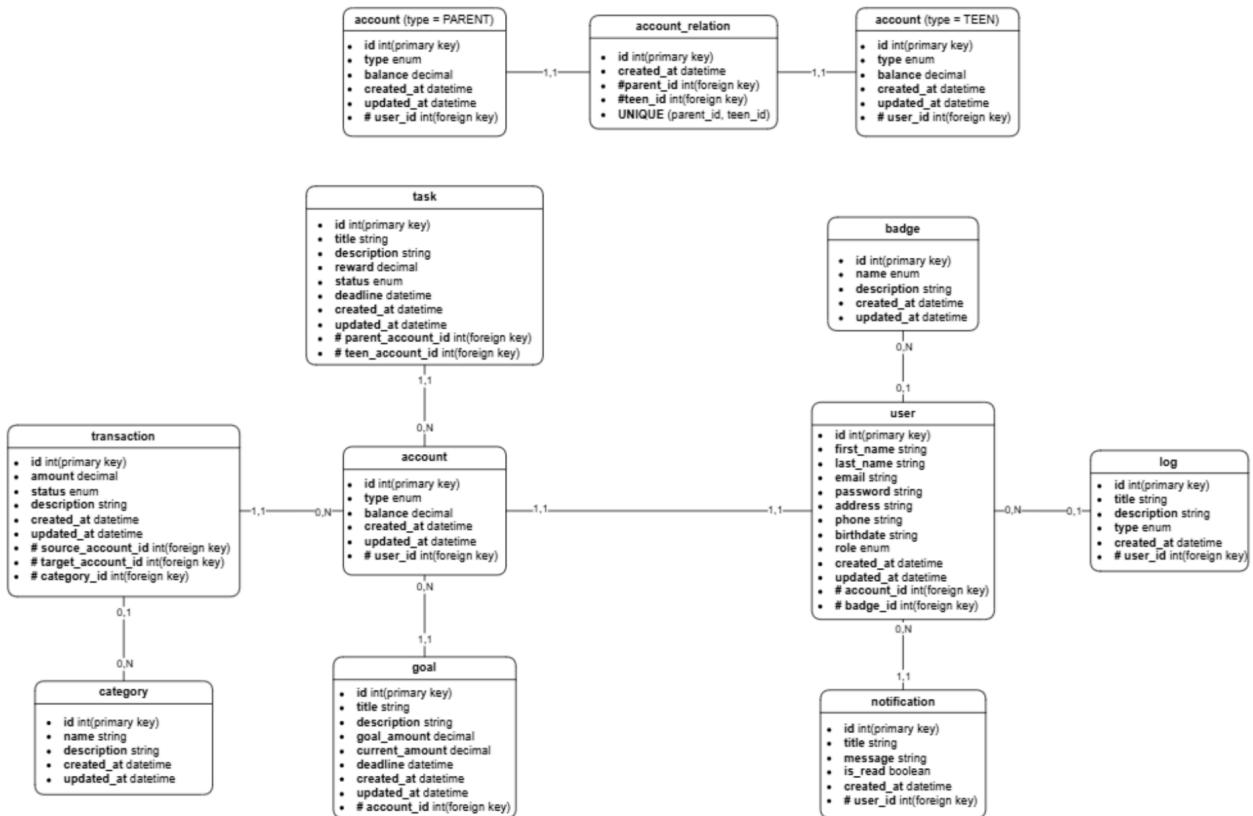
- MLD



DOSSIER PROFESSIONNEL (DP)

- MLD

Modèle Physique



En conclusion :

DOSSIER PROFESSIONNEL (DP)

2.4 – Développer des composants d'accès aux données SQL et NoSQL

YouthWallet (SQL avec Prisma)

Le backend utilise **Prisma ORM** pour simplifier les accès à PostgreSQL.

Chaque appel passe par un **service**, qui utilise Prisma pour effectuer des requêtes CRUD, tout en validant les rôles, permissions, et erreurs.

ANNEXES: Voir ci-dessous

- **Schema.prisma - Model de donnée**

```
model AccountParent {  
    id              Int          @id  
    account         Account      @relation(fields: [id],  
  
    // Relations N-N avec les comptes ado  
    linkedTeens     AccountRelation[]  
  
    // Tasks créées par ce parent  
    tasksCreated    Task[]       @relation("ParentTasks")  
}  
}
```

- **Service avec prisma ainsi que méthode associé pour find une transaction**

```
/**  
 * Récupère toutes les transactions liées à un compte.  
 * @param accountId L'ID du compte.  
 * @returns Une liste de transactions.  
 */  
async getTransactions(accountId: number) {  
    const account = await this.prisma.account.findUnique({  
        where: { id: accountId },  
    });  
  
    if (!account) {  
        throw new NotFoundException('Account not found');  
    }  
  
    return this.prisma.transaction.findMany({  
        where: {  
            account: {  
                id: accountId  
            }  
        }  
    });  
}
```

DOSSIER PROFESSIONNEL (DP)

- Prisma studio pour une vue d'ensemble sur notre BDD

Category	X	Account	●	Transaction	X	AccountParent	X		
		C	Filters	None	Fields	All	Showing	2 of 2	Add record
		currentAmount #		status ≡					
		70		IN_PROGRESS					
		400		IN_PROGRESS					
id		amount							
		search id...		search amount...					
<input checked="" type="checkbox"/>	45	100							

En Conclusion :

Avec YouthWallet, j'ai pu concevoir une application structurée, sécurisée, et adaptée à un besoin réel : aider les adolescents à mieux gérer leur argent avec l'accompagnement des parents.

J'ai défini les besoins à travers des user stories, puis conçu l'architecture logicielle autour de NestJS et Prisma.

La base de données relationnelle PostgreSQL a été pensée dès le départ avec un MCD/MLD cohérent, en respectant les bonnes pratiques de nommage et de sécurité.

Les accès aux données sont centralisés dans des services clairs, avec des contrôles et des retours adaptés.

C'est un projet complet qui m'a permis de mettre en œuvre toutes les compétences du Bloc 2.

2. Précisez les moyens utilisés :

DOSSIER PROFESSIONNEL (DP)

Technologies utilisées – YouthWallet

Conception :

- User stories, cas d'usage
- Maquettes Figma (UI mobile parent / ado)
- MCD / MLD / MPD

Backend :

- NestJS (TypeScript)
- Prisma ORM
- PostgreSQL
- REST API (modulaire)

Frontend :

- Expo / React Native
- Zustand (état global)
- NativeWind (UI responsive)
- fetch pour l'API

Outils & DevOps :

- Docker / docker-compose
- Git / GitHub Actions
- .env pour variables sensibles
- Tests manuels via Postman

3. Avec qui avez-vous travaillé ?

En collaboration avec :

- Chazaut Romain
- Augustin Yvon

4. Contexte

Nom de l'entreprise, organisme ou association ▶ **La PLATEFORME.**

Chantier, atelier, service ▶ **YouthWallet**

Période d'exercice ▶ Du : **06/02/2025** au : **01/08/2025**

DOSSIER PROFESSIONNEL^(DP)

5. Informations complémentaires (*facultatif*)

Aucune

DOSSIER PROFESSIONNEL^(DP)

Activité-type 2 Concevoir et développer une application sécurisée organisée en couches

Exemple n° 2 - Projet SkyRadar

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

2.4 – Développer des composants d'accès aux données SQL et NoSQL

- *YouthWallet montre la partie SQL avec Prisma/PostgreSQL*
- *FlightRADAR couvre la partie NoSQL avec Mongoose/MongoDB - DOCUMENT*

Les accès à la base NoSQL se font via Mongoose, avec des schémas bien définis (*UserSchema*, *NotificationSchema*, etc.).

La validation est intégrée dans les schémas, et chaque service utilise les méthodes Mongoose (*find*, *create*, *update*, etc.) pour interagir avec les données.

Des contrôles de permissions et une gestion fine des erreurs complètent le tout.

Utilisation de l'ORM Mongoose.

Annexes : voir ci-dessous

- **Schéma mongoose**

DOSSIER PROFESSIONNEL (DP)

```
You, 5 months ago | 1 author (You)
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose'; 132.4k (gzipped: 28k)
import { HydratedDocument } from 'mongoose'; 886.1k (gzipped: 236k)
import { ObjectType, Field, ID, registerEnumType } from '@nestjs/graphql';
import { Role } from 'src/auth/roles/role.enum';

export type UserDocument = HydratedDocument<User> & { _id: string };

registerEnumType(Role, { name: 'Role' });

You, 5 months ago | 1 author (You)
@ObjectType('UserType')
@Schema()
export class User {
  @Field(() => ID)
  _id: string;

  @Prop({ required: true })
  @Field()
  username: string;

  @Prop({ required: true })
  @Field()
  nom: string;

  @Prop({ required: true })
  @Field()
  prenom: string;
```

En conclusion :

J'ai utilisé le modèle NoSQL à travers Mongoose, qui repose sur un système de documents (MongoDB).

Les schémas sont bien définis avec des règles de validation intégrées, et les services utilisent les méthodes standard (find, create, etc.).

Cela m'a permis de valider la compétence NoSQL de manière concrète et structurée.

2. Précisez les moyens utilisés :

Les technologies utilisées sont les suivantes :

- **NestJS (backend)**
- **MongoDB**

DOSSIER PROFESSIONNEL (DP)

- **Mongoose (ORM)**

3. Avec qui avez-vous travaillé ?

J'ai travaillé seul sur ce projet dans le cadre de mon alternance

4. Contexte

Nom de l'entreprise, organisme ou association ➤ **OPSKOV**

Chantier, atelier, service ➤ **App en interne**

Période d'exercice ➤ Du : **Avril 2024** au : **Juin 2025**

5. Informations complémentaires (facultatif)

Pour ce projet, j'ai choisi MongoDB car le modèle de données était plus souple, notamment pour les notifications ou les logs utilisateurs.

*J'ai utilisé Mongoose comme ODM pour définir des schémas précis (**User**, **Notification**, etc.), intégrer la validation des données directement dans les modèles, et simplifier les opérations courantes (**findOne**, **updateOne**, etc.).*

L'approche NoSQL m'a permis d'être plus rapide dans l'itération des fonctionnalités sans me soucier des migrations lourdes.

DOSSIER PROFESSIONNEL^(DP)

Préparer le déploiement d'une application sécurisée

Activité-type 3

Exemple n° 1 - Youthwallet

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

3.1 – Préparer et exécuter les plans de tests d'une application

J'ai testé manuellement tous les endpoints critiques (authentification, transactions, objectifs, tâches) à l'aide de **Postman**.

Chaque scénario a été vérifié : statut HTTP, structure des données, et respect des permissions.

- J'ai testé tous les endpoints liés aux utilisateurs, transactions, objectifs, tâches, etc.
- Des vérifications systématiques ont été faites : statuts HTTP, formats des données, permissions.

Annexes: Voir ci-dessous

DOSSIER PROFESSIONNEL (DP)

- Test API via postman (Login avec tokenJWT)

The screenshot shows the Postman interface. On the left, the 'Collections' sidebar lists 'Youthwallet NOT VERIFIED' with sections for 'Account', 'Auth', 'Category', and others. In the main area, a 'POST Login' request is selected under the 'Auth' collection. The 'Body' tab shows a raw JSON payload:

```
1 {
2   "email": "user1@example.com",
3   "password": "password1234"
4 }
```

The 'Body' tab also shows the response received from the server:

```
1 {
2   "user": {
3     "id": 1,
4     "firstName": "John",
5     "lastName": "Doe",
6     "email": "user1@example.com",
7     "address": "123 Main St",
8     "phone": "0123456789",
9     "birthdate": "2005-10-05T00:00:00.000Z",
10    "forceResetPassword": false,
11    "role": "USER",
12    "createdAt": "2025-07-03T13:30:01.402Z",
13    "updatedAt": "2025-07-03T13:33:05.567Z",
14    "badgeId": 1
15  },
16  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlc2VvSwQ1OjE5InJvbGU1OjVUOVSTIwZW1haW1oIjJc2VvMUB1eGFtcGxLmNvbSIiMzpcnN0TmFtZSI6Ikpvag4iLCJzYXN0TmfzSI6IKRvZSIiM5hbWUiOjKb2huIERVZSIiMlhdCI6MtC1MzE4NTA0MSwiZXhwIjoxNzUzMtg4NjQxfQ.9qo8P42By4h-4Sd75dR0C3Zbf.vcGHHToeSP8RGw0"
```

- Test via JEST

```
describe('applyTransactionToBalance', () => {
  const txId = 42;

  it('moves balance from source to target', async () => {
    const source = { id: 1, balance: { toNumber: () => 200 } };
    const target = { id: 2, balance: { toNumber: () => 50 } };

    prisma.account.findUnique
      .mockResolvedValueOnce(source)
      .mockResolvedValueOnce(target);
    prisma.account.update.mockResolvedValue({});

    await service.applyTransactionToBalance(1, 2, null, 100, txId);

    expect(prisma.account.update).toHaveBeenCalledWith({
      where: { id: 1 },
      data: { balance: { decrement: 100 } },
    });
    expect(prisma.account.update).toHaveBeenCalledWith({
      where: { id: 2 },
      data: { balance: { increment: 100 } },
    });
    expect(prisma.transaction.update).toHaveBeenCalledWith({
      where: { id: txId },
      data: { status: TransactionStatus.COMPLETED },
    });
  });
});
```

DOSSIER PROFESSIONNEL (DP)

En conclusion :

J'ai utilisé une méthode de tests manuels rigoureuse pour assurer la fiabilité des fonctionnalités essentielles de mes projets.

2. Précisez les moyens utilisés :

Outils : Postman, application Expo (sur mobile), NestJS

3. Avec qui avez-vous travaillé ?

Collaborateur :

- Chazaud Romain
- Augustin YVON

4. Contexte

Nom de l'entreprise, organisme ou association ▶ **La PLATEFORME**

Chantier, atelier, service ▶ **YOUTHWALLET**

Période d'exercice ▶ Du : **06/01/2025** au : **01/08/2025**

5. Informations complémentaires (*facultatif*)

DOSSIER PROFESSIONNEL^(DP)

Préparer le déploiement d'une application

Activité-type 3

sécurisée

Exemple n° 2 - SAFEbase

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

3.2 – Préparer et documenter le déploiement d'une application

J'ai mis en place une infrastructure complète avec Docker :

- Container PostgreSQL
- Container backend Fastify
- Sauvegarde automatique dans un volume
- **.env** bien structuré
- **docker-compose.yml** propre

Annexes : A voir ci-dessous

- **DockerFile pour le frontEnd**

DOSSIER PROFESSIONNEL (DP)

```
1 FROM node:16
2
3 WORKDIR /frontend
4
5 COPY package*.json ./
6
7 RUN npm install
8
9 COPY . .
10
11 RUN npm run build
12
13 EXPOSE 5173
14
15 CMD ["npm", "run", "dev", "--", "--host", "0.0.0.0"]
```

- Extrait du docker-compose.yaml pour le backend

```
25      ▷ Run Service
26 backend:
27   build:
28     context: ./backend
29     container_name: backend_container
30     restart: always
31     ports:
32       - "3000:3000"
33     depends_on:
34       - db
35     environment:
36       - DB_HOST=db
37       - DB_PORT=5432
38       - DB_USER=postgres
39       - DB_PASSWORD=mysecretpassword
40       - DB_NAME=postgres
41     volumes:
42       - backend_src:/backend
43       - backend_backups:/backend/backups
44     healthcheck:
45       test: ["CMD", "curl", "-f", "http://localhost:3000"]
46       interval: 30s
47       timeout: 10s
48       retries: 5
49
```

- Extrait du [README.md](#)

DOSSIER PROFESSIONNEL (DP)

```
1 # Plateforme de Backup de Données avec PostgreSQL et pgAdmin
2
3 Ce projet configure un environnement Docker pour une plateforme de backup de données utilisant PostgreSQL
4 comme système de gestion de base de données et pgAdmin comme interface d'administration.
5 ## Prérequis
6
7 - Docker
8 - Docker Compose
9
10 ## Configuration
11
12 Le fichier `docker-compose.yml` configure PostgreSQL et pgAdmin. PostgreSQL est configuré sans base de
13 données prédefinie pour permettre l'importation de bases de données externes.
```

En Conclusion :

*Grâce à Docker et une documentation claire, j'ai rendu mes projets facilement déployables et
reproductibles sur d'autres machines.*

2. Précisez les moyens utilisés :

3. Avec qui avez-vous travaillé ?

4. Contexte

Nom de l'entreprise, organisme ou association ▶ **Cliquez ici pour taper du texte.**

Chantier, atelier, service ▶ **Cliquez ici pour taper du texte.**

Période d'exercice ▶ Du : **Cliquez ici** au : **Cliquez ici**

5. Informations complémentaires (facultatif)

DOSSIER PROFESSIONNEL (DP)

Activité-type 3 Préparer le déploiement d'une application sécurisé

Exemple n° 1 - Projet Flyradar

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

3.3 – Contribuer à la mise en production d'une démarche DevOps

Sur ce projet, j'ai :

- Travaillé en équipe avec validation de PR par un tech lead
- Utilisé **Git** avec branches **main, dev, feature/***
- Intégré une **CI** via GitHub Actions : lint + tests automatiques à chaque **push**

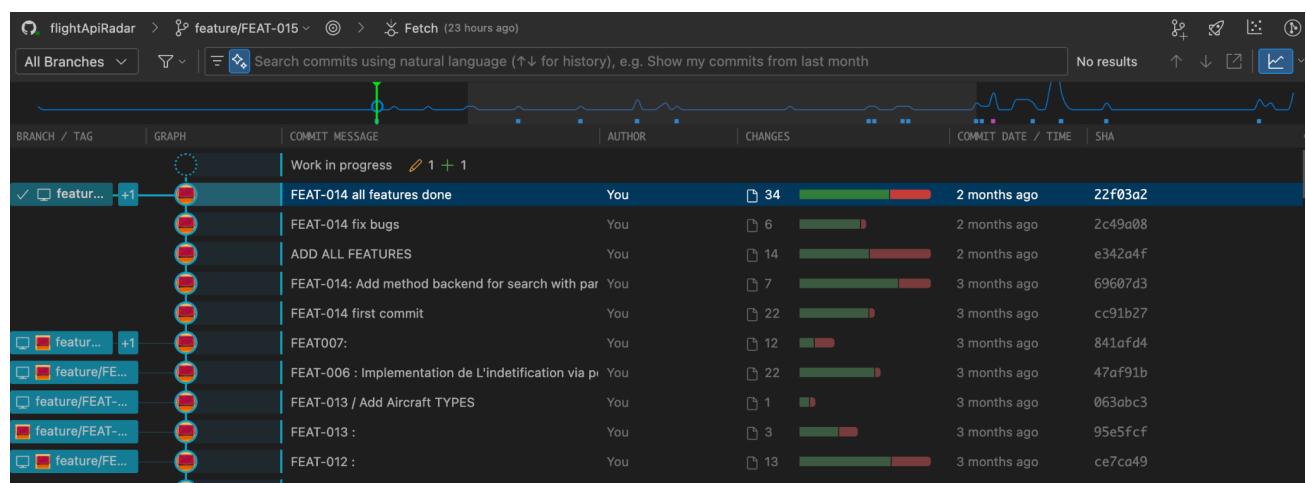
Annexes: A voir ci-dessous

- **Extrait d'une pipeline CI/CD via github actions**

```
jobs:  
  You, 6 months ago | 2 authors (You and one other)  
  lint-and-notify:  
    runs-on: ubuntu-latest  
    steps:  
      You, 6 months ago | 1 author (You)  
      - name: Checkout code  
        uses: actions/checkout@v4  
  
      You, 6 months ago | 1 author (You)  
      - name: Setup Node.js  
        uses: actions/setup-node@v3  
      You, 6 months ago | 1 author (You)  
      with:  
        node-version: '18'  
  
      You, 6 months ago | 1 author (You)  
      - name: Run ESLint  
        run: npm run lint  
  
      You, 6 months ago | 1 author (You)  
      - name: Send Gmail Notification  
        uses: davidd6/action-send-mail@v2  
      You, 6 months ago | 1 author (You)  
      with:  
        server_address: smtp.gmail.com      You, 6 months ago via PR #5 * next try  
        server_port: 465  
        username: ${{ secrets.EMAIL_USER }}  
        password: ${{ secrets.EMAIL_PASS }}  
        subject: Nouvelle Pull Request ouverte  
        body: Une nouvelle pull request a été ouverte et le linting a été effectué.  
        to: mehdi.romdhani1@gmail.com  
        from: GitHub Actions <${{ secrets.EMAIL_USER }}>
```

- **Extrait via Gitlens pour avoir un visuel sur l'ensemble des branches**

DOSSIER PROFESSIONNEL (DP)



- Extrait PR (pull-request)

The screenshot shows a list of pull requests on GitHub:

- 3 Open, 2 Closed
- Author, Label, Projects, Milestones, Reviews, Assignee dropdowns
- Three open pull requests:
 - #5 Feature/frontend - Review required
 - #4 Feature/backend - Review required
 - #3 Feature/cicd - Review required

Un suivi a été mis en place via **Trello** (kanban), avec des **dailys** réguliers pour assurer l'organisation du travail et éviter les blocages.

En Conclusion :

J'ai appliqué les bases d'une démarche DevOps : gestion de version, intégration continue et communication en équipe.

2. Précisez les moyens utilisés :

Git, Github, GitFORK

DOSSIER PROFESSIONNEL (DP)

3. Avec qui avez-vous travaillé ?

Entre Colloborateur

4. Contexte

Nom de l'entreprise, organisme ou association ▶ **OPSKOV**

Chantier, atelier, service ▶ **Flyradar**

Période d'exercice ▶ Du : **Avril 2024** au : **Juin 2025**

5. Informations complémentaires (*facultatif*)

Aucune

DOSSIER PROFESSIONNEL (DP)

Titres, diplômes, CQP, attestations de formation

(facultatif)

DOSSIER PROFESSIONNEL (DP)

Déclaration sur l'honneur

Je soussigné(e) Romdhani MEHDI,

déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je suis
l'auteur(e) des réalisations jointes.

Fait à Marseille

le **01/08/2025**

pour faire valoir ce que de droit.

Signature : MR

DOSSIER PROFESSIONNEL^(DP)

Documents illustrant la pratique professionnelle *(facultatif)*

Intitulé

Cliquez ici pour taper du texte.

DOSSIER PROFESSIONNEL^(DP)

ANNEXES

Toutes les annexes ont été placées à la suite du projet afin d'illustrer concrètement les compétences attendues, pour faciliter la compréhension de l'apprenant.